Addressing Concept Drift in Machine Learning-Based Monitoring of Manufacturing Processes

Konzeptdrift bei der auf maschinellem Lernen basierenden Überwachung von Fertigungsprozessen
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von Nicolas Jourdan aus Frankfurt am Main
Tag der Einreichung: 28. Mai 2024, Tag der Prüfung: 27. August 2024

1. Gutachten: Prof. Dr.-Ing. Joachim Metternich
2. Gutachten: Prof. Dr.-Ing. Uwe Klingauf
Darmstadt, Technische Universität Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Mechanical Engineering
Department

Institute for Production
Management, Technology
and Machine Tools

Addressing Concept Drift in Machine Learning-Based Monitoring of Manufacturing Processes
Konzeptdrift bei der auf maschinellem Lernen basierenden Überwachung von Fertigungsprozessen

Accepted doctoral thesis by Nicolas Jourdan

Date of submission: 28. Mai 2024
Date of thesis defense: 27. August 2024

Darmstadt, Technische Universität Darmstadt

Acknowledgements

This thesis summarizes the research that I conducted during my time as a research associate at the Institute for Production Management, Technology and Machine Tools (PTW) at the Technical University of Darmstadt. Throughout this journey, I have had the privilege of encountering numerous individuals who provided valuable support and guidance. I would like to take this opportunity to convey my sincere appreciation to them.

I want to express my gratitude to my supervisor, Prof. Dr.-Ing. Joachim Metternich, for his trust, guidance, and productive discussions as well as the freedom in choosing and pursuing my research topic. Next, I want to thank Prof. Dr.-Ing. Uwe Klingauf for reviewing this thesis and the valuable discussions in the doctoral seminar of the Insitute for Flight Systems and Automatic Control (FSR).

Within PTW, I want to thank all my colleagues for their support and for making the time enjoyable. In particular, I would like to mention some colleagues and friends: Beatriz Cassoli for her feedback on my research ideas and her help in revising and improving this document; Tobias Biegel for his support in shaping the research ideas and the focus group 'Data Science and Artificial Intelligence in Manufacturing' at PTW; and Florian Mitschke for his endurance during the seemingly endless shared hours of assembling the utilized CNC machines from scratch and spending significant time listening to the agonizing sounds of milling with worn cutters. In addition, I thank Christoph Schwarz, Jan Hofmann, Willi Wünschel and Alexander Fertig for their support with the machines, the measuring equipment, and the experiments.

On a personal note, I want to thank my whole family, especially my mother, Birgit, my father, Frank, and my stepfather, Martin, for their constant support, trust, and encouragement throughout this academic endeavor, and indeed, throughout my life.

Last, but, in this instance, particularly not least, I want to acknowledge my wife, Sara. I am quite certain that the completion of this thesis — in terms of both time and quality, and possibly its completion at all — would not have been possible without your love, your patience and endurance while proofreading and listening to practice presentations, your encouragement, and our AIFA sessions, during which significant parts of this thesis were developed. For that, I thank you and I am deeply grateful to have you by my side.

Darmstadt, September 2024 Nicolas Jourdan

Abstract

Machine learning (ML) has significantly contributed to the development of advanced process and condition monitoring systems in manufacturing, enabling real-time monitoring and analysis of equipment and processes to detect deviations from normal operations and predict potential failures. As ML applications transcend from academic research to real-world usage, questions regarding their continuous reliability on the shopfloor arise. The training dataset of an ML model only captures a snapshot of the manufacturing process in time. After model deployment, the environment will encounter changes such as wear, aging or defective sensors as well as changes in factory layout and machine placement that are not captured in the training dataset, a scenario referred to as *concept drift*, which is often neglected in academic studies. Concept drift leads to performance degradation of an ML model which is not obvious to machine or plant operators, potentially causing unnoticed downstream issues unless addressed properly. Recent process models for structuring industrial ML projects such as CRISP-ML(Q) have started to consider this issue, but do not offer guidance on implementing mechanisms to detect or counteract concept drift. Motivated by this gap, this thesis analyzes methods for the detection of concept drift in the context of ML applications for process and condition monitoring in manufacturing, aiming to improve the application's reliability and acceptance. First, a literature review and expert interviews are conducted to gain insights on concept drift handling in manufacturing research and practice. Consequently, a framework is derived that concretizes the monitoring phase of the CRISP-ML(Q) process model for the target domain by outlining active and passive concept drift detection strategies accompanied by decision criteria for their respective usage. Existing concept drift detection strategies often rely on two-sample tests assuming independent and identically distributed (i.i.d.) data, an assumption that proves invalid in the targeted use cases. Thus, a refinement method called Localized Reference Drift Detection (LRDD) is proposed as a preprocessing step for two-sample testing. The developed framework as well as the proposed LRDD method are validated in terms of applicability and performance through three case studies. In the first case study, a tool condition monitoring scenario is investigated. It is shown how variations in the operating conditions degrade the model performance and how the framework can reliably detect drifts, employing LRDD for active concept drift detection. In the second case study, the use case of process monitoring in milling is analyzed. It is shown that concept drift is present in the dataset due to the aging of the machine components between experiment runs. The drift is reliably detected through the configured active concept drift detection. In the third case study, a condition monitoring use case is implemented within a pigment production line at a company. Within the case study, passive concept drift detection is implemented as the data distributions vary strongly between production batches. It is shown that the passive concept drift detection paired with automatic retraining enables effective condition monitoring of a critical component within the production line. Overall, this thesis provides solution approaches to dealing with concept drift in the manufacturing domain. The proposed methods and decision criteria can either be directly applied to existing use cases or serve as inspiration and guidance for use cases beyond the scope of the case studies.

Zusammenfassung

Maschinelles Lernen (ML) hat wesentlich zur Entwicklung fortschrittlicher Prozess- und Zustandsüberwachungssysteme in der Fertigung beigetragen. Sie ermöglichen die Überwachung und Analyse von Anlagen und Prozessen in Echtzeit, um Abweichungen zu erkennen und mögliche Ausfälle vorherzusagen. Da ML-Anwendungen zunehmend von der Forschung in die Praxis übergehen, stellt sich die Frage nach ihrer dauerhaften Zuverlässigkeit in der Produktion. Der Trainingsdatensatz eines ML-Modells erfasst nur eine Momentaufnahme des Fertigungsprozesses. Während der Nutzungsphase des ML-Modells kommt es in der Umgebung zu Veränderungen, wie z.B. Werkzeug- und Maschinenverschleiß, alternden oder defekten Sensoren sowie Änderungen im Fabriklayout und in der Maschinenaufstellung, die im Trainingsdatensatz nicht erfasst sind. Dieses Szenario wird als Konzeptdrift bezeichnet und in akademischen Studien oft vernachlässigt. Konzeptdrift führt zu einer Leistungsverschlechterung des ML-Modells, die für die Maschinen- oder Anlagenbediener/-innen nicht offensichtlich ist und zu nachgelagerten Problemen führen kann, wenn sie nicht richtig adressiert wird. Prozessmodelle für industrielle ML-Projekte wie CRISP-ML(Q) berücksichtigen dieses Problem, bieten aber keine konkreten Richtlinien für die Implementierung von Mechanismen, um Konzeptdrift zu erkennen oder entgegenzuwirken. Motiviert durch diese Lücke, analysiert diese Dissertation Methoden zur Erkennung von Konzeptdrift im Kontext von ML-Anwendungen für die Prozess- und Zustandsüberwachung in der Fertigung, mit dem Ziel, die Zuverlässigkeit und Akzeptanz der ML-Anwendungen zu verbessern. Zunächst werden eine Literaturrecherche und Experteninterviews durchgeführt, um Erkenntnisse über den Umgang mit Konzeptdrift in Forschung und Praxis zu gewinnen. Daraufhin wird ein Framework abgeleitet, das die Überwachungsphase des CRISP-ML(Q)-Prozessmodells für die Zieldomäne konkretisiert, indem aktive und passive Strategien zur Erkennung von Konzeptdrift zusammen mit Entscheidungskriterien skizziert werden. Bestehende Strategien zur Erkennung von Konzeptdrift beruhen häufig auf Zwei-Stichproben-Tests unter der Annahme unabhängiger und identisch verteilter Daten, eine Annahme, die sich in den betrachteten Anwendungsfällen als ungültig erweist. Daher wird eine Vorverarbeitungsmethode namens Localized Reference Drift Detection (LRDD) für den Vergleichsdatensatz vorgeschlagen. Das entwickelte Framework und die LRDD-Methode werden anhand von drei Fallstudien auf ihre Anwendbarkeit und Leistungsfähigkeit hin überprüft. In der ersten Fallstudie wird ein Szenario zur Überwachung des Werkzeugzustands untersucht. Es wird gezeigt, wie Schwankungen in den Betriebsbedingungen die Leistung des ML-Modells beeinträchtigen und wie das Framework Drifts zuverlässig erkennen kann, wobei LRDD zur aktiven Erkennung von Konzeptdrift eingesetzt wird. In der zweiten Fallstudie wird der Anwendungsfall der Prozessüberwachung beim Fräsen analysiert. Es wird gezeigt, dass Konzeptdrift aufgrund der Alterung der Maschinenkomponenten zwischen den Versuchsläufen vorhanden ist. Die Drift wird durch aktive Konzeptdrift-Erkennung zuverlässig erkannt. In der dritten Fallstudie wird ein Anwendungsfall für die Zustandsüberwachung in einer Pigmentproduktionslinie in einem Unternehmen implementiert. In der Fallstudie wird eine passive Konzeptdrift-Erkennung implementiert. Es wird gezeigt, dass die passive Konzeptdrift-Erkennung in Verbindung mit einer automatischen Neutrainierung eine effektive Zustandsüberwachung ermöglicht. Insgesamt bietet diese Dissertation Lösungsansätze für den Umgang mit Konzeptdrift im Fertigungsbereich. Die vorgeschlagenen Methoden und Entscheidungskriterien können entweder direkt auf bestehende Anwendungsfälle angewendet werden oder als Inspiration und Anleitung für Anwendungsfälle dienen, die über den Rahmen der Fallstudien hinausgehen.

Contents

List of Figures xiii

List of Tables xv

List of Symbols xvii

List of Abbreviations xix

1. Introduction 1

1.1. Problem statement and research questions . . . 2
1.2. Thesis structure . . . 3

2. Fundamentals 5

2.1. Machine learning paradigms and evaluation . . . 5
2.1.1. Learning paradigms and notation . . . 6
2.1.2. Performance evaluation . . . 8
2.2. Machine learning algorithms . . . 13
2.3. Practical aspects of developing machine learning applications . . . 30
2.3.1. Data distribution shifts . . . 30
2.3.2. Process models and MLOps . . . 34
2.4. Two-sample hypothesis testing . . . 37
2.5. Machine learning in manufacturing . . . 39
2.5.1. Condition monitoring . . . 40
2.5.2. Process monitoring and predictive quality . . . 42
2.6. Time series feature extraction . . . 43
2.7. Summary . . . 46

3. Challenges and required action 47

3.1. Literature on concept drift in machine learning for manufacturing . . . 47
3.1.1. Condition monitoring . . . 48
3.1.2. Predictive quality and process monitoring . . . 49
3.1.3. Other industrial use cases . . . 50
3.1.4. Summary . . . 51
3.2. Practitioner views on concept drift in machine learning for manufacturing . . . 52
3.2.1. Interview methodology and participants . . . 52
3.2.2. Interview results . . . 53
3.3. Summary . . . 55

4. Framework development 59
4.1. Research design . . . 59
4.2. Framework overview . . . 61
4.3. Active drift detection . . . 63
4.3.1. Systematic literature review . . . 63
4.3.2. Challenges regarding hypothesis testing . . . 72
4.4. Passive drift detection . . . 75
4.5. Implementation . . . 77
4.6. Summary . . . 78

5. Case studies and validation 79
5.1. Case study 1: Tool condition monitoring in a CNC milling process . . . 79
5.1.1. Use case description and machine learning application design . . . 80
5.1.2. Data collection with concept drift . . . 84
5.1.3. Concept drift detection . . . 86
5.1.4. Interim summary . . . 91
5.2. Case study 2: Predictive quality and process monitoring in a CNC milling process . . . 92
5.2.1. Use case description and machine learning application design . . . 92
5.2.2. Data collection with concept drift . . . 94
5.2.3. Concept drift detection . . . 95
5.2.4. Interim summary . . . 97
5.3. Case study 3: Condition monitoring in a pigment sieving process . . . 97
5.3.1. Use case description . . . 98
5.3.2. Experiment setup and data exploration . . . 99
5.3.3. Machine learning application design and concept drift detection . . . 101
5.3.4. Interim summary . . . 103
5.4. Summary . . . 103

6. Conclusion and outlook 105

Bibliography 109

A. Supplementary material for Chapter 4 125
A.1. Literature review studies . . . 125

B. Supplementary material for Chapter 5 129
B.1. Scalar features . . . 129
B.2. Case study 1: Tool condition monitoring in a CNC milling process . . . 130
B.3. Case study 2: Predictive quality and process monitoring in a CNC milling process . . . 137
B.4. Case study 3: Condition monitoring in a pigment sieving process . . . 140

C. Comparison of neural network uncertainty estimation methods for drift detection 143
C.1. Methodology and experiments . . . 143
C.2. Uncertainty estimation methods . . . 144
C.3. Drift detector . . . 145
C.4. Datasets . . . 145
C.5. Metrics and results . . . 145
C.6. Conclusion . . . 147

C.7. Hyperparameters and architectures 147

List of Figures

1.1. Conceptual visualization of ML model performance after deployment 2
1.2. Thesis structure. 4

2.1. Confusion matrix of a binary classifier. 8
2.2. Exemplary visualization of threshold-based classification. 10
2.3. Example for Receiver Operating Characteristic. 10
2.4. Training and testing error as a function of the model complexity. 12
2.5. Model validation and evaluation using fixed training, validation and testing splits. 13
2.6. Model validation and using 4-fold cross validation. 13
2.7. Visualization of the k-nearest neighbor algorithm. 15
2.8. Visualization of the decision tree algorithm. 17
2.9. Visualization of the random forest algorithm. 19
2.10. Visualization of a SVM for binary classification and OC-SVM for anomaly detection. 20
2.11. Visualization of a multilayer perceptron with two layers. 23
2.12. Convolutional neural network architecture for classification (LeNet). 26
2.13. Examples of a convolution operation. 27
2.14. Dimensionality reduction techniques applied to the MNIST dataset. 28
2.15. Manufacturing process and ML model behavior over time. 31
2.16. Categorization of distribution shifts according to the affected part of the joint data distribution. 32
2.17. Categorization of distribution shifts according to their behavior over time. 33
2.18. The phases of the Cross Industry Standard Process for Data Mining (CRISP-DM). 35
2.19. Differences between CRISP-DM and CRISP-ML(Q) process model phases. 36
2.20. Application areas of AI in manufacturing-related tasks. 39
2.21. Categorization of maintenance types according to DIN EN 13306. 40
2.22. Open System Architecture for Condition-Based Maintenance (OSA-CBM). 41
2.23. General approach to predictive quality applications. 43
2.24. Time series feature types used for machine learning in the case studies. 44

4.1. Design science research approach used in this thesis. 60
4.2. Proposed framework for monitoring of deployed ML models. 61
4.3. Framework for active drift detection. 65
4.4. Time series contextualization and partitioning for feature extraction. 66
4.5. Overview and categorization of the major options for data modeling. 67
4.6. Data modeling options for unsupervised active drift detection visualized in a predictive quality example application. 68
4.7. Feature extraction from multivariate time series data. 68
4.8. Spectrogram / Scalogram extraction from multivariate time series data. 69

4.9. Synthetic condition monitoring dataset with reference and live window data points and the respective densities. 73
4.10. Localized reference drift detection for two exemplary scenarios. 74
4.11. Visualization of transient anomalies vs persistent anomalies. 76
4.12. Factors of influence on the manufacturing process visualized as an Ishikawa diagram. 76

5.1. Stepcraft M.500 3-axis CNC milling testbed. 80
5.2. Milling tool in new and worn states. 82
5.3. Synchronized accelerometer and NC control data of the tool condition monitoring case study. . 84
5.4. t-SNE visualization of the experiments conducted in the tool condition monitoring case study. . 86
5.5. Schematic representation of the part geometry and quality-relevant features in the predictive quality case study. 92
5.6. PCA and t-SNE visualizations of the experiments in the predictive quality case study. 95
5.7. Components of the sieving machine used in the pigment production case study. 98
5.8. Examples of sieve mesh tears in the pigment production case study. 99
5.9. Spectrograms of two full experiment runs in the piegment sieving case study. 100
5.10. PCA and t-SNE visualizations of pre-trial experiments in the pigment production case study. . . 101
5.11. Visualization of the data partitioning for each experiment run. 102

B.1. Example of the tool assignment on the aluminum plates in the tool condition monitoring case study. 136

C.1. Approach to uncertainty drift detection. 144
C.2. Calibration of the employed uncertainty estimation methods. 147

List of Tables

3.1. Expert interview partners and corresponding industry roles. 52

4.1. Overview of strategies for concept drift detection with submethods and primary requirements. 77

5.1. Overview of case studies and their respective configuration. 79
5.2. Properties of the Bosch BMA280 accelerometer used in the Bosch CISS sensor for the tool condition monitoring case study. 81
5.3. Tool condition monitoring case study cutting parameters. 81
5.4. ML model performance on the reference condition test dataset of the tool condition monitoring case study. 83
5.5. Reference and drift conditions in the tool condition monitoring case study. 85
5.6. Performance degradation of the tool condition monitoring application with respect to the introduced process changes. 85
5.7. Drift detection performance in scenario b.2 of the tool condition monitoring case study. 89
5.8. Drift detection performance in scenario c.1 of the tool condition monitoring case study. 90
5.9. Drift detection performance in scenario c.2 of the tool condition monitoring case study. 91
5.10. Available NC controller signals in the predictive quality case study. All signals are recorded at 500 Hz. 93
5.11. ML model performance on the test dataset of the predictive quality and process monitoring case study. 94
5.12. Performance degradation of the predictive quality and process monitoring application concerning the recording dates of the data used for testing. 94
5.13. Drift detection performance in the predictive quality and process monitoring case study. 97
5.14. Properties of the accelerometer used in the pigment production case study. 99
5.15. Results of model selection in pigment production. 102

A.1. List of studies analyzed within the systematic literature review. 125

B.1. Statistical and spectral features used for shallow models. 129
B.2. Final hyperparameter settings in the tool condition monitoring case study. 130
B.3. CNN architecture used for the tool condition monitoring case study. 131
B.4. Drift detection performance of LRDD for different choices of the number of neighbors k and two-sample tests. 131
B.5. Drift detection precision in scenario b.2 of the tool condition monitoring case study. 132
B.6. Drift detection recall in scenario b.2 of the tool condition monitoring case study. 132
B.7. Drift detection precision in scenario c.1 of the tool condition monitoring case study. 133
B.8. Drift detection recall in scenario c.1 of the tool condition monitoring case study. 134
B.9. Drift detection precision in scenario c.2 of the tool condition monitoring case study. 135
B.10. Drift detection recall in scenario c.2 of the tool condition monitoring case study. 135

B.11. Selected features in the predictive quality and process monitoring case study. 137
B.12. Final hyperparameter settings in the predictive quality and process monitoring case study. . . 138
B.13. Drift detection precision in the predictive quality and process monitoring case study. 139
B.14. Drift detection recall in the predictive quality and process monitoring case study. 139
B.15. Final hyperparameter settings in pigment sieving case study. 140
B.16. CAE architecture used in pigment production case study. 141

C.1. Performance comparison of uncertainty estimation methods for drift detection 146
C.2. Overview of model architectures. 148
C.3. Sensitivity values for ADWIN detector. 148

List of Symbols

General notation

Scalars	Lowercase letters or symbols, e.g., a, b, δ, γ
Vectors	Bold lowercase letters or symbols, e.g., $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{\delta}, \boldsymbol{\gamma}$
Matrices	Uppercase letters or symbols, e.g., A, B, Δ, Γ
Tensors	Bold uppercase letters or symbols, e.g., $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\Delta}, \boldsymbol{\Gamma}$

General mathematical symbols

$\hat{a}$	Estimate of a
$\tilde{a}$	Transformation of a
$\cdot$	Arbitrary argument of a function or operator, e.g., $f(\cdot)$
$\mathbb{I}(\cdot)$	Indicator function
$\lVert \cdot \rVert_2$	Euclidian norm
$\lfloor \cdot \rfloor$	Floor function
$\nabla_{\boldsymbol{w}} f(\cdot)$	The gradient of a function f with respect to $\boldsymbol{w}$

Machine learning

$x, \boldsymbol{x}, X, \boldsymbol{X}$	Input data
$y, \boldsymbol{y}, Y, \boldsymbol{Y}$	Target values / labels
$\hat{y}, \hat{\boldsymbol{y}}, \hat{Y}, \hat{\boldsymbol{Y}}$	Predictions for target values
$\mathcal{D}$	Dataset
$\boldsymbol{\theta}$	Trainable parameters of a model
$\mathcal{L}(y, \hat{y})$	Loss function with respect to the target value y and the prediction $\hat{y}$
$k(x_1, x_2)$	Kernel function that measures similarity between any pair of inputs x_1, x_2
δ	Confidence threshold
$p(y = j \mid \boldsymbol{x})$	Estimated conditional probability that the data point $\boldsymbol{x}$ belongs to class j
$h(\boldsymbol{x})$	Continuous outlier score of the data point $\boldsymbol{x}$

Probabilities and distributions

P, Q	Probability distributions of random variables
$\mathbb{E}(\cdot)$	Expected value of the argument
μ	Mean
σ	Variance
Σ	Covariance matrix
α	Significance level in hypothesis tests

List of Abbreviations

ADAM	Adaptive Moments Estimation.
ADC	Analog-to-Digital Converter.
ADWIN	Adaptive Windowing.
AE	Auto-Encoder.
AI	Artificial Intelligence.
ANN	Artificial Neural Network.
ASH	Activation Shaping.
AUROC	Area Under the Receiver Operating Characteristic.
BCE	Binary Cross-Entropy.
BMA	Bayesian Model Averaging.
CAE	Convolutional Auto-Encoder.
CAM	Computer-aided Manufacturing.
CbM	Condition-based Maintenance.
CDBD	Confidence Distribution Batch Detection.
CE	Cross-Entropy.
CI/CD	Continuous Integration / Continuous Delivery.
CNC	Computerized Numerical Control.
CNN	Convolutional Neural Network.
CPS	Cyber-Physical Systems.
CRISP-DM	Cross-Industry Standard Process for Data Mining.
CRISP-ML(Q)	Cross-Industry Standard Process Model for the Development of Machine Learning Applications with Quality Assurance Methodology.
CV	Cross-Validation.
CWT	Continuous Wavelet Transformation.
DFT	Discrete Fourier Transformation.
DOI	Digital Object Identifier.
DSR	Design Science Research.
DT	Decision Tree.
ECE	Expected Calibration Error.
EDF	Empirical Distribution Function.
FN	False Negative.

FP	False Positive.
FPR	False Positive Rate.
GD	Gradient Descent.
HHT	Hilbert-Huang Transformation.
i.i.d.	Independent and Identically Distributed.
ID3	Iterative Dichotomiser 3.
IF	Isolation Forest.
IG	Information Gain.
IoT	Internet of Things.
IP	Interview Partner.
kNN	k-Nearest Neighbor.
KPI	Key Performance Indicator.
KS	Kolmogorov-Smirnov.
LFR	Linear Four Rates.
LOF	Local Outlier Factor.
LR	Learning Rate.
LRD	Local Reachability Density.
LRDD	Localized Reference Drift Detection.
LSTM	Long Short-Term Memory.
MAP	Maximum a Posteriori.
MCC	Matthew's Correlation Coefficient.
MCD	Monte Carlo Dropout.
MD3	Margin Density Drift Detection.
MEMS	Micro-Electro-Mechanical System.
MES	Manufacturing Execution System.
ML	Machine Learning.
MLOps	Machine Learning Operations.
MLP	Multilayer Perceptron.
MMD	Maximum Mean Discrepancy.
MSE	Mean Squared Error.
NC	Numerical Control.
NN	Neural Network.
OC-SVM	One-Class Support Vector Machine.
OOD	Out-of-Distribution.
OPC UA	Open Plattform Communications Unified Architecture.
OSA-CBM	Open System Architecture for Condition-Based Maintenance.

PCA	Principal Component Analysis.
PHM	Prognostics and Health Management.
PLC	Programmable Logic Controller.
QA	Quality Assurance.
QC	Quality Control.
RBF	Radial Basis Function.
ReLU	Rectified Linear Unit.
RF	Random Forest.
RGB	Red Green Blue.
RMS	Root Mean Square.
RO	Research Objective.
ROC	Receiver Operating Characteristic.
RUL	Remaining Useful Lifetime.
SGD	Stochastic Gradient Descent.
SHAP	Shapley Additive Explanations.
SME	Small and Medium-sized Enterprise.
SPC	Statistical Process Control.
SRQ	Sub Research Question.
STFT	Short-Time Fourier Transformation.
SVC	Support Vector Classifier.
SVM	Support Vector Machine.
SWA	Stochastic Weight Averaging.
SWAG	Stochastic Weight Averaging Gaussian.
t-SNE	t-Distributed Stochastic Neighbor Embedding.
TN	True Negative.
TP	True Positive.
TPR	True Positive Rate.
UDD	Uncertainty Drift Detection.
VB	Verschleißmarkenbreite.

1. Introduction

Artificial Intelligence (AI) in general and Machine Learning (ML) in particular have emerged as key technologies of the 21st century [Chui23]. Independent of the recent introduction of general purpose AI tools such as ChatGPT and DALL·E which leverage generative AI, the adoption of ML technology within manufacturing environments has seen significant advancements in the last decade [Chui21; Manu23]. Due to the capital-intensive nature of manufacturing operations, the potential benefits of ML in this sector are perceived as particularly high [Pric17; Mett21]. Ongoing digitization and the deployment of advanced technologies in the context of Internet of Things (IoT) and Industry 4.0 are transforming manufacturing lines of today into Cyber-Physical Systems (CPS), which generate large amounts of data during operation, essential for the training of ML models [Cass22]. Companies are progressively integrating ML into their manufacturing processes, acknowledging its operational advantages. A 2021 McKinsey & Company study [Chui21] indicates that among companies operating ML applications, 87% have reported cost benefits from their deployment.
ML-related research has significantly contributed to the development of advanced process and condition monitoring systems in manufacturing, enabling real-time monitoring and analysis of equipment and processes to detect deviations from normal operation and predict potential failures [Wues16; Chui21]. In the quality management domain, ML models can analyze sensor and machine control data to detect defects early in the manufacturing process, thus reducing or preventing the production of scrap [Jour21b; Fert23].
Companies face a strong upfront investment to build up the talent, the knowledge and the infrastructure required to conduct ML projects at scale [Mett21; Chui23]. Long-term operational deployment beyond pilot projects and proof of concepts is thus necessary to justify the investments involved and maximize the economic benefits of developing ML applications in this domain [Chui21; Elle23]. However, the adoption journey of ML in manufacturing is still in its early stages with the majority of applications in companies being smaller-scale pilot projects that do not reach truly operational status [Delo20; Mett21; Manu23].

As ML applications become increasingly mature and transition from academic research and proof of concepts to real-world operational usage, questions regarding their continuous reliability and robustness after deployment arise, which is typically not the focus of academic research [Kühl21]. While this topic may not be necessary if the research objective is to prove the feasibility of a new approach, it is paramount for the practical implementation in industrial environments where it is difficult to guarantee that an ML application operates in the intended domain of use on which it has been trained and validated [Sche15; Jöhn21; ISO/IEC 24029-2]. The training dataset of an ML model is limited to a certain state of the manufacturing process in time. After model deployment though, the manufacturing environment will likely encounter changes such as tool and machine wear, aging or defective sensors or changes in the factory layout and machine placement that are not captured in the training dataset [Wu21], a scenario referred to as *concept drift* [Žlio10b]. Concept drift can lead to reduced performance during the operation of an ML model in contrast to the performance that was evaluated on a static test dataset during development, *cf.* Figure 1.1, up to the point where the ML model becomes useless [Acke20b; Huye22]. Critically, the performance degradation due to concept drift is typically not evident to the user or equipment operator, as most often no continuous feedback of true labels is available for comparison with the model prediction in manufacturing scenarios. Thus, ML models and the data they process must be continuously monitored for concept drift. Additionally, appropriate alarms and mechanisms should be in place to address external changes and technical defects [Kusi17; Stud21; Wu21; Huye22].

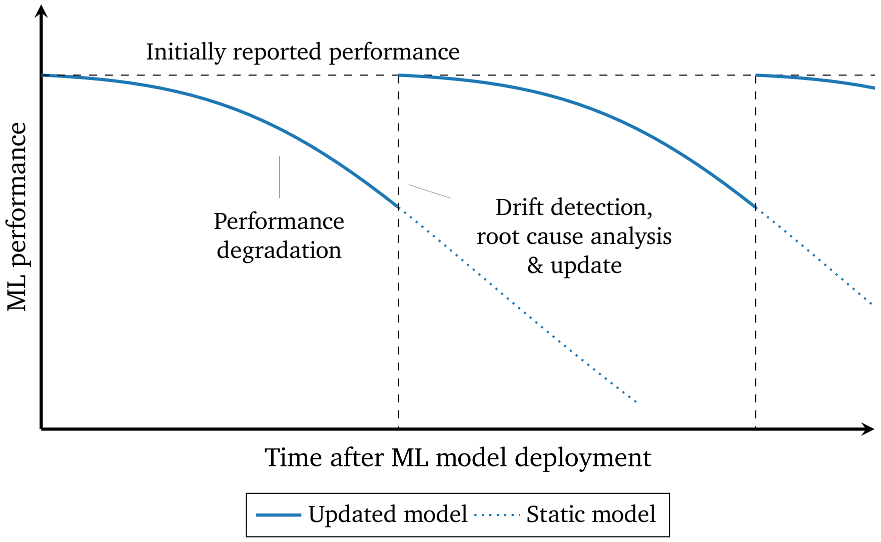


Figure 1.1.: Conceptual performance visualization of a deployed ML model over time in the presence of gradual concept drift. Concept drift can lead to a degradation in performance over time which is not evident to the user. The illustration demonstrates the necessity of detecting drift, identifying the root cause and applying model updates in contrast to a purely static model. Own illustration.

1.1. Problem statement and research questions

In recent years, process models for structuring industrial ML projects such as the Cross-Industry Standard Process Model for the Development of Machine Learning Applications with Quality Assurance Methodology (CRISP-ML(Q)) [Stud21] have emerged that explicitly account for degrading performance over time as this phenomenon applies to a wide range of application areas. These models, however, lack detailed instructions on the practical implementation of mechanisms to detect or counteract performance degradation, merely emphasizing their importance [Elle23].

Concept drift detection and handling are crucial for the reliable long-term operation of ML applications in the manufacturing sector. While research work exists on the methods for detecting or preventing performance degradation in general ML literature, it remains unanswered how these methods can be applied to ML applications in the manufacturing context. The practical problem of data scientists and production experts to design reliable ML applications results in a scientific need of developing a framework that provides implementation guidance.

The Research Objective (RO) of this thesis is thus defined as the following:

Research Objective

Development of a framework for detecting concept drift in ML applications used in process and condition monitoring of manufacturing processes.

Based on this research objective, three research questions are derived which shall be answered throughout this thesis.

Research Question 1

What impact does concept drift have on ML applications used in process and condition monitoring of manufacturing processes and how is it currently addressed in both research and practical applications?

Research Question 2

What methods for detecting concept drift in ML applications exist in the literature?

Research Question 3

How should the detection of concept drift be implemented for ML applications used in process and condition monitoring of manufacturing processes?

1.2. Thesis structure

The work in this thesis is subdivided into six chapters. The chapter structure is summarized and visualized in Figure 1.2.

Chapter 1 introduces the topic and its practical importance. Following this introduction, the research objective is established, and the research questions to be addressed in the thesis are derived. In Chapter 2, the theoretical background is presented, on which the subsequent chapters are built, including ML paradigms, algorithms, performance metrics, practical aspects of implementation as well as the relevant use cases within the manufacturing domain. Chapter 3 analyzes the state-of-the-art in research and practice through a literature review and expert interviews, thus investigating the first research question. The findings are aggregated and factors are derived that guide the development in the subsequent chapter. On this basis, a framework for detecting concept drift in ML applications for monitoring manufacturing applications is developed in Chapter 4, addressing the second and third research questions. To this end, a systematic literature review is conducted to highlight existing approaches as well as their properties and requirements. Additionally, a novel method for conducting hypothesis tests is derived that is particularly suited to the targeted use cases in the manufacturing domain. The developed framework and method for hypothesis testing are consequently validated through three case studies in Chapter 5. The case studies involve condition monitoring as well as process monitoring use cases. Chapter 6 concludes this thesis by summarizing the findings and providing concise answers to the research questions. Lastly, the limitations of the study are discussed and avenues for future research work are derived.

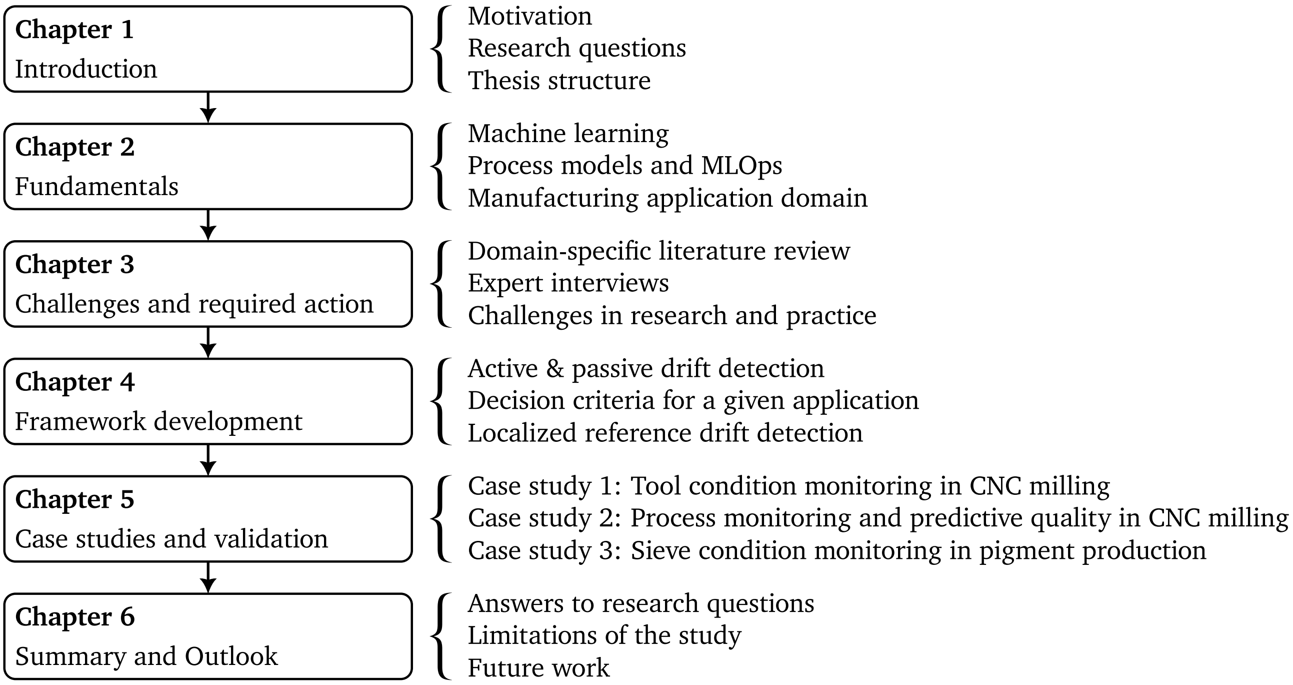

Chapter 1
Introduction
Motivation
Research questions
Thesis structure
Chapter 2
Fundamentals
Machine learning
Process models and MLOps
Manufacturing application domain
Chapter 3
Challenges and required action
Domain-specific literature review
Expert interviews
Challenges in research and practice
Chapter 4
Framework development
Active & passive drift detection
Decision criteria for a given application
Localized reference drift detection
Chapter 5
Case studies and validation
Case study 1: Tool condition monitoring in CNC milling
Case study 2: Process monitoring and predictive quality in CNC milling
Case study 3: Sieve condition monitoring in pigment production
Chapter 6
Summary and Outlook
Answers to research questions
Limitations of the study
Future work

Figure 1.2.: Thesis structure.

2. Fundamentals

In this chapter, the essential theoretical and conceptual foundations for the research in this thesis are introduced. Section 2.1 outlines the relevant parts of fundamental ML theory, while Section 2.2 presents the specific algorithms used throughout this thesis. Section 2.3 presents ML lifecycle concepts, process models as well as the theoretical foundation of concept drift and related phenomena. Section 2.4 introduces the specific hypothesis testing methods that are utilized in the case studies. Section 2.5 provides an overview of process monitoring, predictive quality and condition monitoring as they are the relevant ML application areas for this thesis. Consequently, Section 2.6 introduces relevant features for the respective ML applications in manufacturing that can be extracted from time series data. This chapter concludes in Section 2.7 with a short summary of the presented fundamentals and the derived implications for the following chapters.

The following two sections draw heavily from the books titled *Pattern recognition and machine learning* by Bishop et al. [Bish06], *Machine learning: a probabilistic perspective* by Murphy [Murp12] and *An introduction to outlier analysis* by Aggrawal et al. [Agga17].

2.1. Machine learning paradigms and evaluation

This section provides the required background knowledge concerning the fundamental paradigms of ML relevant to this thesis. First, an overview of high-level concepts in ML is given and the notation is introduced. Consequently, concepts and metrics for the performance evaluation of ML algorithms are explained.

ML is a subset of the broad field of AI which comprises different techniques to resemble intelligent behavior in machines or computer programs [Kühl20]. An often-cited definition of ML by Prof. Tom Mitchell states the following [Mitc97]:

Definition 1 (Machine Learning) *A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

Within Definition 1, the task T refers to the relevant problem that the application addresses, such as predicting the wear condition of a machine tool, e.g., [Hess19]. The performance measure P indicates the quality with which the ML model is able to perform the task T, e.g., the average ratio of correct tool wear classifications. Lastly, the experience E refers to the dataset that is used to create or train the ML model. Another well-cited description of ML attributed to Prof. Arthur Samuel states that ML algorithms build a model based on sample data, in order to make predictions or decisions without being explicitly programmed to do so [Samu59].

2.1.1. Learning paradigms and notation

When analyzing ML in the context of industrial applications, a further differentiation between learning paradigms is necessary. A distinction is typically made between supervised learning, unsupervised learning and reinforcement learning [Jord15; Wues16]. While all three paradigms have seen widespread usage in industrial and scientific contexts, supervised and unsupervised learning are the most relevant for the specific tasks that are analyzed within this thesis. Thus, they are introduced in more detail in this section.
In addition, another relevant distinction can be made temporally between online/sequential learning and offline learning, which is introduced thereafter.

Supervised learning

In supervised learning, the model learns a mapping between a vector of inputs $\boldsymbol{x}_i \in \mathbb{R}^d$ and the corresponding label $y_i \in \mathbb{R}$ given a set of input-output pairs $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. In this notation, $\mathcal{D}$ is referred to as the dataset, and n is the number of data samples. $\boldsymbol{x}_i$ is a d-dimensional vector that contains representative attributes of the sample i. $X \in \mathbb{R}^{n \times d}$ is the matrix that contains the sample vectors in its rows. The attributes are often referred to as features whereas the label y_i is often alternatively referred to as the annotation or target variable. Attributes can be real-valued, continuous values such as sensor readings or discrete, categorical values such as a system status or mode. Note that in certain use cases such as working with image-like data or multivariate time series, the single input data samples can have matrix shape $X_i \in \mathbb{R}^{d_1 \times d_2}$ with two dimensions or tensor shape $\boldsymbol{X}_i \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ with three or more dimensions.
The label y_i can be either continuous or discrete. The discrete case is referred to as **classification** and y_i is assumed to take values from a finite set of c options $j \in \{1, \ldots, c\}$, usually referred to as classes. An exemplary classification task from the domain of manufacturing is the classification of the product quality as $y_i \in \{\text{OK}, \text{NOK}\}$. In this example, $\boldsymbol{x}_i$ might contain attributes like the amplitude of the measured vibration or the average temperature during the machining process. Classification tasks with $c = 2$ classes are referred to as binary classification tasks and the classes are often termed positive (1) and negative (0), $y_i \in \{0, 1\}$. Classification tasks with $c > 2$ classes are referred to as multiclass classification. On the other hand, if y_i is real-valued, the corresponding ML task is referred to as **regression**. An exemplary regression task from the manufacturing domain is the regression of a specific quality parameter such as a geometric diameter, based on relevant sensor measurements from the process. Depending on the use case, the target variable for a single sample may also be shaped as a vector $\boldsymbol{y}_i \in \mathbb{R}^{d_1}$, matrix $Y_i \in \mathbb{R}^{d_1 \times d_2}$ or tensor $\boldsymbol{Y}_i \in \mathbb{R}^{d_1 \times d_2 \times d_3}$.

The following parts of this chapter focus on classification tasks, as these are most relevant for the subsequent chapters. Classification problems can be formulated as function approximation problems with the goal of learning an estimate $\hat{y} = \hat{f}(\boldsymbol{x})$ of the true function $y = f(\boldsymbol{x})$ using the dataset $\mathcal{D}$. The estimates of classification algorithms are typically represented as probability distributions over possible labels $j \in \{1, \ldots, c\}$, conditioned on the input sample $\boldsymbol{x}$ and model parameters $\boldsymbol{\theta}$ as $p(y \mid \boldsymbol{x}, \boldsymbol{\theta})$. Models with a fixed number of parameters $\boldsymbol{\theta}$ are called parametric models while models where the number of parameters grows with the amount of training data are called nonparametric. For a given input sample $\boldsymbol{x}_i$, a probability $p(y = j \mid \boldsymbol{x}_i, \boldsymbol{\theta})$ is associated with every class j and the final class prediction $\hat{y}_i \in \{1, \ldots, c\}$ is typically inferred as the class with the highest associated probability:

$$\hat{y}_i = \hat{f}(\boldsymbol{x}_i) = \underset{j}{\operatorname{argmax}}\, p(y = j \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \tag{2.1}$$

This way of estimation is referred to as Maximum a Posteriori (MAP) estimation [Bish06; Murp12].

Unsupervised learning

Unsupervised learning describes the identification of relevant patterns from unlabelled input data $\mathcal{D} = \{(\boldsymbol{x}_i)\}_{i=1}^n$. Most tasks in the field of unsupervised learning can be formulated as density estimation tasks, building models that represent the likelihood of the data itself as $p(\boldsymbol{x}_i \mid \boldsymbol{\theta})$. Common tasks within unsupervised learning are clustering, dimensionality reduction and anomaly detection out of which anomaly detection is the most relevant for this thesis.

Anomaly detection, also referred to as outlier detection, novelty detection and out-of-distribution detection [Agga17], has great relevance for various practical applications, including fault detection in manufacturing, intrusion detection for cyber-security as well as fraud detection in payment systems [Chan09]. In its simplest form, an anomaly is is defined as a data point that differs significantly from the rest of the data. Data within applications like the ones previously mentioned typically has a *normal* mode, which anomalies are deviating from. Many algorithms for anomaly detection exist, three of which are introduced in the following parts of this chapter. For a given data point $\boldsymbol{x}_i$, anomaly detection algorithms output either

- a continuous outlier score $h(\boldsymbol{x})$, quantifying the level of abnormality in the data point under the assumptions of the algorithm, or
- a binary label $\hat{y}_i \in \{0, 1\}$, indicating whether the data point is an outlier or not, which is a type of binary classification.

While a continuous score is useful for, e.g., ranking possible outliers as well as for the performance evaluation of models, practical applications typically require binary labels. Outlier scores h can be converted to binary labels $\hat{y}$ by applying a threshold δ as

$$\hat{y}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } h(\boldsymbol{x}) \geq \delta \\ 0 & \text{if } h(\boldsymbol{x}) < \delta \end{cases} . \tag{2.2}$$

The threshold δ is typically chosen based on the statistical distribution of the anomaly scores.

A number of variants exist that combine parts of both supervised and unsupervised learning paradigms. While fully unsupervised anomaly detection tasks assume that anomalies in the respective dataset are rare, there are multiple levels of partial supervision possible, with varying terminologies in the literature. Typically, semi-supervised anomaly detection tasks contain either some (labeled) examples of anomalies within the data or explicitly only normal data [Agga17].

Sequential learning and offline learning

A further distinction is made between different temporal learning paradigms that are relevant for this thesis: offline learning and sequential learning, often referred to as online learning [Gama09; Žlio10b].
Offline learning largely overlaps with the scenario described for supervised learning above. A fixed set of historical data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ is used to train a model which is consequently deployed to make predictions on new data points. The model is not updated or influenced by the predictions it is making on the new data points after deployment. **Sequential learning** on the other hand, describes a paradigm where the model is not static over time. An initial model is trained on historical data. After deployment though, sequential learning assumes that for a new data point $\boldsymbol{x}_t$ at time t, the ground truth label y_t is revealed after the model has made its prediction $\hat{y}_t$. The new data point $(\boldsymbol{x}_t, y_t)$ is then added to the dataset $\mathcal{D}$ and the model is retrained on the whole or parts of this extended dataset. Thus, the model is updated after every new data point [Žlio10b].

In practice, sequential learning is used in scenarios where data continuously streams, ground truth labels quickly become available and it is crucial to adapt to new data in real-time or near-real-time. Examples include online recommender systems where labels can be inferred from user behavior, as well as forecasting systems where the true value is observed after the forecasting period. In the scope of ML applications for the monitoring of manufacturing processes, offline learning is prevalent as ground truth labels are typically not observable after deployment [Jour23a].

2.1.2. Performance evaluation

In the following section, the most common metrics for performance evaluation of classification and anomaly detection algorithms are introduced. Consequently, concepts for evaluating model performance and their respective reasoning are introduced, including dataset splits and Cross-Validation (CV).

Metrics

As explained in Section 2.1.1, classification can be done either in a binary ($c = 2$) or multiclass ($c > 2$) fashion. The metrics explained in the following are targeting binary classification problems with a positive (1) and a negative (0) class, $y \in \{0, 1\}$, where P is defined as the number of positive samples, while N is defined as the number of negative samples in the dataset that is used for evaluation. Most metrics assume that the positive class is the class of interest, *i.e.* that should be detected, while the negative class is the more common, general case [Murp12; Agga17]. Multiclass problems can be evaluated in a similar fashion to binary problems, by evaluating on a per-class basis, treating the class of interest as positive, and all other classes as the combined negative class. This is also referred to as One-vs-Rest evaluation. Classified samples in binary classification problems belong to one of the following four categories:

- **True Positive (TP)**: Number of positive samples which are correctly classified as positive.
- **True Negative (TN)**: Number of negative samples which are correctly classified as negative.
- **False Negative (FN)**: Number of positive samples which are incorrectly classified as negative.
- **False Positive (FP)**: Number of negative samples which are incorrectly classified as positive.

These metrics are often visualized in a confusion matrix as shown in Figure 2.1. The confusion matrix for multiclass classification similarly shows the predicted classes as rows and true classes as columns.

		True class y	
		Positive ($y = 1$)	Negative ($y = 0$)
Prediction $\hat{y}$	Positive ($\hat{y} = 1$)	TP	FP
	Negative ($\hat{y} = 0$)	FN	TN
	Total	P	N

Figure 2.1.: Confusion matrix of a binary classifier. Own illustration.

Further metrics can be calculated based on these scores:

The **accuracy**

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} \tag{2.3}$$

describes the fraction of all instances that have been classified correctly. Accuracy is a common metric used to evaluate classification systems. A fundamental problem of this metric is its sensitivity to class imbalance. Class imbalance exists if the samples are not evenly distributed across the set of classes in a dataset. As accuracy uses values from both columns in the confusion matrix, its value changes for a different class distribution even if the classification performance of the evaluated model remains the same [Bish06].

The **True Positive Rate (TPR)**, also called **hit rate** and **recall**, is defined as the fraction of correctly classified positive samples:

$$\text{TPR} = \text{recall} = \frac{\text{TP}}{\text{P}}. \tag{2.4}$$

The **False Positive Rate (FPR)**, also called **false alarm rate**, is defined as the fraction of incorrectly classified negative samples:

$$\text{FPR} = \frac{\text{FP}}{\text{N}}. \tag{2.5}$$

TPR/recall and FPR do not depend on both columns of the confusion matrix and are therefore insensitive against changes in class distribution.

The **precision** is defined as:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.6}$$

The **F_1-score** is defined as the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{2.7}$$

Most supervised classification algorithms, *cf.* Section 2.1.1, produce scores $p(y = j \mid \boldsymbol{x})$ for each class, which are interpreted as pseudo-probabilities that describe the likelihood of the sample $\boldsymbol{x}$ belonging to a certain class j. Binary classifiers often only produce a single score for the positive class, *cf.* Section 2.2. The predicted class label $\hat{y}$ is commonly chosen as the class with the highest score via Equation (2.1). Similarly, anomaly detection algorithms often produce anomaly scores h, indicating the degree to which a sample is considered an anomaly. A threshold δ is subsequently used in Equation (2.2) to infer the binary predicted class label $\hat{y} \in \{0, 1\}$. When evaluating a classifier, one may be interested in the overall quality of the score in addition to the absolute class predictions. Depending on the application scenario, even the class predictions of supervised classification algorithms may be decided using a fixed threshold δ instead of simply choosing the maximum scoring class since the threshold can be used to control the proportion of FPs vs FNs, as visualized in Figure 2.2. In the binary case, the decision function then becomes Equation (2.2), analogous to anomaly detection.

A common method to assess the quality of classification scores as well as anomaly scores independent of the threshold is the Receiver Operating Characteristic (ROC). The ROC visualizes the relative trade-off between TPR and FPR of binary classifiers [Fawc06]. In ROC space, a classifier is represented by its FPR on the abscissa and its TPR on the ordinate. As the ROC depends only on rates, it is robust against changes in class distribution. A discrete classifier with a threshold δ is represented by a single value pair (TPR, FPR), which corresponds to a point in ROC space also called the operating point. A perfect classifier is represented by the point $(0, 1)$, classifying every positive sample correctly with no FPs. Points on the diagonal line TPR $=$ FPR represent a random classifier that has the same likelihood of choosing either class.

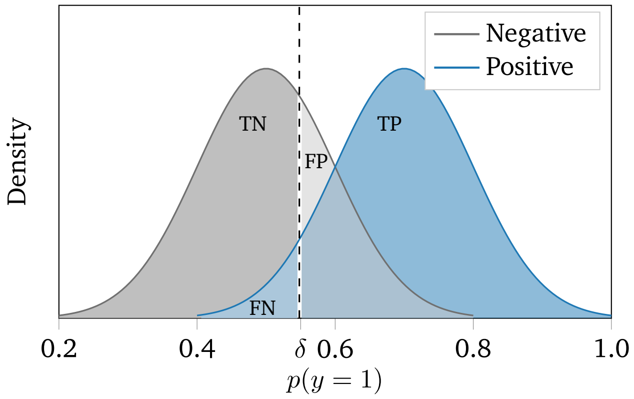


Figure 2.2.: Threshold-based classification and resulting proportions of error types. High thresholds δ lead to conservative classification, with proportionally more FNs, while low thresholds lead to more FPs. The threshold setting can thus be tuned to reflect the costs of incurring either error in the application scenario. Own illustration.

Classifiers that produce a continuous output score can be visualized as a curve in ROC space by varying the threshold δ of Equation (2.2). Each threshold value corresponds to a point in ROC space. The analysis in ROC space can be used to tune the threshold for a specific use case and the desired behavior of the classifier. The highest threshold value results in the ROC point $(0, 0)$, rejecting all samples as negatives. Lowering the threshold results in a movement to the upper right corner of the ROC space, reaching the point $(1, 1)$ for the lowest threshold. An example of the ROC curves of two classifiers is shown in Figure 2.3. In this example, classifier A shows superior performance compared to classifier B, as its ROC curve is closer to the top left of the ROC space. The diagonal TPR $=$ FPR representing a random classifier is shown as a dashed line. A ROC curve beneath this diagonal indicates performance that is worse than random guessing. The ROC space visualizes the classifier's ability to produce meaningful relative scores rather than calibrated ones.

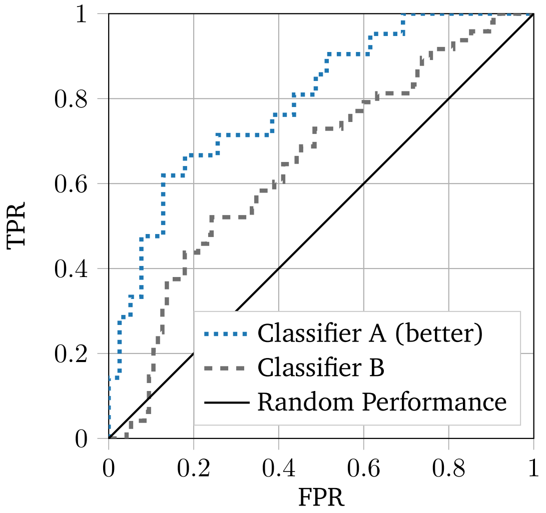


Figure 2.3.: ROC curves of two example classifiers. Classifier A shows superior performance (AUROC $= 0.79$) compared to Classifier B (AUROC$= 0.65$). The solid line shows the ROC for random performance yielding AUROC $= 0.5$. Own illustration.

The ROC curve of a classifier that produces a continuous score can be summarized in a single scalar by calculating the Area Under the Receiver Operating Characteristic (AUROC). Since both FPR and TPR are limited to the interval $[0, 1]$, the AUROC is limited to the interval $[0, 1]$ as well. The AUROC has an intuitive statistical property. The AUROC of a classifier is equivalent to the probability that the classifier will assign a higher score to a randomly chosen positive instance than a randomly chosen negative one. A value of AUROC $= 0.5$ corresponds to a random classifier, while a perfect classifier corresponds to AUROC $= 1.0$ [Fawc06]. In the example of Figure 2.3, Classifier A has a higher AUROC than Classifier B indicating better performance.

Model selection and assessment

There are a variety of ML algorithms for both supervised learning and unsupervised learning with different properties and capacities, of which the developer has to select one given the configuration of the use case and other requirements. In addition, ML algorithms typically have hyperparameters, which configure their learning and inference process. In contrast to learnable parameters $\boldsymbol{\theta}$, hyperparameters are not optimized in the learning process but have to be chosen by the developer, either in a manual or partially automated way. The process of model selection involves choosing both the algorithm as well as its appropriate hyperparameter settings [Murp12]. This requires several considerations which are briefly explained in the following.

Generalization, overfitting and bias-variance trade-off In the context of model selection and assessment, several aspects are of importance for the investigation in this thesis. In most real-world application scenarios, ML models are used to classify new data that is similar but not equal to the training data. The ability to correctly categorize these new samples is known as **generalization**, which is a central goal of ML engineering. This is because even a large set of training data can only comprise a tiny fraction of all possible occurrences within the real world [Bish06]. The degree to which a model is able to fit the training data is not a good indicator of how well this model will perform on unseen (test) data due to the possibility of **overfitting** to the training data. Overfitting describes the phenomena of a model being overly adapted to the particularities of the training data, e.g., by memorizing parts of it internally. In the case of k-Nearest Neighbor (kNN) classification, *cf.* Section 2.2, an extreme case of overfitting can be observed for $k = 1$. In this case, the training error will be 0 as the kNN always returns the correct label for each training sample without necessarily having any generalization ability. Overfitting is often caused by using ML models with too much complexity / high degrees of freedom, *cf.* Figure 2.4, for the problem at hand [Murp12].

As the ML training process commonly involves the fastest way of minimizing some kind of loss function which depends on the training error, the model will memorize the training data if it has the capability and capacity to do so. Especially with high-dimensional input data, complex models tend to get sensitive to random patterns and small fluctuations in the training data such as measurement noise which is not descriptive of the underlying data generating process they are used to model. The error due to this phenomenon is termed **variance**. It is thus important to select an ML model with a capacity that is appropriate for the problem that should be modeled. Overfitting can be identified by analyzing the discrepancy between the model error on the training and test set, *cf.* Figure 2.4. Conversely, if a model is chosen that is too simple for the problem, underfitting can occur, e.g., when a linear model is applied to data in which the dependencies are nonlinear. While simple models such as linear regression tend to not overfit as strongly as more complex models such as Neural Networks (NNs), they introduce errors caused by their limitations and assumptions on the data. The error due to this phenomenon is termed **bias**.

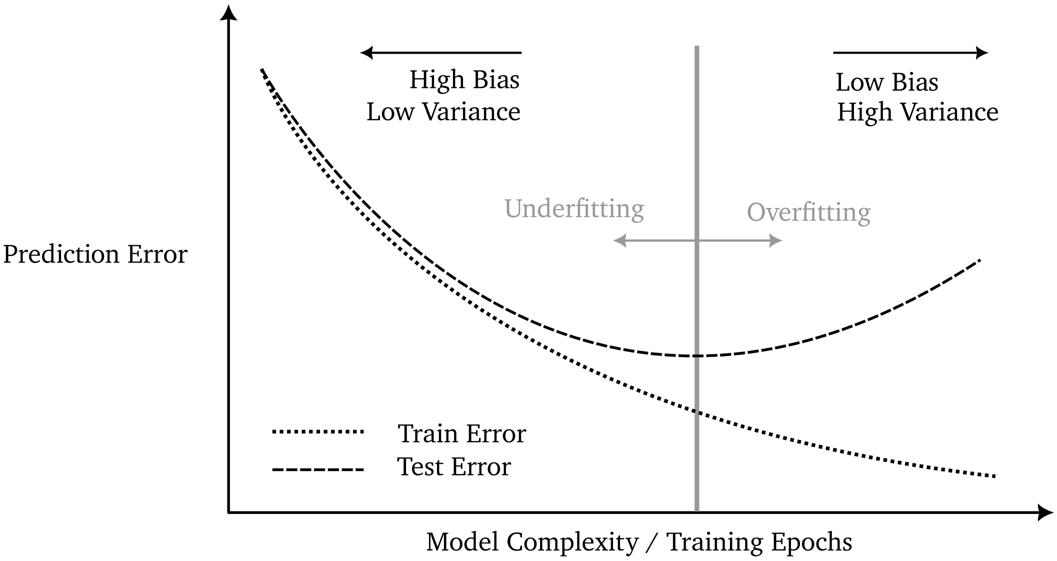


Figure 2.4.: Behavior of test and training set error as the model complexity is varied. Meaningful learning stops when the test error plateaus. In the context of deep learning, the abscissa in this figure can also refer to the number of epochs the model is allowed to train, *cf.* Section 2.2. Own illustration adapted from [Hast09].

In this context, the **bias-variance trade-off** describes the conflict of trying to minimize both error sources during the model selection process. Models that have low bias, such as NNs, are more vulnerable to variance and vice versa [Murp12], *cf.* Figure 2.4.

For the reasons stated above, the introduced performance metrics are not calculated and assessed on the data the model is trained with but on a reserved part of the data that is not used during the training process. The typical process is visualized in Figure 2.5. The available dataset $\mathcal{D}$ is randomly split into three sets:

- The **training set** $\mathcal{D}_{\text{train}}$ of length n_{train} used to train the models,
- the **validation set** $\mathcal{D}_{\text{val}}$ of length n_{val} used to optimize the hyperparameters that control the training process, and
- the **test set** $\mathcal{D}_{\text{test}}$ of length n_{test} used to assess the final model and estimate its generalization performance on unseen data.

The test set should only be used to calculate the metrics explained in Section 2.1.2 after all optimizations of the model and the training process are final, to give a realistic estimate of the achievable real-world performance in the model assessment step [Murp12].

Fixed training-validation splits have several downsides with the major one being that the validation set only gives a very noisy estimate of the final model performance and can thus lead to suboptimal choices of the hyperparameters. This is due to the often small portion of the available data that is assigned to it, *i.e.* 10%-20%. It is thus best practice to use Cross-Validation (CV) instead, as visualized in Figure 2.6. Here, the training set is not split into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$ but rather split into k folds that are iteratively used to train the model and validate its performance k times. The final validation performance that is used to select the optimal set of hyperparameters is the average of the k obtained validation results of the single folds.

The advantage of CV is the usage of the full training set for validation and thus a less noisy estimate of the final model performance, often leading to a better selection of hyperparameters [Murp12]. CV is adopted for all experiments in this thesis.

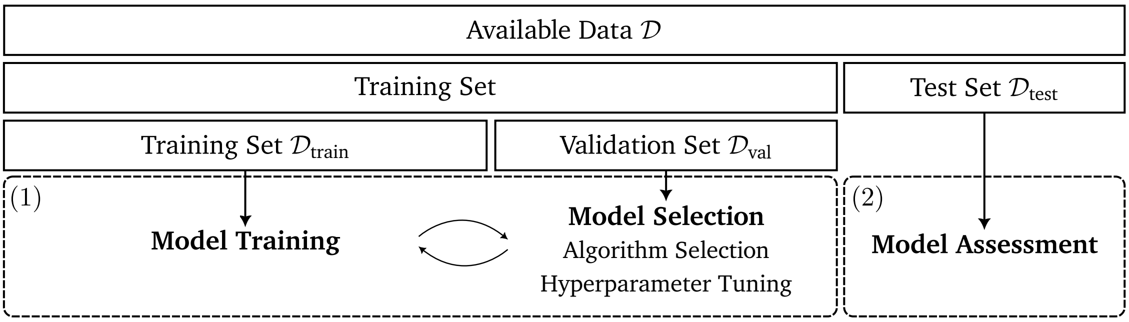

Available Data $\mathcal{D}$
Training Set
Test Set $\mathcal{D}_{\text{test}}$
Training Set $\mathcal{D}_{\text{train}}$
Validation Set $\mathcal{D}_{\text{val}}$
(1)
Model Training
Model Selection
Algorithm Selection
Hyperparameter Tuning
(2)
Model Assessment

Figure 2.5.: Dataset splits for ML training, model selection and model assessment. Own illustration adapted from [Pedr11].

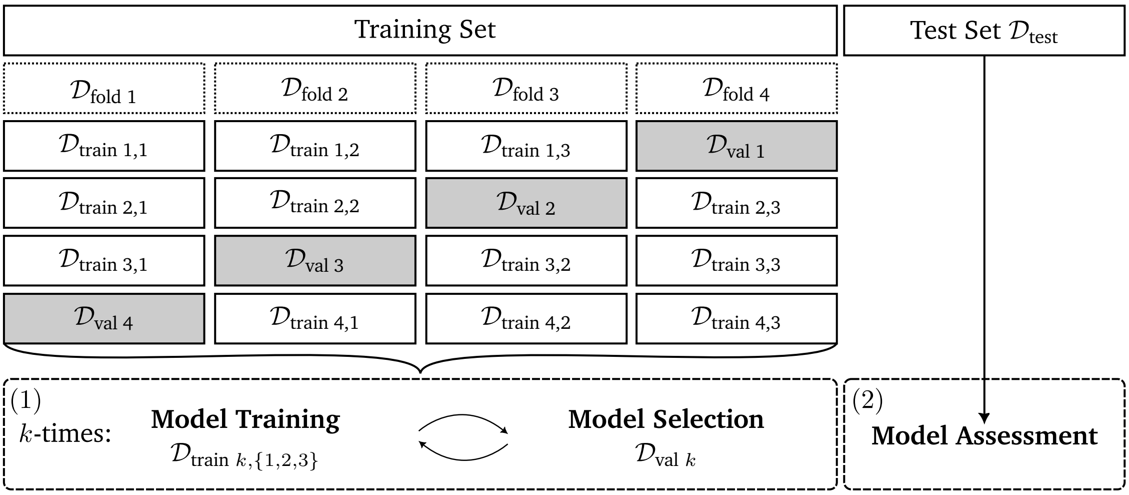

Training Set
Test Set $\mathcal{D}_{\text{test}}$
$\mathcal{D}_{\text{fold 1}}$
$\mathcal{D}_{\text{fold 2}}$
$\mathcal{D}_{\text{fold 3}}$
$\mathcal{D}_{\text{fold 4}}$
$\mathcal{D}_{\text{train 1,1}}$
$\mathcal{D}_{\text{train 1,2}}$
$\mathcal{D}_{\text{train 1,3}}$
$\mathcal{D}_{\text{val 1}}$
$\mathcal{D}_{\text{train 2,1}}$
$\mathcal{D}_{\text{train 2,2}}$
$\mathcal{D}_{\text{val 2}}$
$\mathcal{D}_{\text{train 2,3}}$
$\mathcal{D}_{\text{train 3,1}}$
$\mathcal{D}_{\text{val 3}}$
$\mathcal{D}_{\text{train 3,2}}$
$\mathcal{D}_{\text{train 3,3}}$
$\mathcal{D}_{\text{val 4}}$
$\mathcal{D}_{\text{train 4,1}}$
$\mathcal{D}_{\text{train 4,2}}$
$\mathcal{D}_{\text{train 4,3}}$
(1)
k-times:
Model Training
$\mathcal{D}_{\text{train } k,\{1,2,3\}}$
Model Selection
$\mathcal{D}_{\text{val } k}$
(2)
Model Assessment

Figure 2.6.: k-fold CV with $k = 4$ as a more comprehensive alternative to training-validation set splits. Instead of a fixed training-validation set split, a k-fold split is applied. Own illustration adapted from [Pedr11].

Independent of whether training-validation splits or k-fold CV is used, optimization of the hyperparameters is done in an iterative fashion. The most common way to optimize hyperparameters is **grid search**. In grid search, the hyperparameter space is discretized in a n-dimensional grid, with n being the number of hyperparameters that should be optimized. For each hyperparameter, a suitable set of options or a discretized value range is selected and consequently a model is trained using every parameter combination, either on the training split, if fixed splits are used, or via CV. Finally, the parameter combination with the lowest validation error is selected as the final parameter set [Zhan23].

2.2. Machine learning algorithms

This section introduces the supervised classification algorithms as well as the unsupervised anomaly detection algorithms that were utilized in the experiments throughout this thesis. The section is divided into neighbor-based algorithms, tree-based algorithms, Support Vector Machines (SVMs) as well as NNs. Lastly, the relevant ML-based visualization techniques for high-dimensional data are introduced. Within this thesis, the adaptation of a model's learnable parameters $\boldsymbol{\theta}$ or memory to a given dataset $\mathcal{D}$ is referred to as training, while the usage of a trained model on new data points is referred to as inference.

Neighbor-based algorithms

Neighbor-based algorithms rely on data point proximity to analyze patterns, emphasizing spatial relationships for predictions or classifications, based on the principle that similar data points tend to cluster together. The relevant neighbor-based algorithms for this thesis include kNN classification as well as Local Outlier Factor (LOF).

k-nearest neighbor classification

The kNN algorithm is a classification[1] algorithm that estimates the class membership of an input data sample $\boldsymbol{x}$ by determining the majority class of the k closest points out of the training dataset $\mathcal{D}_{\text{train}}$ given some distance metric $d(\cdot)$. An example for kNN classification is visualized in Figure 2.7. Formally, the kNN algorithm is based on the assumption that the likelihood of an input data sample $\boldsymbol{x}$ belonging to class j can be expressed as

$$p(y = j \mid \boldsymbol{x}, \mathcal{D}_{\text{train}}, k) = \frac{1}{k} \sum_{i \in n_k(\boldsymbol{x}, \mathcal{D}_{\text{train}})} \mathbb{I}\left(y_i = j\right), \tag{2.8}$$

where $n_k(\boldsymbol{x}, \mathcal{D}_{\text{train}})$ are the k nearest training samples to the input data sample $\boldsymbol{x}$ in $\mathcal{D}_{\text{train}}$ and $\mathbb{I}(e)$ is the indicator function defined as follows:

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases}. \tag{2.9}$$

k is a hyperparameter which is manually optimized and should be set to an odd number $k \in \{2\mathbb{N} - 1\}$ to avoid ties in the classification [Murp12]. kNN is a nonparametric algorithm as it does not have trainable parameters $\boldsymbol{\theta}$ and instead uses memory-based learning. A common metric utilized to compute the distance between two d-dimensional data points $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{R}^d$ with continuous features is the Euclidian distance

$$d_{\text{EUC}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 = \sqrt{\sum_{l=1}^{d} \left(\boldsymbol{x}_i^{(l)} - \boldsymbol{x}_j^{(l)}\right)^2}. \tag{2.10}$$

The choice of distance metric represents another hyperparameter of the kNN algorithm. In addition to the majority voting in Equation (2.8), most implementations of the kNN algorithm such as in the popular framework *scikit-learn* [Pedr11] allow a weighting of the neighbors by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away. If $\boldsymbol{x}$ contains categorical features then special distance metrics such as the Hamming distance or a prior conversion of the categorical data to a numerical format is required.

[1] kNN can also be used for regression tasks, however, this is out of this thesis' scope.

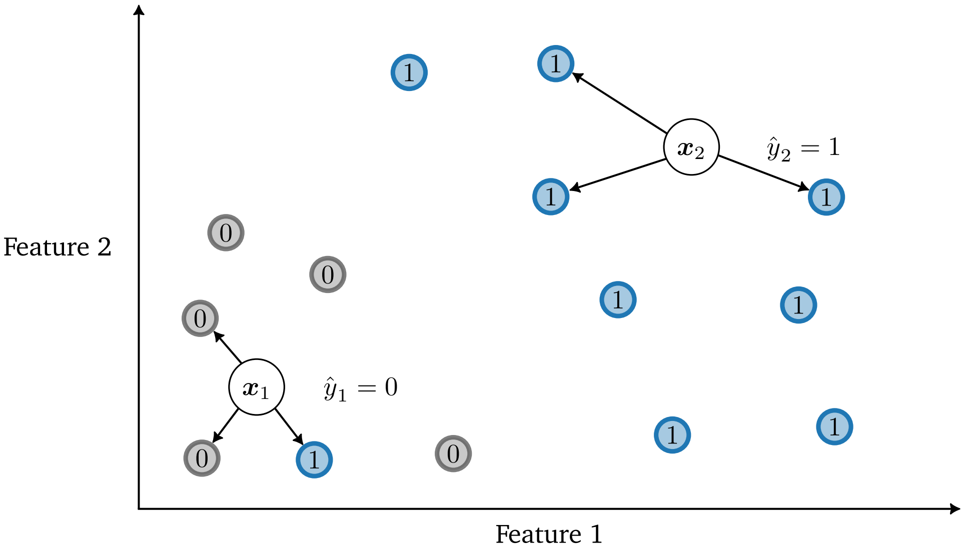


Figure 2.7.: Example visualization of kNN with $k = 3$ applied to a two-dimensional dataset with two classes $y_i \in \{0, 1\}$. Data point $\boldsymbol{x}_1$ would be classified as negative with $p(y = 0 \mid \boldsymbol{x}_1, \mathcal{D}_{\text{train}}, k = 3) = \frac{2}{3}$, $\boldsymbol{x}_2$ as positive with $p(y = 1 \mid \boldsymbol{x}_2, \mathcal{D}_{\text{train}}, k = 3) = \frac{3}{3} = 1$. Own illustration adapted from [Murp12].

The most commonly used method for converting categorical features to numerical values is **one-hot encoding**, which converts a categorical scalar variable x with m possible states to a vector $\tilde{\boldsymbol{x}} \in \mathbb{R}^m$ as

$$\tilde{\boldsymbol{x}} = [\mathbb{I}(x = 1), \ldots, \mathbb{I}(x = m)]. \tag{2.11}$$

For instance, if there are $m = 5$ possible states, then a data point from state $x = 2$ would be given the one-hot encoded vector

$$\tilde{\boldsymbol{x}} = (0, 1, 0, 0, 0)^T.$$

This form of preprocessing the data is common for a number of algorithms and may also be used to preprocess labels y for algorithms such as NNs, *cf.* Section 2.2.

As distance metrics such as the Euclidian distance rely on absolute distances of the feature values, they are sensitive to the scale of the feature values. If features represent different physical units or have inherently different value ranges such as, e.g., vibration amplitudes vs temperatures, **standardization** of the input data is a requirement. A real-valued variable $x \in \mathbb{R}$ can be standardized as

$$\tilde{x} = \frac{x - \hat{\mu}}{\hat{\sigma}} \tag{2.12}$$

where $\hat{\mu}$ is the sample mean of the training dataset, computed as

$$\hat{\mu} = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n} x_i \tag{2.13}$$

and $\hat{\sigma}$ is the sample standard deviation of the training dataset, computed as:

$$\hat{\sigma} = \sqrt{\frac{1}{n_{\text{train}} - 1} \sum_{i=1}^{n_{\text{train}}} (x_i - \hat{\mu})^2}. \tag{2.14}$$

As mentioned for one-hot encoding, standardization of the input data to align the feature value ranges is a common preprocessing method for a number of algorithms besides kNN such as NNs. The standardization parameters $\hat{\mu}, \hat{\sigma}$ directly influence the training process of the model. They should thus be computed on the training set instead of the whole set of available data as knowledge about $\mathcal{D}_{\text{test}}$ might otherwise *leak* into the model, making the later model assessment less trustworthy with respect to the actual real-world performance [Murp12].

Local outlier factor LOF [Breu00] is a neighbor-based algorithm for unsupervised anomaly detection that identifies abnormal data points by measuring the local deviation in density around a given data point with respect to its neighbors. Calculation of the LOF score is based on reachability distances. The reachability distance of a point $\boldsymbol{x}_i$ with respect to another point $\boldsymbol{x}_j$ is defined as $\max\{k\text{-distance}(\boldsymbol{x}_j), d(\boldsymbol{x}_i, \boldsymbol{x}_j)\}$ where the k-distance is defined as the distance between $\boldsymbol{x}_j$ and its $k-$th nearest neighbor. The reachability distances of a data point are aggregated to Local Reachability Density (LRD). The LRD of a data point is the inverse of the average reachability distance of it to its k neighbors. A low LRD for $\boldsymbol{x}_i$ means it is far from its neighbors compared to how close those neighbors are to each other. The LOF calculates an anomaly score $h(\boldsymbol{x}_i)$ for each data point as

$$h(\boldsymbol{x}_i) = \frac{\sum_{j \in n_k(\boldsymbol{x}_i)} \frac{\mathrm{LRD}(\boldsymbol{x}_j)}{\mathrm{LRD}(\boldsymbol{x}_i)}}{|n_k(\boldsymbol{x}_i)|}, \tag{2.15}$$

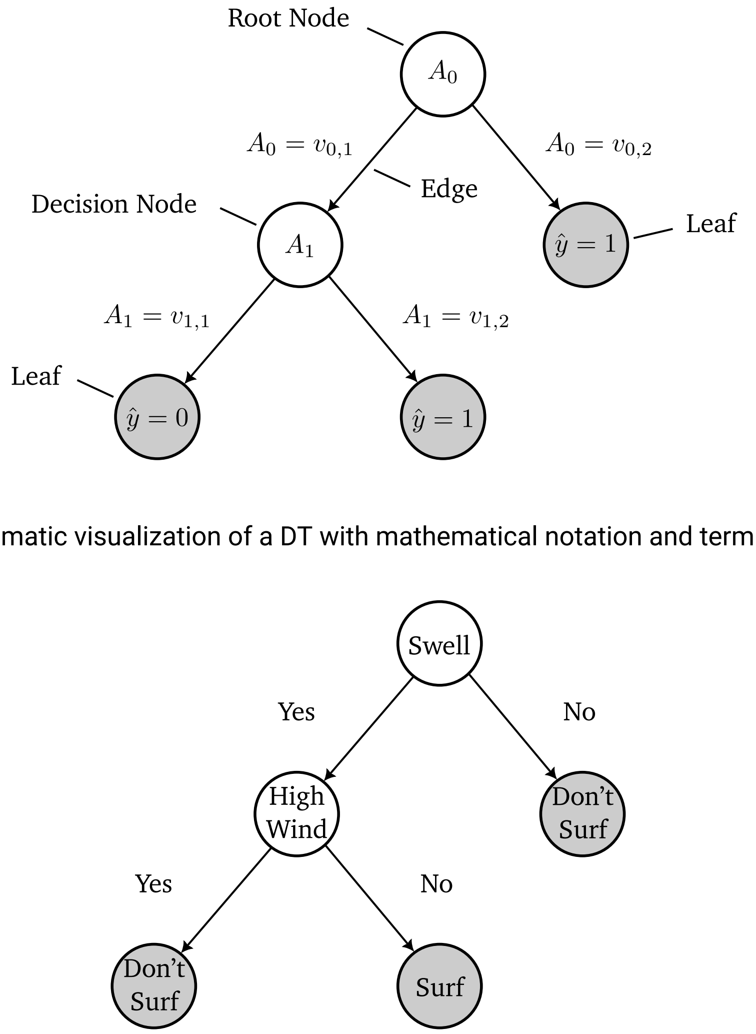
where $n_k(\boldsymbol{x}_i, \mathcal{D}_{\text{train}})$ are the k nearest points of the training samples in $\mathcal{D}_{\text{train}}$ to $\boldsymbol{x}_i$. A score around 1 indicates that the data point has a similar density to its neighbors (considered normal), while a score significantly greater than 1 suggests that the data point is an outlier. The greater the LOF score, the more anomalous the data point is. As with kNN, the most important hyperparameters of the LOF are the choice of the distance measure as well as the number of neighbors k.

Tree-based algorithms

Tree-based algorithms organize data using tree structures to infer predictions, segmenting the dataset into branches based on decision rules. This approach facilitates efficient classification tasks by hierarchically breaking down the data. First, simple Decision Trees (DTs) are introduced. Consequently, the concept of ensemble learning is described, before explaining the Random Forest (RF) and Isolation Forest (IF) algorithms.

Decision trees DTs recursively partition the input space using decision rules in a hierarchical fashion such that samples with the same classes are grouped together. DTs consist of three basic building blocks: nodes, edges and leaves, *cf.* Figure 2.8. At each node, a single feature of the data point is tested. Edges connect the possible options of the feature tested at a given node to either further tests in other nodes or leaves. Leaves constitute the class predictions of the DT.

DTs are trained using a dataset $\mathcal{D}_{\text{train}}$ and the training process is done in a recursive fashion. The basic algorithm for learning DTs from datasets that only contain categorical features is the Iterative Dichotomiser 3 (ID3) algorithm [Quin86]. It starts by selecting the feature of the dataset that best separates the instances into different classes for the root node. The selection of the feature should yield edges that ideally contain pure



(a) Schematic visualization of a DT with mathematical notation and terminology.

(b) Exemplary instantiation with the decision of going surfing depending on the environmental conditions.

Figure 2.8.: Decision tree for a binary decision with $y \in \{0, 1\}$ and two nodes, testing binary features A_0 and A_1 each. Own illustration.

sets with respect to the classes of the training set instances. To this end, the selection is based on maximizing a certain criterion, such as the Information Gain (IG) which is defined as

$$\mathrm{IG}(\mathcal{D}, A) = H(\mathcal{D}) - \sum_{v \in \text{values}(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} H(\mathcal{D}_v). \tag{2.16}$$

In Equation (2.16), A is a feature in the dataset and H is a measure of impurity. $\mathcal{D}_v$ refers to the samples in $\mathcal{D}$ that have the value v within feature A. Possible choices for H include the Shannon Entropy

$$H_{\text{Entropy}} = -\sum_j p_j \log_2 p_j, \tag{2.17}$$

where p_j is the proportion of elements with class j in the training dataset $\mathcal{D}_{\text{train}}$. An alternative to using the Entropy is using the Gini coefficient

$$H_{\text{Gini}} = -\sum_j p_j(1 - p_j). \tag{2.18}$$

The choice of the criterion for assessing the split quality is a central hyperparameter of the decision tree. In the case of real-valued features, this selection does not only involve the feature but also the split value within the feature's value range. This is implemented in the successor algorithm of ID3, called C4.5 [Quin14].

After a feature is selected, edges are constructed for each value of that feature. Depending on the pureness of the data points that correspond to each edge, either leaf nodes are constructed, or another decision node is constructed, using the same mechanism for the selection of the next test feature. The pureness is assessed using either the Shannon Entropy or the Gini coefficient as described before. If a leaf node is constructed, the corresponding prediction class is set as the majority class of the corresponding data points in the training dataset. A prediction probability for a sample $\boldsymbol{x}$, and a trained DT $\mathcal{T}$ is calculated similar to the kNN, *cf.* Equation (2.8), as the proportion of training samples of each class j in the respective leaf node

$$p(y = j \mid \boldsymbol{x}, \mathcal{T}) = \frac{\sum_{i \in \text{Leaf}(\boldsymbol{x})} \mathbb{I}(y_i = j)}{|\text{Leaf}(\boldsymbol{x})|}. \tag{2.19}$$

In Equation (2.19), $|\text{Leaf}(\boldsymbol{x})|$ is the total number of training instances out of $\mathcal{D}_{\text{train}}$ in the leaf node that $\boldsymbol{x}$ falls into. A special type of DT that only has a single node and thus only performs a single test is referred to as a decision stump. The inference process of a new data point $\boldsymbol{x}_i$ involves traversing the tree from the root until a leaf node with a class label c is reached which corresponds to executing "If *condition* then ..." rules for each node.

DTs can create overly complex trees that do not generalize well to new data as they tend to overfit on the training data [Pedr11], *cf.* Section 2.1.2. DT implementations such as *scikit-learn* thus offer additional hyperparameters to control and limit the complexity of the learned tree:

- *max_depth* controls the maximum depth *i.e.* maximum levels of decision nodes in the tree. If this parameter is not used, the tree will grow until the leaf nodes are pure, *i.e.* contain only samples of a single class or until one of the other, hyperparameter-controlled, conditions is met.
- *min_samples_split* controls the minimum amount of training samples required to generate a new decision node. If the amount of samples on an edge is below this parameter, a leaf node is generated instead of a decision node.
- *min_samples_leaf* similarly governs the creation of new decision nodes. If this parameter is set, a new decision node will only be generated if it produces edges or leaves that contain at least the configured amount of samples each. If this is not reached, a leaf node is generated instead of a decision node.

Ensemble learning As explained above, the DT can be an algorithm with inherently high variance, as it is able to strongly overfit on small variations in the training dataset, *cf.* Section 2.1.2. One popular way to reduce the variance of predictions and to generate more powerful models is to combine multiple models into an ensemble of models. In this context, bootstrap aggregating – bagging – refers to drawing m subsets of the training dataset $\mathcal{D}_{\text{train}}$ with replacement (bootstrapping) and training a separate model on each of them. During inference, the predictions of the models are combined (aggregated) by averaging the individual probabilities

$$p_{\text{bag}}(y = j \mid \boldsymbol{x}, \{(\boldsymbol{\theta}_l)\}_{l=1}^{m}) = \frac{1}{m}\sum_{l=1}^{m} p(y = j \mid \boldsymbol{x}, \boldsymbol{\theta}_l) \tag{2.20}$$

before applying Equation (2.1) or, alternatively, by using majority voting. $\boldsymbol{\theta}_l$ in Equation (2.20) corresponds to the learned parameters of model l inside the ensemble. The more uncorrelated the errors of the individual ensemble models are, the greater the performance benefit of using this method [Bish06; Murp12].

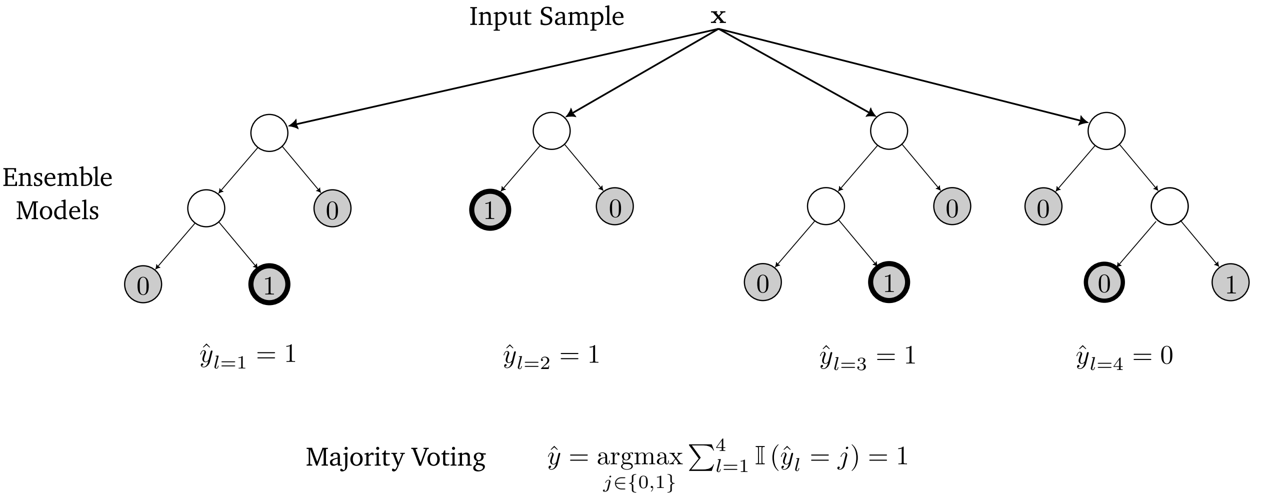


Figure 2.9.: Example visualization of a RF for a binary classification problem with $m = 4$ DTs as ensemble members. Decisions of the ensemble are combined via majority voting. Own illustration.

Random forests The RF is a popular algorithm that utilizes ensembles of m DTs. RFs use bagging on sample level and feature level. The individual models are therefore trained on a bootstrapped subset of the features as well as on a bootstrapped subset of the dataset samples. While RFs typically reach significantly higher performance levels compared to DTs, the prediction process is not easily interpretable anymore. In addition to the hyperparameters of the DTs in the RF ensemble, the number of trees m is an important additional hyperparameter of the RF. A visualization of a RF is shown in Figure 2.9.

Isolation forest Unlike RFs, which are primarily used for supervised classification tasks, the IF [Liu08] is a popular ensemble algorithm for anomaly detection [Agga17]. The IF consists of m isolation trees that are grown by randomly selecting a single feature A for each isolation tree and recursively selecting random split points within the value range of this feature until either

- all data points at the leaf nodes have the same values, or
- the leaf nodes only contain a single data point, or
- a predefined tree height limit is reached.

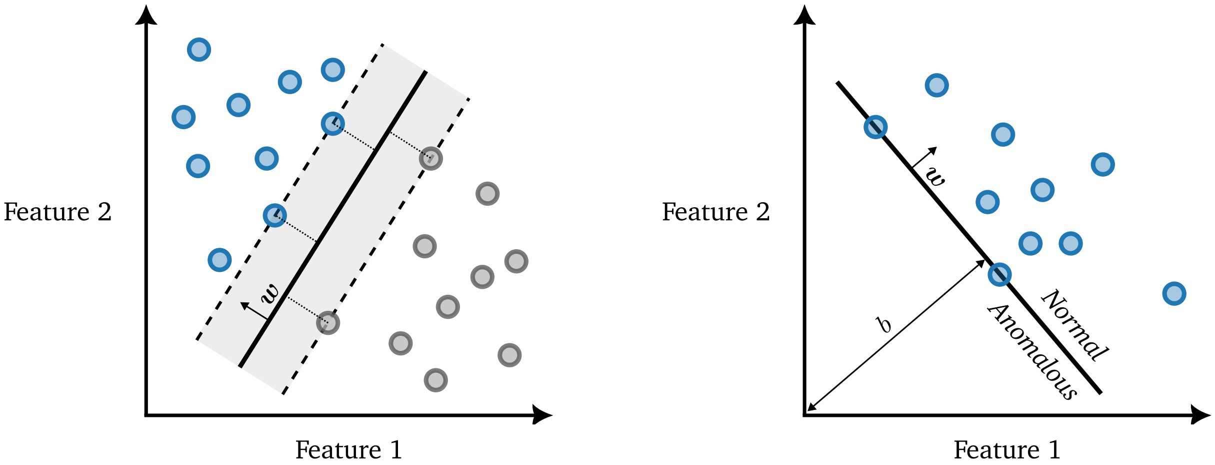
During inference, the path length from the root node to the terminating root in the isolation trees serves as a measure of normality for a given data point. The anomaly score for a data point $\boldsymbol{x}$ and a training dataset of size n_{train} is defined as

$$h(\boldsymbol{x}, n_{\text{train}}) = 2^{-\frac{E(l(\boldsymbol{x}))}{c(n_{\text{train}})}}, \tag{2.21}$$

where $E(l(\boldsymbol{x}))$ denotes the average path length $l(\boldsymbol{x})$ over all trees in the forest, and $c(n_{\text{train}})$ is a normalization factor that represents the average path length h for a dataset of size n_{train}, independent of the current data point. A high score h – approaching 1 – implies that the data point was isolated with fewer splits, indicating its significant deviation from the rest of the dataset. Such points are more likely to be anomalies. A lower score – near 0 – means that the data point required more splits to be isolated, suggesting that it behaves similarly to many other points in the dataset, and is therefore likely to be normal.

Support vector machines

In the following, two algorithms based on the SVM are introduced, the Support Vector Classifier (SVC) and the One-Class Support Vector Machine (OC-SVM).



(a) SVM for binary classification (SVC).

(b) OC-SVM for anomaly detection.

Figure 2.10.: Visualization of an SVM for binary classification and OC-SVM for anomaly detection applied to a two-dimensional dataset. In both (a) and (b), hard margin SVMs are visualized, with no slack variables, thus not allowing for misclassifications of the training data. Own illustration adapted from [Bish06; Agga17].

Support vector classifiers SVC is an algorithm for supervised binary classification that can be extended to multiclass classification. The fundamental idea behind SVMs in general and SVCs in particular is to find a hyperplane in the feature space that optimally separates the data points into binary classes. The SVC algorithm is used to find the hyperplane that maximizes the margin between the hyperplane and the nearest data points from either class, referred to as the support vectors, visualized as the dotted lines in Figure 2.10a. The support vectors influence the position and orientation of the hyperplane, represented by the equation

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - b = 0, \tag{2.22}$$

where $\boldsymbol{x}$ is an individual data point in the feature space, $\boldsymbol{w}$ is a weight vector and b is a bias term that defines the distance of the hyperplane from the origin. From Equation (2.22), the size of the margin can be expressed as $\frac{2}{\|\boldsymbol{w}\|}$ and is maximized in the learning process. As real-world data is typically not perfectly separable, SVCs employ two additional measures. First, the kernel trick is used. This technique involves transforming the input data into a higher-dimensional space where it is possible to find a linear separating hyperplane. Different types of kernels, like linear, polynomial, and Radial Basis Function (RBF), are used to perform this transformation. The objective of maximizing the margin can be expressed as the following quadratic programming problem with a training dataset $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n_{\text{train}}}$:

$$\min_{\boldsymbol{w},b,\xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + c\sum_{i=1}^{n_{\text{train}}} \xi_i, \tag{2.23}$$

$$\text{subject to } \; y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - b\right) \geq 1 - \xi_i, \;\; \xi_i \geq 0 \;\; \forall i \in \{1, \ldots, n_{\text{train}}\}.$$

Here, ξ_i are the so-called slack variables, which allow for some data points to lie on the wrong side of the hyperplane to maximize the margin and account for the effects of noise and other real-world phenomena that prevent perfect separation of the training data. The regularization parameter c serves as a trade-off between maximizing the margin and minimizing the classification error [Bish06]. The choice of kernel as well as the parameter c are the most important hyperparameters of the SVC.

For inference of a new data point $\boldsymbol{x}_i$ the SVC calculates the decision function

$$f(\boldsymbol{x}_i) = \boldsymbol{w} \cdot \boldsymbol{x}_i + b \tag{2.24}$$

and determines the class membership as

$$\hat{y}_i = \begin{cases} 1 & \text{if } f(\boldsymbol{x}_i) \geq 0, \\ 0 & \text{if } f(\boldsymbol{x}_i) < 0. \end{cases} \tag{2.25}$$

Different to the ML models that were presented so far, SVCs have no in-built mechanism for estimating class probabilities. If class probabilities are required for a given application, they are typically estimated depending on the distance of the data points to the hyperplane, e.g., using isotonic regression [Plat99].

One-class support vector machine The OC-SVM is a model for anomaly detection tasks, visualized in Figure 2.10b. In many aspects, OC-SVMs work similarly to SVCs. While SVCs focus on maximizing the margin between two classes, OC-SVMs aim to encapsulate the *normal* data points in the feature space. This is achieved by finding a hyperplane that separates the majority of the data from the origin with maximum margin, effectively isolating outliers or anomalies. In OC-SVMs the decision function is similar to that of SVCs, tailored to a one-class scenario:

$$\hat{y} = \begin{cases} +1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) - \rho \geq 0, \\ -1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) - \rho < 0, \end{cases} \tag{2.26}$$

where $\phi(\boldsymbol{x})$ is a function that maps the input data to a higher-dimensional space (often using the same kernel trick as described above for SVCs), and ρ is a threshold parameter. During training, the OC-SVM aims to solve the following optimization problem:

$$\min_{\boldsymbol{w},\rho,\xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{\nu n_{\text{train}}}\sum_{i=1}^{n_{\text{train}}} \xi_i - \rho, \tag{2.27}$$

$$\text{subject to } \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}_i) \geq \rho - \xi_i, \ \ \xi_i \geq 0 \ \ \forall i \in \{1, \dots, n_{\text{train}}\}.$$

Here, ν is a parameter that sets an upper bound on the fraction of outliers and a lower bound on the fraction of support vectors. This formulation essentially creates a soft boundary around the data, allowing for some flexibility in terms of what is considered an outlier, comparable to the slack variables in SVCs. Important hyperparameters of the OC-SVM are the choice of the kernel $\phi(\cdot)$ as well as ν [Agga17].

Neural networks and deep learning

The Artificial Neural Network (ANN), commonly referred to as NN, is an ML algorithm which originated from attempts to find mathematical representations of biological structures that process information in the brains of humans and animals [Bish06]. Within this subsection, the fundamental building blocks of NNs are explained and consequently more complex architectures such as Convolutional Neural Networks (CNNs) and Auto-Encoders (AEs) are introduced.

This subsection draws heavily from the books titled "*Deep Learning: Foundations and Concepts*" by Bishop et al. [Bish23] and "*Deep Learning*" by Goodfellow et al. [Good16].

Perceptron and multilayer perceptron Preceding today's NNs is the perceptron algorithm. It was introduced in 1958 as a model for understanding brain activities [Rose58]. The perceptron exhibits the most important attributes of the more complex NNs that are being used today. The perceptron takes a d-dimensional numerical input vector $\boldsymbol{x} \in \mathbb{R}^d$. Every scalar input x_i is multiplied with its respective weight w_i out of the weight vector $\boldsymbol{w} \in \mathbb{R}^d$. A bias term w_0 is added to the linearly combined inputs and weights as

$$a(\boldsymbol{x}, w_0, \boldsymbol{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d = w_0 + \sum_{i=1}^{d} w_i x_i. \tag{2.28}$$

The set of weights $\boldsymbol{w}$ and bias w_0 define the learnable parameters $\boldsymbol{\theta}$ of the perceptron. In order to achieve a clean notation, it is common to append the bias term w_0 to the 0-indexed position of $\boldsymbol{w}$ and in turn append a constant $x_0 = 1$ to $\boldsymbol{x}$ so that Equation (2.28) becomes

$$a(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=0}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}, \tag{2.29}$$

with $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^{d+1}$. The resulting value a is passed through an activation function $h(\cdot)$ to calculate the output value $\hat{y} = h(a)$. In case of the original perceptron, the activation function is a simple threshold function [Rose58]. While the capacity of the perceptron to model complex functions is limited, it can be significantly increased by adding multiple outputs and *hidden* layers. In this context, the neurons are called units that are connected in m layers. While multiple definitions exist, NNs with more than two layers of weights are commonly called *deep* NNs and the sub-field of machine learning that focuses on such networks is called deep learning [LeCu15; Bish23]. A Multilayer Perceptron (MLP) with a single hidden layer that takes as input the vector $\boldsymbol{x} \in \mathbb{R}^d$ and outputs the vector $\boldsymbol{y} \in \mathbb{R}^c$ is shown in Figure 2.11. The weights of a single layer in the MLP are defined as matrices $W^{(1)}, \cdots, W^{(m)}$. In each layer i, the outputs of the linear combination in Equation (2.29), referred to as pre-activations $\boldsymbol{a}^{(i)}$ are transformed to activations $\boldsymbol{z}^{(i)} = h(\boldsymbol{a}^{(i)})$ using an activation function $h(\cdot)$, similar to the perceptron. The activation function enables the MLP to model nonlinear functions. The choice of the activation function as well as the number of layers m and their respective number of units are important hyperparameters of the MLP. Common choices for activation functions are:

- The sigmoid function

$$\mathrm{sigmoid}(a) = \frac{1}{1 + \exp(-a)} \tag{2.30}$$

- The hyperbolic tangent function

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{2.31}$$

- The Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(a) = \max(0, a) \tag{2.32}$$

The pre-activations $\boldsymbol{a}^{(m)}$ of the final layer of the MLP are transformed using an appropriate output activation function $f(\cdot)$ to the vector of network outputs $\boldsymbol{y} = f(\boldsymbol{a}^{(m)})$. For a two layer MLP as visualized in Figure 2.11, the final activations are calculated as

$$\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}) = f\left(W^{(2)} h\left(W^{(1)}\boldsymbol{x}\right)\right). \tag{2.33}$$

The choice of the output activation function mainly depends on the type of task that should be performed by the NN, *i.e.* classification or regression, as the output activation function determines the value range of the output.

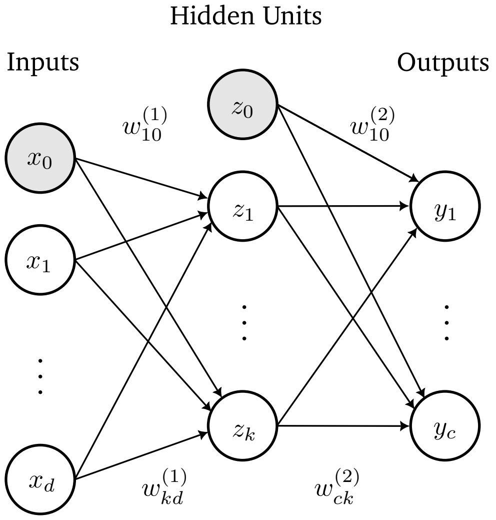


Figure 2.11.: Example visualization of a two-layer MLP with an input vector of size d, k hidden units and an output vector of size c. The hidden layers of the MLP are often called fully connected layers since every unit of a layer is connected to all of the units of the subsequent layer. As visualized in Figure 2.11, NNs exhibit a graph-like structure. This structure is often called the computational graph, the architecture, or the model of the NN. Own illustration adapted from [Bish23].

NNs used for classification tasks use output activation functions to normalize the numerical values of the last network layer to pseudo-probabilities that describe the likelihood of a given sample $\boldsymbol{x}$ belonging to a certain class j as a scalar score $p(y = j \mid \boldsymbol{x}) \in [0, 1]$. NNs that are used for binary classification tasks where $y_i \in \{0, 1\}$ typically use an output vector of size 1 and the sigmoid function, *cf.* Equation (2.30), as the output activation function. The resulting score is then treated as the likelihood of the sample belonging to the positive class (1). The likelihood of the negative class (0) is in turn inferred as

$$p(y=0 \mid \boldsymbol{x}) = 1 - p(y=1 \mid \boldsymbol{x}). \tag{2.34}$$

In case the model is used to perform multiclass classification, the softmax function can be used as the final activation function. The softmax function can be seen as the multiclass generalization of the sigmoid function [Good16] and originates from the Boltzmann distribution used in statistical mechanics [Jayn57]. The softmax function normalizes the c-dimensional network output over the discrete set of c classes so it can be treated as a probability distribution $\boldsymbol{y} : \mathbb{R}^c \rightarrow \left\{\boldsymbol{y} \in \mathbb{R}^c \mid \boldsymbol{y}_j > 0, \sum_{j=1}^{c} \boldsymbol{y}_j = 1\right\}$. The softmax score in the m-th layer of the NN for class j is computed as

$$\operatorname{softmax}\left(a_j^{(m)}\right) = p(y=j \mid \boldsymbol{x}) = \frac{e^{a_j^{(m)}}}{\sum_{i=1}^{c} e^{a_i^{(m)}}} \forall i, j \in \{1, \cdots, c\}. \tag{2.35}$$

Training of neural networks The NN is an algorithm that can be used for function approximation as described in Section 2.1.1. The NN is therefore a function with a set of parameters or weights $\boldsymbol{w}$ that accepts an input $\boldsymbol{x}$ and computes the output as $\hat{y} = \hat{f}(\boldsymbol{x}, \boldsymbol{w})$. In the context of NNs, this inference process is also referred to as forward pass and NN architectures that operate in this way are called feedforward networks, as information flows forward through the computational graph of the network. The objective of the learning process of an NN is to minimize a distance measure $\mathcal{L}(y, \hat{y})$ between the ground truth y and the prediction $\hat{y}$. In this context, $\mathcal{L}(y, \hat{y})$ is referred to as the loss function which mathematically describes the discrepancy between ground truth and prediction. In classification tasks, the Cross-Entropy (CE) is a widely used loss function [Bish23]. The CE loss for a binary classification task, commonly referred to as Binary Cross-Entropy (BCE), for a single sample is defined as

$$\mathcal{L}_{\text{BCE}}(y, \hat{y}) = -\left(y \log\left(\hat{y}\right) + (1-y) \log\left(1-\hat{y}\right)\right), \tag{2.36}$$

where $y \in \{0, 1\}$ represents the true label and $\hat{y} \in [0, 1]$ is the predicted probability. The CE is a measure for the difference between distributions and closely related to the Shannon Entropy, *cf.* Equation (2.17). The BCE loss in Equation (2.36) approaches zero as the predicted probability $\hat{y}$ closely aligns with the true label y. Specifically, the loss decreases as $\hat{y}$ approaches 1 when $y = 1$, or as $\hat{y}$ approaches 0 when $y = 0$, reflecting a lower penalty for predictions that closely match the actual labels. Optimization of the CE-loss thus pushes the network output to be close to the extremes of its value range [Bish23]. For the multiclass case, the CE loss for a single sample is defined as:

$$\mathcal{L}_{\text{CE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{c} y_i \log\left(\hat{y}_i\right), \tag{2.37}$$

where $\boldsymbol{y}$ typically is a one-hot-encoded ground truth vector, *cf.* Equation (2.11). Similar to before, the multiclass CE loss in Equation (2.37) is minimized if the distribution of estimated class probabilities $\hat{\boldsymbol{y}}$ match the ground truth vector $\boldsymbol{y}$. The loss is aggregated over the respective n samples

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{\text{CE}}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i). \tag{2.38}$$

NNs for regression tasks, where $y \in \mathbb{R}$ is scalar, typically employ the Mean Squared Error (MSE) loss

$$\mathcal{L}_{\text{MSE}}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{2.39}$$

Popular optimization algorithms for NNs are based on Gradient Descent (GD) [Good16; Bish23; Zhan23]. During the training process, the weights $\boldsymbol{w}$ of the NN are optimized iteratively using the gradient of the loss function, calculated for the whole dataset or parts of it. Arguably the most popular algorithm for gradient-based optimization of NNs is Stochastic Gradient Descent (SGD) [Good16; Bish23; Zhan23]. In each iteration τ of SGD, the current weights $\boldsymbol{w}^{(\tau-1)}$ of the NN are updated as

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{(\tau-1)}), \tag{2.40}$$

where $\nabla_{\boldsymbol{w}} J(\boldsymbol{w})$ is the estimated gradient of the loss function L with respect to the weights. The gradient of the loss function is used to determine the influence of the specific weights on the loss and is calculated by the backpropagation algorithm. This algorithm recursively uses the chain rule of calculus to derive the gradients with respect to specific weights going backward in the computational graph of the network [Bish06]. Thus, this step is called the *backward pass*.

In contrast to classic GD, the loss function and respective gradients in SGD are not calculated on the entire training set but rather estimated using random subsets of the training set referred to as batches with b samples in each batch, commonly referred to as batchsize:

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \frac{1}{b} \sum_{i=1}^{b} \nabla_{\boldsymbol{w}} L\left(\hat{y}_i, y_i\right). \tag{2.41}$$

A full iteration of the training set corresponds to $\lfloor \frac{n_{\text{train}}}{b} \rfloor$ batch forward and backward passes, which is called an epoch. η is a hyperparameter referred to as the Learning Rate (LR) as it controls the magnitude of the parameter updates in each iteration. The batch size, number of epochs or choice of stopping criterion as well as the LR are important hyperparameters that control the learning process of NNs.

Multiple extensions to SGD exist. Notably, momentum tries to solve the problem of the learning process converging in local minima. Momentum achieves this by accumulating the previous gradient steps

$$\boldsymbol{v}^{(\tau)} = \alpha \boldsymbol{v}^{(\tau-1)} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{(\tau-1)}), \tag{2.42}$$

and consequently using this accumulation for the update step [Good16]:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \boldsymbol{v}^{(\tau)}. \tag{2.43}$$

Additionally, momentum effectively reduces the noise introduced by using batches for gradient estimation in SGD. The optimizer used for all experiments involving neural networks in this thesis is the Adaptive Moments Estimation (ADAM) optimizer [King14] which is a further extension of SGD and momentum. ADAM calculates an adaptive LR for each parameter of the network individually by tracking the first and second-order momentum.

Convolutional neural networks NNs that are used for computer vision applications today mainly rely on a special type of NN architecture called the CNN [Bish23]. The CNN architecture, as it is used today, was first introduced in 1989 by Le Cun et al. for the classification of handwritten digits [LeCu10], *cf.* Figure 2.12.

CNNs are particularly suited for image data as they exploit important properties of images such as nearby pixels being more strongly semantically correlated than distant pixels [Bish06]. Furthermore, CNNs enable NN architectures that are invariant to certain kinds of transformations such as variations in the position of objects within images. This property is important for tasks such as image classification. For example, the position of a

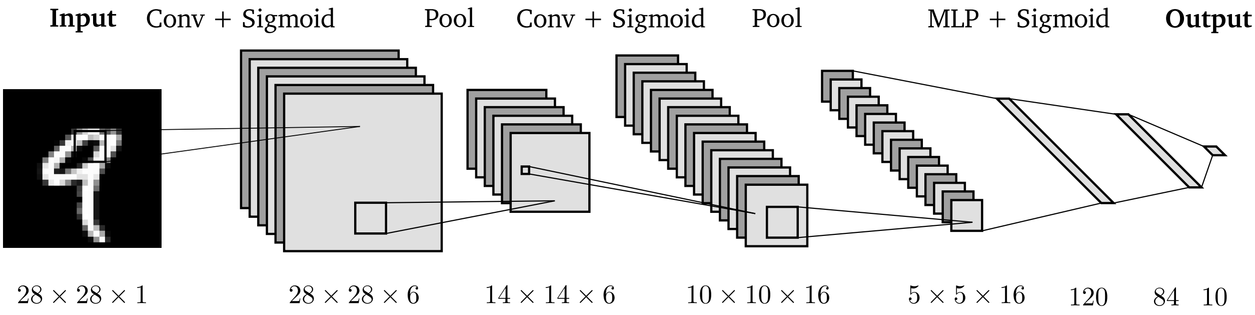


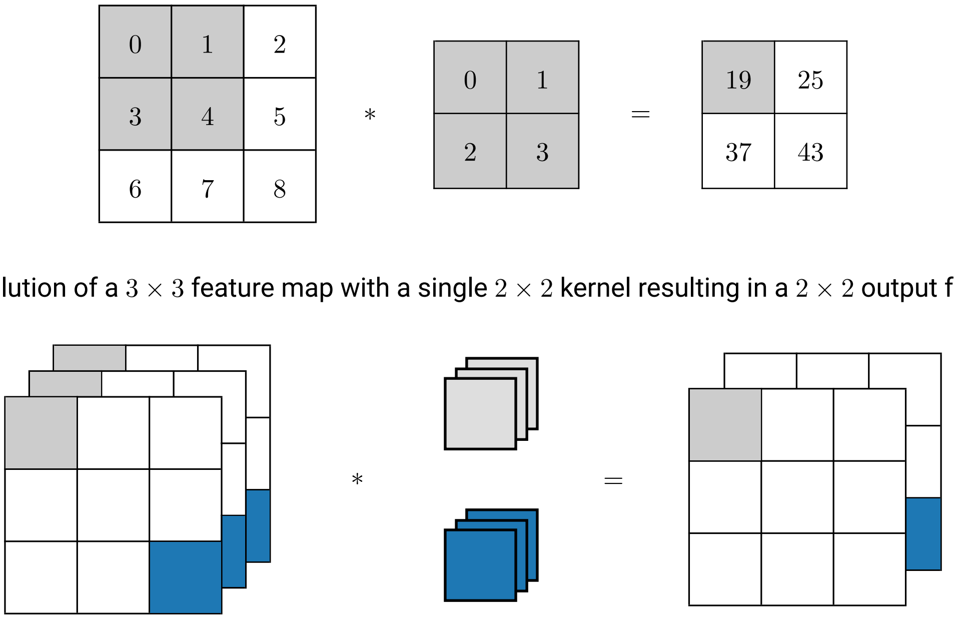
Figure 2.12.: A simple CNN architecture for handwritten digit classification from the MNIST dataset [LeCu10] with $c = 10$ classes representing digits $0, ..., 9$ (LeNet [LeCu98]). LeNet uses 5×5 convolutional kernels and 2×2 average pooling layers. The penultimate layers before the output classification are referred to as the learned embeddings. Own illustration in part generated using [LeNa19].

dog in an image should not matter in classifying whether the image is showing a dog or not. Moreover, CNNs implement a weight-sharing mechanism through kernels which lowers the model capacity and computational load significantly when compared to image processing with MLPs which is typically infeasible. Although CNNs find applications in a broad spectrum of computer vision tasks, including semantic segmentation [Long15], object detection [Redm16], and depth estimation [Ming21], the scope of the investigations in this thesis is confined to their use in image classification. Consequently, the architectural explanations and analyses presented herein are tailored to this particular task. CNNs became popular for the task of image classification in 2012, when *AlexNet* won the ImageNet challenge, significantly outperforming traditional methods [Kriz12]. CNN architectures for classification commonly consist of three components: Convolutional layers, pooling layers and a final classification module that is typically an MLP.

Convolutional layer The input of a CNN is typically a 3-dimensional tensor $\boldsymbol{X} \in \mathbb{R}^{w \times h \times d}$ where h is the height, w the width and d the depth, or number of channels of the input. In the case of images, $d = 3$ typically represent the Red Green Blue (RGB) color matrices. Features within intermediate layers are referred to as feature maps [Bish06]. In contrast to the MLP, the layers in a CNN are not fully connected but represented by kernels $\boldsymbol{K} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ that are swiped over the previous feature maps, performing a discrete convolution operation. A simple example regarding a convolution operation with one channel without nonlinearity is shown in Figure 2.13a, and a multichannel example is shown in Figure 2.13b.

In the convolution operation, the kernel weights are linearly combined with the feature values of the patch of the input feature map. Similar to the MLP, CNN network architectures commonly include a nonlinearity as the activation of the linear combination of the kernel and the feature map. Intuitively, a kernel can be thought of as a detector that reacts to specific patterns of features. Swiping the kernel across the feature map allows the network to be invariant to the location of specific patterns. A convolutional layer usually includes multiple kernels. The spatial dimensions of a kernel constitute the local receptive field of this layer. Stacking convolutional layers increases the size of the resulting receptive field, enabling the CNN to capture complex patterns hierarchically.

The convolutional kernel is swiped over the input feature map with a displacement between each patch that is called the stride. The stride is an important hyperparameter as it controls the size reduction by the convolution operation. Depending on the size of the kernel, the stride and the size of the input feature map, the spatial size of the feature map may be extended by a padding. Padding enlarges the input feature map by adding values around the borders of the feature map. Commonly, zero padding is used which extends the feature



(a) 2D convolution of a 3×3 feature map with a single 2×2 kernel resulting in a 2×2 output feature map.

(b) Convolution of a $3 \times 3 \times 3$ feature map with two 1×1 kernels resulting in a $3 \times 3 \times 2$ output feature map.

Figure 2.13.: Examples for convolution operations on a single channel input with a single kernel (a) and a 3-channel input with two kernels (b). Own illustration adapted from [Zhan23].

map borders by zeros. Due to the use of kernels, a convolutional layer significantly reduces the number of weights compared to a fully connected layer.

Pooling layer Pooling operations reduce the size of feature maps by summarizing subregions [Dumo16]. This subsampling is done by applying a pooling function to patches of the feature map. The majority of popular CNN architectures use either maximum or average pooling [Bish23]. This reduces the input patch to the maximum value or the average value respectively. Average pooling smooths the feature map while max-pooling only passes the highest activation of the input feature map. While convolutional and pooling layers are input size agnostic, the attached classification MLP typically forces a fixed input size.

The possible applications of CNNs are not limited to image data and generally include structured data that has a known grid-like topology [Good16; Bish23]. Especially in industrial and speech recognition applications, CNNs have proven to be effective in processing image-like data originating from time series, such as spectrograms and scalograms [Abde14; Wang17; Bieg23]. See Section 2.6 for explanations regarding the data types and Sections 2.5.1 and 2.5.2 for applications.

Autoencoder The AE is a special type of NN that is trained to copy its input to its output [Good16]. The underlying idea of an AE is to reconstruct its own input $\boldsymbol{x} \in \mathbb{R}^d$ after compressing it to a lower dimensional latent space $\boldsymbol{z}(\boldsymbol{x}) \in \mathbb{R}^m$, also referred to as a bottleneck. AEs consist of two components, the encoder network, which compresses the input into the latent space and the decoder, which reconstructs the input from the latent space. As the latent space has a lower dimensionality $m < d$ than the input layers of the AE, the AE has to learn an efficient compression algorithm to be able to reconstruct the data from the latent space. The

MSE loss, *cf.* Equation (2.39), is typically used for training AEs with continuous input/output space so the minimization objective becomes

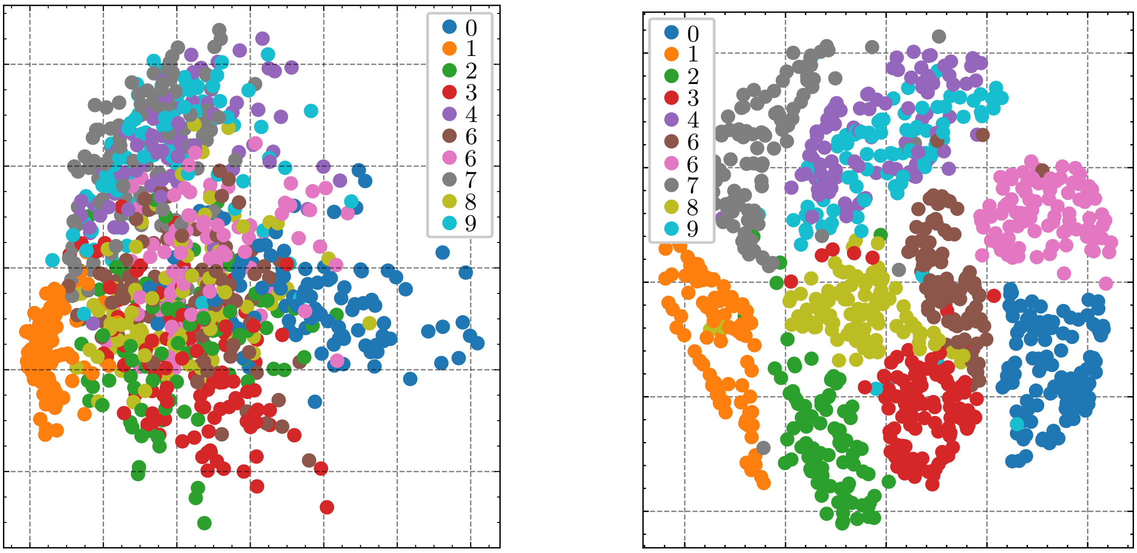
$$\mathcal{L}_{\mathrm{AE}}(\boldsymbol{x}, \hat{\boldsymbol{y}}(\boldsymbol{x})) = \frac{1}{n}\sum_{i=1}^{n} \left\| \hat{\boldsymbol{y}}(\boldsymbol{x}_i) - \boldsymbol{x}_i \right\|^2 . \tag{2.44}$$

AEs can be built out of fully connected layers / MLPs or convolutional layers. In the case of convolutional layers, the resulting architecture is typically referred to as a Convolutional Auto-Encoder (CAE). While the encoder in a CAE consists of convolutional and pooling layers as introduced above, the decoder is built out of transpose convolutional layers and upsampling layers that commonly use nearest-neighbor or bilinear upsampling [Bish23].

Use cases for AEs include dimensionality reduction, representation learning and anomaly detection [Bish23]. Typically, the reconstruction performance of an AE degrades if the input data distribution deviates from the training data, which is exploited when AE are used in anomaly detection tasks [Garc22]. In this sense, the reconstruction error $\mathcal{L}$ is used as the anomaly score, *cf.* Section 2.1.1.

Visualization techniques

Within this subsection, two ML-based visualization techniques are introduced, which are utilized to depict data distributions throughout the thesis.



(a) Dimensionality reduction using Principal Component Analysis (PCA).

(b) Dimensionality reduction using t-SNE.

Figure 2.14.: Examples of dimensionality reduction techniques applied to the MNIST handwritten digit dataset [LeCu10], *cf.* Figure 2.12. As the samples in MNIST contain complex, nonlinear relationships between the features, t-SNE is able to capture the data-inherent clusters in a more distinctive way than PCA. Own illustration.

Principal component analysis PCA is an unsupervised algorithm used for applications such as dimensionality reduction with the purpose of feature extraction and data visualization as well as anomaly detection [Joll02; Bish06; Bieg23]. PCA is the orthogonal projection of data onto a lower dimensional space, known as the principal subspace. This projection (1) maximizes the variance of the projected data and (2) minimizes the average reprojection cost $\mathcal{L}$ defined as the MSE, *cf.* Equations (2.39) and (2.44), between the original data points and their projections

$$\mathcal{L}_{\text{PCA}} = \frac{1}{n}\sum_{i=1}^{n} \|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|^2 = \left\|WZ^T - X\right\|_F^2 \tag{2.45}$$

where $\hat{\boldsymbol{x}}_i = W\boldsymbol{z}_i$ is the reprojection of sample $\boldsymbol{x}_i$ in $X \in \mathbb{R}^{n\times d}$ out of the corresponding score $\boldsymbol{z}_i = W^T\boldsymbol{x}, \boldsymbol{z}_i \in \mathbb{R}^m$ using m linear basis vectors $\boldsymbol{w}_j \in \mathbb{R}^d$ [Pear01]. The optimal solution minimizing Equation (2.45) is obtained by setting $W = V_m$ where V_m contains the m eigenvectors with the largest eigenvalues of the empirical covariance matrix

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i\boldsymbol{x}_i^T. \tag{2.46}$$

Importantly, the input data X must be standardized, *cf.* Equation (2.12), before applying PCA as it is sensitive to feature scales. The scores $Z \in \mathbb{R}^{n\times m}$ contain the projected samples in lower dimensional space $m < d$. For visualization, m is typically set to $m = 2$, so the data can be plotted in two-dimensional scatter plots, as visualized for an example dataset in Figure 2.14a. For anomaly detection, the reprojection error $\mathcal{L}$ can be monitored, similar to AEs. The main constraint of PCA is its limitation to linear projections which inhibits finding dependencies in real datasets which often contain nonlinear dependencies [Bish06]. The projection and reprojection of PCA is comparable to a linear autoencoder [Good16]. In the context of this thesis, PCA is used for dimensionality reduction with the goal of data distribution visualization.

t-distributed stochastic neighbor embedding t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised algorithm used for visualizing high-dimensional nonlinear data [Van 08]. t-SNE starts by converting the high-dimensional (d-dimensional) Euclidian distances between data points in the dataset into conditional probabilities that represent similarities between data points. The similarity of a data point $\boldsymbol{x}_i$ to another data point $\boldsymbol{x}_j$ is given by a conditional probability $p_{j|i}$ which is proportional to the probability that $\boldsymbol{x}_i$ would pick $\boldsymbol{x}_j$ as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at $\boldsymbol{x}_i$ [Van 08]. This can be expressed as

$$p_{j|i} = \frac{\exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2/2\sigma_i^2)}, \tag{2.47}$$

where σ_i is the variance of the Gaussian centered on data point $\boldsymbol{x}_i$. The value of σ_i is chosen such that it corresponds to a perplexity, which is a hyperparameter reflecting the effective number of local neighbors of each point. Based on the conditional probabilities, the joint probability is defined as $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$. In the lower-dimensional (m-dimensional) space, t-SNE calculates probabilities q_{ij} using a Student's t-distribution, aiming to preserve the local structure of the data:

$$q_{ij} = \frac{(1 + \|\boldsymbol{z}_i - \boldsymbol{z}_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|\boldsymbol{z}_k - \boldsymbol{z}_l\|^2)^{-1}}, \tag{2.48}$$

where $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ are the low-dimensional embeddings of high-dimensional points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, respectively. The goal of t-SNE is to minimize the Kullback-Leibler (KL) divergence between the two distributions P and Q, which quantifies the difference between the high-dimensional and low-dimensional representations:

$$KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{2.49}$$

This minimization is typically achieved through gradient descent, where the positions of points in the low-dimensional space are iteratively adjusted. The optimization process is sensitive to the initial configuration and the choice of perplexity and is usually run for a fixed number of iterations or until convergence. t-SNE excels in revealing the underlying structure of the data, particularly in forming distinct clusters. However, due to its nonlinear and iterative nature, t-SNE does not offer an inverse transformation from the low-dimensional embedding back to the high-dimensional space and can thus not be used for anomaly detection which stands in contrast to PCA. Similar to PCA, t-SNE is used within this thesis for dimensionality reduction with the goal of visualizing high-dimensional datasets in 2D scatter plots, with an example shown in Figure 2.14b.

2.3. Practical aspects of developing machine learning applications

While the previous sections focus on the theoretical background of ML, this section is concerned with the practical aspects of implementing ML-based applications in the real world, introducing several central themes of this thesis. First, the phenomena of data distribution shifts in general and concept drift in particular are explained, which are of great relevance for the practical application of ML models. Consequently, process models are introduced, which are a tool for structuring ML projects in industrial applications.

2.3.1. Data distribution shifts

In Section 2.1.2, the concept of generalization was presented. ML algorithms are employed to model the underlying data generating function that produced the datasets that are used for training and testing [Bish06]. One of the central goals in model selection and assessment is to ensure that the trained model leverages the knowledge gained from the training process to generate accurate predictions on unseen data. Performance metrics, *cf.* Section 2.1.2, are used to estimate the error on unseen data, using a hold-out test set that is typically randomly split from the whole set of available data. Both the training and the performance estimation are done under the central assumption that the training and testing data used in development come from the same, or very similar, data distribution as the model will be exposed to in production after deployment [Huye22]. In the long term, however, this assumption is commonly violated, as is elaborated for manufacturing use cases in Section 3.1.

Following the book "*Designing machine learning system*" by Huyen [Huye22], this is generally due to two main reasons. First, training data is finite and constrained by the time and effort that was put into recording the dataset, while real-world data is multifaceted and theoretically infinite in its possible variations. Thus, the training data can only capture a subset of the data distribution that might occur after deployment. Second, the real world is not stationary, which is the main theme of this thesis and visualized in Figure 2.15. A popular example of non-stationarity that affected many ML systems such as demand forecasting or product recommendation systems is the COVID-19 pandemic. Non-stationarity is not only caused by extreme, unusual events but is rather common in ML practice, e.g., seasonal temperature variations [Ditz15; Huye22; Vela22].

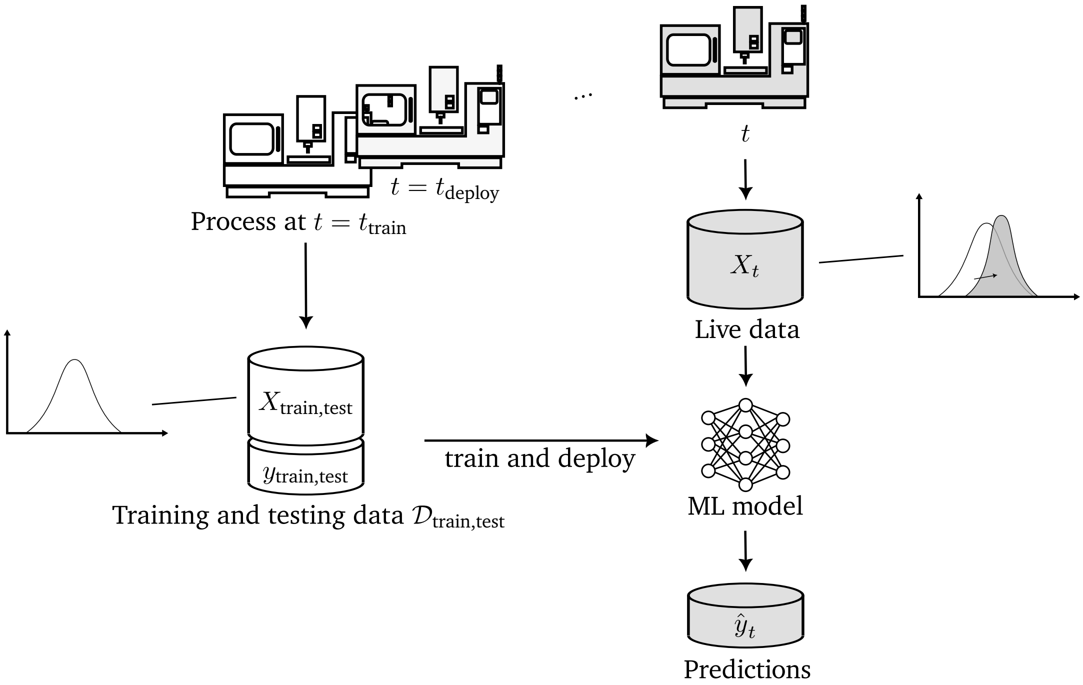


Figure 2.15.: Manufacturing process, data distributions and ML model over time. Models are typically trained on a static dataset but the process may evolve over time, changing the data distribution acquired from it. Own illustration.

Definitions According to [Gama14; Webb16], a concept in the context of supervised ML is defined as the joint probability distribution $P(X, Y)$ of input features X and target labels Y. In the unsupervised case, the concept is reduced to the prior probability distribution over the input features $P(X)$. Still, a joint distribution $P(X, Y)$ might exist, if (unknown) labels are assumed to exist for the data, such as *normal* vs *anomalous*, *cf.* Section 2.1.1. As such, concepts describe the observations of the underlying data-generating function that shall be modeled using ML. Especially in the context of real-world applications, concepts may change over time [Webb16], which makes the concept dependent on a given time t as $P_t(X, Y)$. Data distribution shifts can thus generally be defined as differing concepts over time:

$$P_{t=t_1}(X, Y) \neq P_{t=t_2}(X, Y). \tag{2.50}$$

Oftentimes, t_1 manifests as the training and testing dataset that was used in model development, while t_2 may be a point in time after the deployment of the model, thus during productive use. This specific kind of shift is often referred to as dataset shift [Žlio10b; Baie21]. The underlying data-generating function as well as the observation model (e.g. sensors) are not directly available to the ML model which renders detection of changes as in Equation (2.50) non-trivial [Widm96; Ditz15]. The joint distribution $P(X, Y)$ can be decomposed in two ways [Huye22]:

$$P(X, Y) = P(Y \mid X)P(X), \tag{2.51}$$

and as

$$P(X, Y) = P(X \mid Y)P(Y). \tag{2.52}$$

$P(Y \mid X)$ denotes the conditional probability of labels given the input data, which is typically what ML-algorithms are used to model, *cf.* Section 2.1.1, also referred to as discriminative learning. $P(X \mid Y)$ describes the conditional probability of input data given the labels. ML-algorithms used to model this dependency are typically referred to as generative models.

Depending on the decompositions in Equations (2.51) and (2.52), the following terminologies are defined [Gama14; Huye22], which are visualized in Figure 2.16:

- **Covariate shift**, visualized in Figure 2.16b, describes a type of distribution shift where $P(X)$ changes but $P(Y \mid X)$ remains the same, referring to the first decomposition of the joint distribution. Consider an ML application for the prediction of product quality based on process measurements such as temperatures, pressures and forces. Covariate shift could manifest, e.g., when temperature sensors measure on average slightly higher temperatures in the summer than in the winter without affecting the resulting product quality. The prevalence of specific values in the feature space thus changes. Even if this change does not influence $P(Y \mid X)$, it might still degrade model performance as the model encounters input distributions it is not optimized for.

- **Label shift**, visualized in Figure 2.16c, describes a type of distribution shift where $P(Y)$ changes but $P(X \mid Y)$ remains the same, referring to the second decomposition of the joint distribution. In the manufacturing example from before, this shift could correspond to a difference in the rate of quality problems between the training and testing data. While the training data might be artificially enriched with quality problems to train the algorithms, failures in actual production are rare, thus a label shift is present between training and testing data distributions. Similar to covariate shift, label shift can degrade the model performance as it can result in a shift towards a class where the proportional performance levels are lower.

- (Pure) **Concept drift**, visualized in Figure 2.16d, describes a type of distribution shift where $P(Y \mid X)$ changes but $P(X)$ remains the same, referring to the first decomposition of the joint distribution. Concerning the manufacturing example from before, concept drift might manifest if the sensors used to gather the input data of the ML model start to drift or get uncalibrated. As a consequence, the mapping from sensor data values to product quality outcome changes, invalidating the function approximation that was learned by the ML model, resulting in potentially degraded performance.

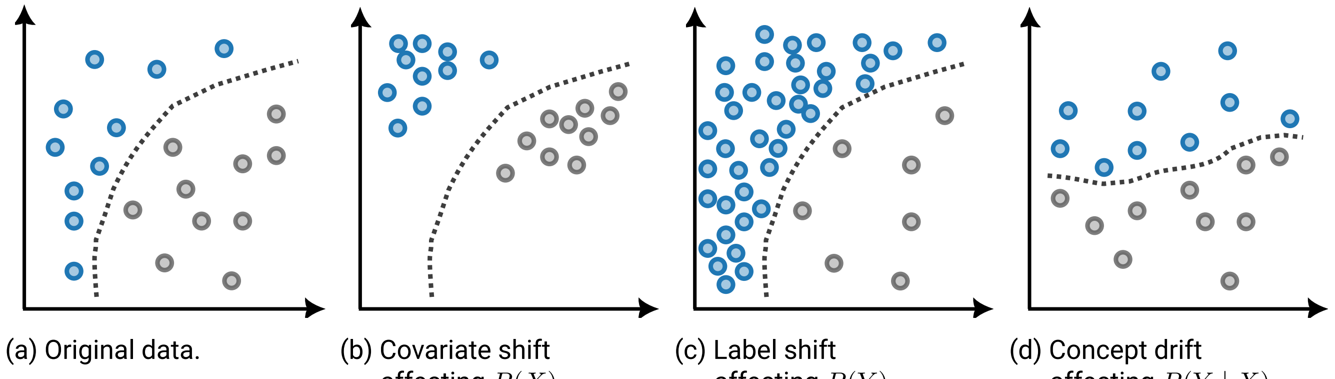

(a) Original data.
(b) Covariate shift affecting $P(X)$.
(c) Label shift affecting $P(Y)$.
(d) Concept drift affecting $P(Y \mid X)$.

Figure 2.16.: Categorization of distribution shifts, depending on which parts of the joint distribution $P(X, Y)$ changes visualized for two-dimensional example data with two classes blue and grey with the dashed line representing the decision boundary of a classifier. Own illustration adapted from [Gama14].

The usage of these terms in literature is often inconsistent and various additional terminologies exist such as real drift and virtual drift [Gama14; Lu18]. However, since concept drift is typically no isolated phenomenon but occurs in conjunction with both covariate and label shift, the majority of reviewed publications refer to the term concept drift when describing the general shift in the joint distributions between two points in time as stated in Equation (2.50) [Widm96; Gama14; Žlio16; Lu18; Baie21]. Thus, the same definition of concept drift is adapted for this thesis:

Definition 2 (Concept drift) *Concept drift refers to the phenomenon of the joint distribution between input data X and target variable Y changing over time, cf. Equation* (2.50).

Literature further distinguishes different types of concepts depending on their temporal structure [Žlio10b; Gama14] as visualized in Figure 2.17, which are also observed in empirical studies such as [Vela22]. In the manufacturing context, sudden and gradual drift may be caused by sensor failure or slow degradation respectively, while incremental and recurring drift may be caused by changes between batches, materials or product variants, among other possible causes.

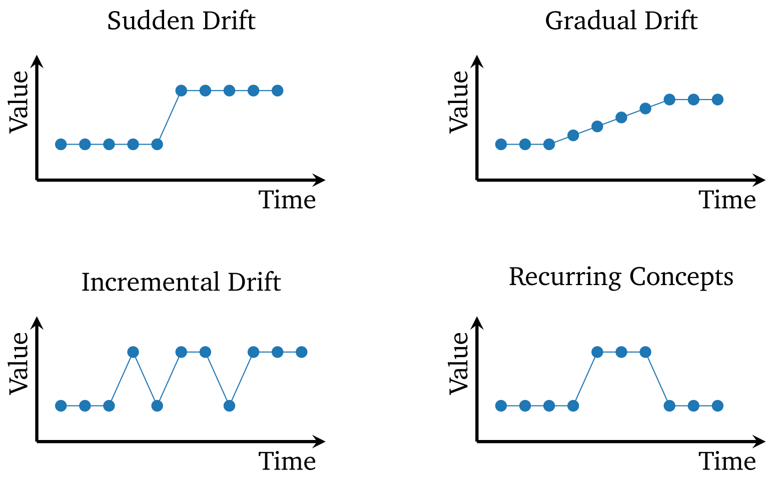


Figure 2.17.: Categorization of distribution shifts according to their behavior over time. The visualization shows a univariate data stream that transitions - *drifts* - between two discrete states. Own illustration, adapted from [Žlio10b; Gama14].

Handling concept drift Concept drift in ML is a critical issue that substantially impacts the deployment and long-term usability of models in real-world applications as it can lead to a gradual or abrupt decline in the performance of predictive models, when the assumptions under which these models were originally trained no longer hold true, *cf.* Figure 1.1 [Gama14; Ditz15; Huye22; Vela22]. A recent, large-scale study showed that performance degradation of deployed ML models is a nearly universal problem [Vela22]. Mechanisms for monitoring deployed ML models are thus required to detect distribution shifts and accompanying performance degradation to trigger the adaptation of models to a changing environment and guarantee reliability. Updating models can be done based on a multitude of triggers such as time-based, performance-based or data-based, depending on the particularities of the ML use case [Ditz15; Huye22]. The selection of a suitable trigger for model updates in manufacturing use cases is a central theme of this thesis which is investigated in Chapter 4. With respect to Figure 2.17, research into concept drift adaption for the first three types typically focuses on how to recognize and adapt to the drift in the fastest possible way, while research concerning recurring concepts emphasize the use of historical concepts and corresponding ML models that shall be matched [Lu18]. In the context of model adaptation, two ML paradigms are of importance and are briefly introduced:

- **Continual learning** Also referred to as lifelong learning, continual learning is a special case of sequential learning, *cf.* Section 2.1.1, which aims at continually accumulating knowledge and experience over time by steadily adapting a model. Typically, continual learning assumes an online learning scenario, where training data including target labels is received incrementally. As continual learning tries to maintain an existing model over time and adapt it to changing concepts, one of the main challenges is retaining past knowledge [Terc22a; Sari23]. This challenge is often referred to as catastrophic forgetting. Methods and literature in the realm of continual learning investigate how models can be assimilated to new concepts without losing the ability to perform well on past data. Concept drift detection plays a key role in continual learning, where it is used to trigger the adaptation of the model once a new concept is detected [Huye22].

- **Transfer learning** Techniques in the realm of transfer learning aim at leveraging knowledge acquired by an ML model in a certain task to improve or expedite learning in a related but different task or domain. This approach is rooted in the observation that certain features or patterns learned by a model in one context can be relevant and beneficial in another, thus reducing the need for extensive data collection and training from scratch for each new task [Huye22]. Specific techniques used in transfer learning are among others, fine-tuning and domain adaptation. Fine tuning of NNs involves performing gradient-based optimization of (typically) the weights of the last layers of a trained NN with respect to data of a new task while keeping the weights of early layers constant [Bish23]. Domain adaptation on the other hand aims at adapting trained ML models to distribution shifts, where the task stays the same by learning domain-invariant representations of the data [Wils20].

More generally, continual learning aims at adapting a model to new tasks or domains while retaining the performance on previously experienced tasks and domains, while transfer learning is typically only concerned with maximizing performance on the new task or domain [Masc23].

2.3.2. Process models and MLOps

Several process models exist that aim to guide practitioners through the complex process of structuring and conducting data science and ML projects within industrial contexts. In this section, two commonly employed process models are introduced, the Cross-Industry Standard Process for Data Mining (CRISP-DM) and the more recent Cross-Industry Standard Process Model for the Development of Machine Learning Applications with Quality Assurance Methodology (CRISP-ML(Q)).

CRISP-DM One of the most popular [Kurg06; Piat14] frameworks for structuring data science projects in industrial contexts is the CRISP-DM proposed in 2000 [Wirt00]. Developed by an industry consortium, CRISP-DM provides a generic stage model which divides data science projects into six cyclical phases as visualized in Figure 2.18.

- **Business understanding** This phase focuses on understanding the project objectives and requirements from a business perspective. The aim is to convert business objectives into data mining problem definitions and prepare a preliminary strategy to achieve the objectives.

- **Data understanding** The data understanding phase starts with an initial data collection and proceeds with activities to get familiar with the data, identify data quality problems, discover first insights into the data, or detect interesting subsets to form hypotheses for hidden information.

- **Data preparation** The data preparation phase covers all activities needed to construct the final dataset from the initial raw data. Tasks include table, record, and attribute selection, as well as transformation and cleaning of data for modeling tools. The data is cleansed, formatted, and structured in a way that is suitable for the chosen modeling technique.
- **Modeling** In this phase, various modeling techniques are selected and applied, and their hyperparameters are calibrated to optimal values. Techniques may include ML algorithms, statistical methods, or other data analysis techniques.
- **Evaluation** Once a model (or models) has been built, it is important to thoroughly evaluate it and review the steps executed to construct the model to be certain it properly achieves the business objectives. This phase involves assessing the model and evaluating its performance based on domain-specific success criteria. The key objective is to determine if there are any important business issues or technical aspects that have not been sufficiently considered.
- **Deployment** The deployment phase involves the actual implementation of the model into the business environment. This could mean implementing an application into a production environment, initiating actions based on the model's outcomes, or simply handing over the results to the client. Depending on the requirements, the deployment phase can be as simple as generating a report or as complex as implementing a repeatable data mining process across the organization.

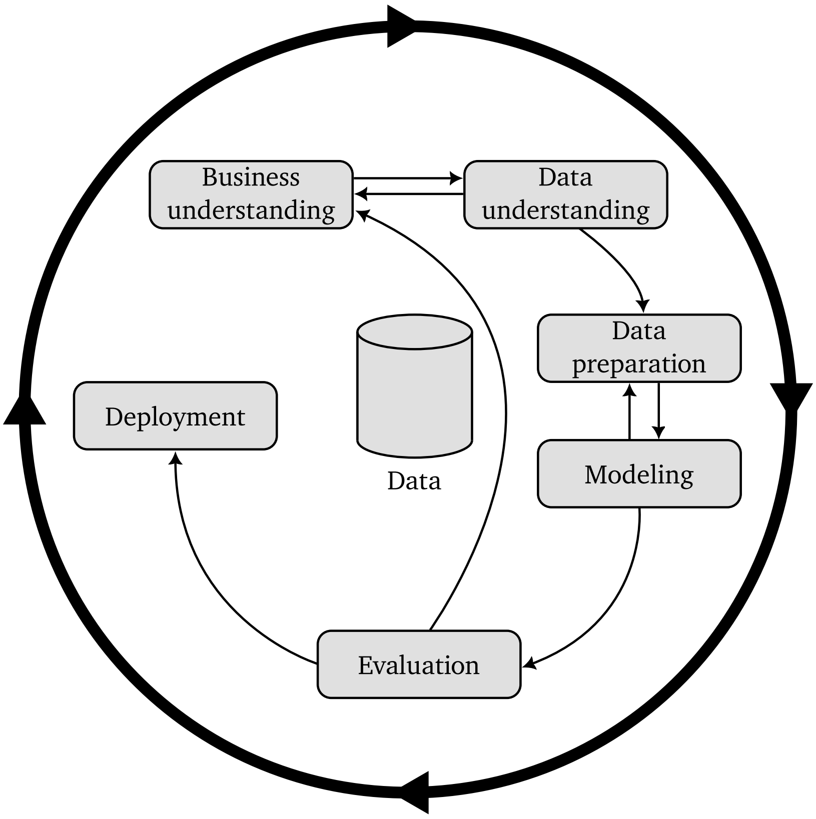


Figure 2.18.: The phases of the CRISP-DM process model. Own illustration adapted from [Wirt00].

As CRISP-DM is not specific to any industry or problem category, the guidance provided is rather abstract even at the lowest, most specific hierarchy level of the model. Furthermore, CRISP-DM was originally proposed for data mining projects, not necessarily the development and deployment of ML applications that should have the capability of reliably inferring decisions over a long period of time, which exhibits special challenges such as concept drift that are not considered in CRISP-DM [Žlio16; Stud21].

CRISP-ML(Q) More recently, [Stud21] proposed the CRISP-ML(Q) model as a successor to CRISP-DM, addressing the shortcomings and accounting for the differences of building ML applications when compared to conducting data mining projects. The changes between CRISP-DM and CRISP-ML(Q) are visualized in Figure 2.19.

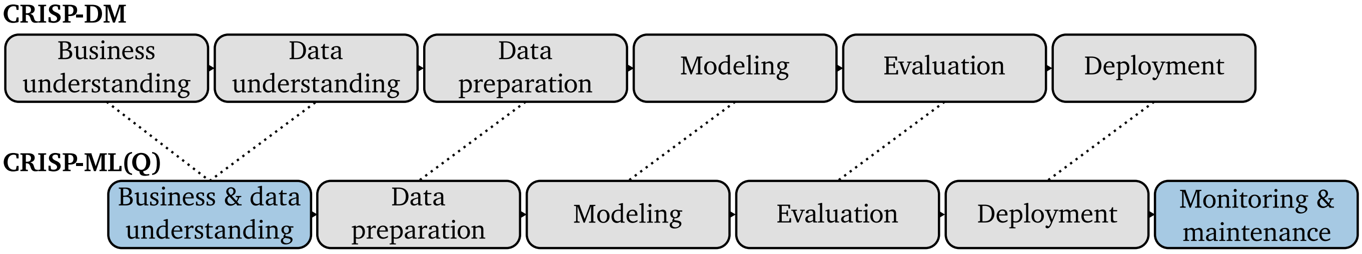


Figure 2.19.: Comparison between the phases of the CRISP-DM and CRISP-ML(Q) process models. Changed or added phases in CRISP-ML(Q) are highlighted in blue. Own illustration.

CRISP-ML(Q) shifts the focus from data mining processes to the full lifecycle of ML-based applications and introduces an explicit phase for **monitoring[2] and maintenance** of the ML model after deployment. The monitoring part of this phase is concerned with identifying non-stationary data distributions, hardware degradation or system updates that jeopardize the predictive performance of the model. The maintenance part of this phase is concerned with adapting the model once a distribution shift or other form of performance degradation has been identified. The information provided by CRISP-ML(Q) on the monitoring and maintenance phase is still on a rather abstract level as it aims at being as broadly applicable as CRISP-DM. Thus, the guidance regarding the monitoring and maintenance of ML models is limited to a description of the general necessity for such mechanisms and high-level examples for the implementation. In addition to the monitoring and maintenance phase, CRISP-ML(Q) adds Quality Assurance (QA) steps to all lifecycle phases, aiming at catching errors as early as possible in the development process and thus minimizing the costs of the project. Lastly, CRISP-ML(Q) merges business understanding and data understanding into a single phase, arguing that they are strongly intertwined in practice as business objectives can be derived or changed based on the available data.

MLOps Machine Learning Operations (MLOps), is an overarching paradigm that integrates ML with data engineering and software engineering practices to optimize the lifecycle management of ML systems. MLOps is a compound of "*machine learning*" and DevOps from the field of software engineering [John21]. It leverages DevOps principles such as Continuous Integration / Continuous Delivery (CI/CD), and automated pipelines, specifically tailored to address the nuances of ML projects, including data versioning, model training and validation, and deployment strategies. At its core, MLOps focuses on automating the ML model development process, encompassing data preprocessing, model training, validation, and deployment phases, with an emphasis on tracking and versioning not only code but also datasets and models. This ensures reproducibility and traceability throughout the ML lifecycle. Automated testing and validation are critical, involving techniques like A/B testing and canary releases to ensure model reliability and performance before full-scale deployment.

[2] Monitoring in the scope of the CRISP-ML(Q) process model refers to the monitoring of deployed ML models, in the sense of ensuring their continuous reliability, thus introducing some ambiguity with respect to process and condition monitoring as the manufacturing-specific use cases of this dissertation.

CI/CD pipelines are adapted for ML workflows, automating the integration of new code changes and the deployment of models to production environments. This process includes the automation of model training, testing, and deployment tasks, enabling faster iteration and deployment cycles while maintaining high-quality standards. Monitoring and governance in MLOps go beyond traditional software metrics to include model performance monitoring, concept drift detection, and retraining triggers [Syme22; Test22]. Tools and platforms in the MLOps ecosystem provide capabilities for logging, monitoring, and alerting that help maintain the health and accuracy of deployed models over time.

Process models that structure the lifecycle of ML models as well as the MLOps paradigms are of special importance for this thesis as the detection of concept drift and the consequent root-cause analysis or model retraining are integral parts of the ML model lifecycle. Thus, the approaches developed within this thesis can be viewed as concretizations of parts of the process models, specifically adapted to ML applications for monitoring manufacturing processes. This is elaborated in Chapter 4.

2.4. Two-sample hypothesis testing

Two-sample hypothesis testing plays an important role in the detection of concept drift within Chapter 4 of this thesis. Thus, the hypothesis tests that are utilized within the experiments are elaborated in this section.

Generally, two-sample hypothesis tests are performed on data of two random samples which are each independently obtained from different given populations. The test is performed to determine whether there is a statistically significant difference between the two populations. Thus, the null hypothesis H_0 typically assumes that the distributions are equal [Chat18]. The null hypothesis H_0 is rejected if the p-value associated with the test statistic is less than α, suggesting a statistically significant difference in the distributions of the two groups.

In the following, the Kolmogorov-Smirnov (KS) test, the Chi-squared test as well as the Maximum Mean Discrepancy (MMD) test will be introduced. While many more tests have been proposed in literature, these tests have proven their utility in concept drift detection and open-source implementations are available, e.g., [Van 19], allowing their implementation in practical settings.

Kolmogorov-Smirnov test

The KS test is a nonparametric test used to determine if two real-valued, univariate samples, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, originate from the same distribution (H_0) [Lehm86]. To employ the KS test for two-sample hypothesis testing, the Empirical Distribution Functions (EDFs) $F_1(z)$ and $F_2(z)$ of the two samples are computed.
The KS statistic, Z, is the maximum absolute difference between the two EDFs over their full value range z:

$$Z = \max_{z} |F_1(z) - F_2(z)| \,. \tag{2.53}$$

Under H_0, the KS statistic Z follows the Kolmogorov distribution. On this assumption, the p-value is computed, indicating the probability of observing a Z statistic as extreme as, or more extreme than, the one calculated from the samples, assuming that H_0 is true.
The KS test does not rely on assumptions on specific distributions or properties regarding the data samples besides them being Independent and Identically Distributed (i.i.d.). A small p-value leads to the rejection of H_0, suggesting that the two samples are likely drawn from different distributions.

Chi-squared test

While the KS test is a popular two-sample test for continuous variables, it is not suitable for categorical variables [Raba19]. In the case of categorical features such as the tool number, a Computerized Numerical Control (CNC) program status or the position of certain switches, the KS test is thus not applicable. In these cases, the nonparametric Chi-squared test for homogeneity can be employed [Lehm86]. To test a feature for the previously defined null hypothesis using the Chi-squared test, a contingency table has to be constructed that tabulates the observed frequencies of each category of the categorical variable in each group. In the contingency table, the r columns represent the different categorical states of the variable, while the two rows represent the two samples – $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ – respectively. The Chi-squared statistic is calculated as

$$X^2 = \sum_{i=1}^{r} \sum_{j=1}^{2} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \tag{2.54}$$

where o_{ij} is the observed frequency of group i in category j and e_{ij} is the expected frequency of group i in category j. The expected frequency e_{ij} is computed under the hypothesis of homogeneity across the groups, based on the marginal totals of the table.

Maximum Mean Discrepancy test

Both the KS test as well as the Chi-squared test are designed as univariate tests, thus testing one-dimensional samples. In contrast, the MMD test [Gret12] is a popular nonparametric two-sample test for multivariate data samples. The MMD test is grounded in the framework of kernel methods, similar to the SVM, explained in Section 2.2.

The MMD is a statistical distance measure for comparing two distributions, P and Q, by evaluating the difference between their sample means within a high-dimensional feature space. A critical component in computing the MMD is the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$. The RBF kernel, defined as $k(\boldsymbol{x}, \boldsymbol{x'}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x'}\|^2}{2\sigma^2}\right)$, with σ as the bandwidth parameter, is typically utilized due to its effectiveness in capturing similarities between high-dimensional vectors based on their Euclidean distance. The RBF kernel is commonly used in SVMs as well, *cf.* Section 2.2.

The squared MMD statistic is formulated as:

$$\mathrm{MMD}^2(P, Q) = \|\mathbb{E}[\phi(\boldsymbol{x}_1)] - \mathbb{E}[\phi(\boldsymbol{x}_2)]\|^2, \tag{2.55}$$

where ϕ represents the mapping function to the feature space, and $\boldsymbol{x}_1, \boldsymbol{x}_2$ are multivariate samples from distribution P and distribution Q, represented as vectors.[3]. Simplifying this using kernel functions yields:

$$\mathrm{MMD}^2(P, Q) = \mathbb{E}[k(\boldsymbol{x}_1, \boldsymbol{x}_1)] + \mathbb{E}[k(\boldsymbol{x}_2, \boldsymbol{x}_2)] - 2\mathbb{E}[k(\boldsymbol{x}_1, \boldsymbol{x}_2)]. \tag{2.56}$$

To determine the statistical significance of the observed MMD statistic, permutation testing is employed. This involves combining the multivariate samples from both distributions and randomly reallocating them into two new groups of the same sizes as the original groups. The MMD statistic is recalculated for each of n_{perm}

[3] Note that $\boldsymbol{x}$ in the context of the MMD describes a single multivariate sample while in the context of the KS and Chi-squared tests it referred to multiple univariate samples.

permutations, forming a distribution of MMD values under the null hypothesis that assumes no inherent difference between P and Q. The p-value, reflecting the proportion of permuted MMD statistics that are as extreme as or exceed the original value, is calculated by dividing the number of permuted MMD values greater than or equal to the original MMD value by the total number of permutations.

The MMD test is able to capture a wide range of distributional properties, making it particularly powerful for analyzing complex or high-dimensional data.

2.5. Machine learning in manufacturing

AI in general and ML in particular have a multitude of application areas in manufacturing-related tasks [Wues16; Hati19]. An overview is shown in Figure 2.20, highlighting the diverse landscape of possible use cases within the manufacturing domain. The interest in applying ML to manufacturing problems has been rising steadily for the past years [Chui21; Chui22], fueled by the increasing amount of available process data through IoT systems and other technologies in the scope of Industry 4.0 as well as the accessibility of ML technology through open source frameworks such as *scikit-learn* [Pedr11] and *pytorch* [Pasz19][Wues16].

Maintenance	Logistics	Quality management and quality control	Product and process development
Digital assistance systems	Process optimization and control	Resource planning	Automation

Figure 2.20.: Application areas of AI in manufacturing-related tasks. Own illustration adapted from [Hati19].

The investigation in this thesis focuses on the use case of monitoring manufacturing processes and equipment, thereby relating to the areas of "Maintenance", "Process optimization and control" and "Quality management and quality control", highlighted blue within Figure 2.20. The overarching aim of monitoring is to assess the behavior of the object that is monitored [Brec21]. Monitoring within the domain of manufacturing can be broadly partitioned into process monitoring and condition monitoring.

Process monitoring focuses on the machining process, aiming to ensure optimal performance, maintaining product quality and identifying deviations [Brec21]. In the context of ML, data-based process monitoring is sometimes referred to as predictive quality [Krau20; Cass22; Cass23; Fert23]. Condition monitoring on the other hand aims at measuring or estimating the health or wear state of machine components with the ultimate goal of preventing unscheduled breakdowns and quality problems [Brec21]. The terminology regarding monitoring of machine tool wear is subject to varying classifications depending on the source: Some authors classify it as process monitoring when the respective study is focussed on overseeing the operational aspects of the process where damaged machine tools are a possible factor that may negatively impact the process quality, e.g., [Brec21]. Other authors classify it as part of condition monitoring when the emphasis of the respective study shifts to estimating and tracking the wear and health of tools over time, e.g. [Reho05].

In the following parts of this section, the tasks of process monitoring, predictive quality and condition monitoring using ML are explained in detail, accompanied by an overview of recent research studies. This section focuses on recent studies related to the general implementation of the aforementioned use cases in manufacturing while an overview of studies involving concept drift in relation to the use cases is given in Section 3.1.

2.5.1. Condition monitoring

Condition monitoring, also referred to as Condition-based Maintenance (CbM) is a type of maintenance activity. Maintenance is defined by the DIN EN 13306 as "*the combination of all technical, administrative and managerial actions during the lifecycle of an item intended to retain it in, or restore it to, a state in which it can perform its required function*" [DIN EN 13306]. Different types of maintenance activities exist, as visualized in Figure 2.21. Corrective maintenance is carried out after a failure has occurred and is aimed at restoring an item to a state in which it can perform its required function. Corrective maintenance can be unplanned (immediate) or planned (deferred) until a suitable break in operations. Condition monitoring on the other hand belongs to the category of preventive maintenance. This type of maintenance is carried out at predetermined intervals or according to prescribed criteria, aimed at reducing the likelihood of failure or the degradation of the functionality of an item. Condition monitoring is part of the discipline of Prognostics and Health Management (PHM). PHM systems enable maintenance action on machines and machine components based on the actual system condition in contrast to the use of preventative schedules [NISTIR 8012]. Figure 2.22 visualizes a reference architecture for the implementation of PHM systems in industrial applications as defined in the [ISO 13374-2].

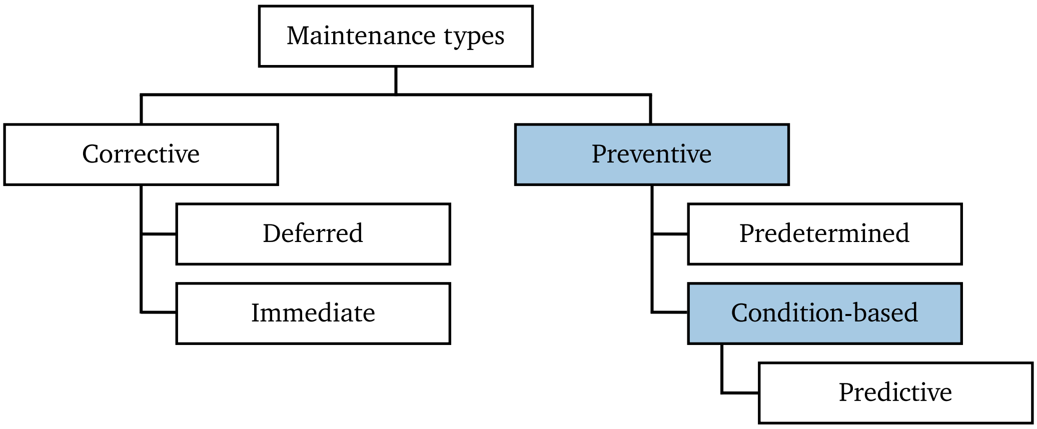


Figure 2.21.: Categorization of maintenance types according to DIN EN 13306. Condition-based maintenance is the focus of the respective case studies within this thesis. Own illustration adapted from [DIN EN 13306].

In the first stage, the data of suitable sensors as well as accompanying metadata is acquired. Subsequently, the data is preprocessed, e.g., by aligning sensor data with timestamps or removing outliers. If the PHM approach involves utilizing ML in subsequent stages, it is essential to extract and select features that effectively describe the machine state. The next five steps represent the development stages of a PHM system. In the state detection stage, univariate features are monitored using simple predetermined thresholds. If a feature exceeds a given limit, an alert or an alarm is raised. Health assessment involves the multivariate analysis of the extracted features to compute a health indicator representing the current condition of the monitored component. Potentially detected problems are diagnosed or classified. In the final two stages, the focus shifts to providing a prognosis of the Remaining Useful Lifetime (RUL) of the equipment, along with subsequent

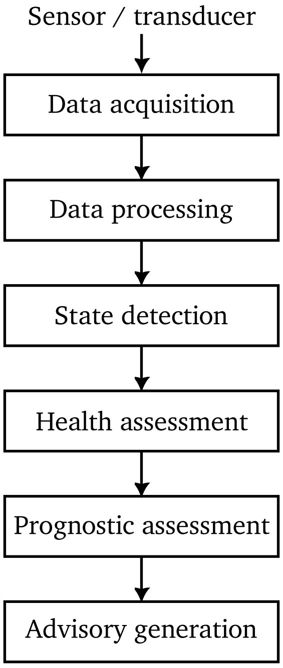


Figure 2.22.: Reference processing architecture *Open System Architecture for Condition-Based Maintenance (OSA-CBM)* of a PHM system according to ISO 13374-2:2007. Own illustration adapted from [ISO 13374-2; NISTIR 8012].

recommendations for maintenance [NISTIR 8012; Preu18]. This prognostic aspect corresponds to predictive maintenance in Figure 2.21.

Data-based PHM In developing algorithms for a PHM system, literature distinguishes between different categories such as model-based, data-based and hybrid approaches. Model-based approaches describe the physical system behavior of a component through a set of mathematical laws which are utilized to model the degradation behavior or detect deviations from the expected operating modes [Preu18; Ting19]. Data-based models rely on historical data and algorithms from the realm of ML to model the dependency between extracted features and the condition of the monitored component [Zhou18]. Hybrid approaches combine model-based and data-based methods.

The case studies within this thesis are concerned with data-based approaches that can be associated with the health assessment stage of OSA-CBM, *cf.* Figure 2.22. Thus, the expected output of the condition monitoring system is the health state of the monitored system. In recent literature, this task is typically posed as either a supervised classification problem or an unsupervised anomaly detection problem that relies on high-frequency data recorded from one or multiple sensors [Bert21]. The classification is done either binary, e.g., with the classes healthy and faulty [Hess19], or in a multiclass way, e.g., with different fault types [Kank11], or severities [Abde18]. In certain cases, a continuous health state is estimated using regression models [Soua14]. Reviews cite NNs, tree-based methods such as RFs as well as SVMs as commonly applied ML techniques in condition monitoring problems in manufacturing applications [Bert21].
Within the last decade, the application of deep learning methods became increasingly relevant for condition monitoring applications [Fink20]. Utilizing sensors and data acquisition electronics capable of high sampling rates, quantities such as acoustic emissions, accelerations or cutting forces are recorded. Consequently, image-like representations of the time series data are generated using time-frequency transformations resulting in

spectrograms or scalograms using Short-Time Fourier Transformation (STFT), Continuous Wavelet Transformation (CWT) or the Hilbert-Huang Transformation (HHT), which is elaborated upon in Section 2.6. These representations can be processed using CNNs, yielding strong performance in empirical evaluations without the need for manual feature engineering. Common architectures include classification CNNs, *cf*. Figure 2.12 as well as CAEs [Vers17; Garc22].

Tool condition monitoring Tool condition monitoring is a subdomain of condition monitoring which comprises the monitoring of machining tools to reduce the number of unplanned downtimes and quality problems due to tool wear or tool damage using hardware for signal acquisition and software for signal analysis and interpretation [Zhan16]. It is estimated that the fraction of unplanned downtime in CNC machining processes that is caused solely by unexpected tool breakage amounts to 7-20% [Dan90]. In addition, untreated excessive tool wear can negatively affect the quality of the workpiece, thereby generating scrap parts [Seri20]. Effective tool condition monitoring thus has great potential in improving manufacturing operations. Methods for tool condition monitoring can be broadly categorized into direct and indirect approaches [Seri20]. Direct methods, e.g., [Jour21a], use, among others, vision sensors, proximity sensors or radioactivity sensors to measure actual changes in the tool geometry that are caused by wear. Direct methods are less susceptible to concept drift as they don't model the dependencies between input features and the wear state but directly measure the quantity of interest. However, they are typically only applicable offline, *i.e.* when the machining process is halted due to the mostly inaccessible cutting area and the continuous contact between tool and workpiece [Ambh15]. Indirect methods on the other hand, e.g. [Ahma20; Moha20], rely on signals that are correlated with tool wear such as spindle current, cutting forces, vibrations or acoustic emissions to infer the tool condition during machining and are thus more common. As their functionality depends on the validity of the learned dependency between signals and the actual tool wear, they are vulnerable to changes in the data distribution [Grim99]. ML-based approaches for indirect tool condition monitoring typically rely on the methodologies described above for data-based PHM systems [Seri20].

2.5.2. Process monitoring and predictive quality

Quality in manufacturing is defined by the DIN EN ISO 9000ff standards as the ability of a product to fulfill its requirements [DIN EN ISO 9000]. Products that do not comply with the requirements or specifications have to be either disposed or reworked, in turn generating costs for the respective company [DIN EN ISO 9001; Krau20]. Manufacturing processes are influenced by random as well as systematic disturbances that can result in fluctuations of the product quality. While systematic disturbances are often addressed using frameworks such as Statistical Process Control (SPC), random disturbances necessitate the monitoring of quality characteristics during production, referred to as Quality Control (QC), to prevent defective products from reaching the customer undetected. Commonly, QC involves the drawing of representative test samples, e.g. from each batch of production, whereas certain scenarios mandate comprehensive 100% inspections to rigorously ascertain the conformity and integrity of the product [Brem15; Fert23].

The monitoring of product quality through explicit measurement of quality characteristics, e.g., in coordinate measurement machines for machining parts, is a costly process that is typically not value-adding [Zieg20]. Data-driven methods that can detect defects early on and provide quality estimates without explicit measurement thus have great potential in saving costs for manufacturing companies. In literature, approaches relying on analytical models, approaches based on experiments or wear trials as well as ML-based approaches can be identified [Bena03; Fert23]. The investigations within this thesis focus on the latter. ML-based quality prediction tasks can be further divided into approaches that aim at optimizing production parameters such as, e.g., machining feed rate, and approaches that estimate the resulting process quality from time series process measurements such as vibrations, forces or internal machine signals [Fert23] which is the focus of the respective case study within this thesis. Similar to the domain of data-based PHM, ML tasks in this realm can be posed as either anomaly detection problems, e.g., [Bieg22], or supervised classification problems, e.g. [Tnan22b; Cass23; Fert23]. In certain cases, continuous quality measurements such as geometrical deviations are estimated with regression methods, e.g., [Neto13].

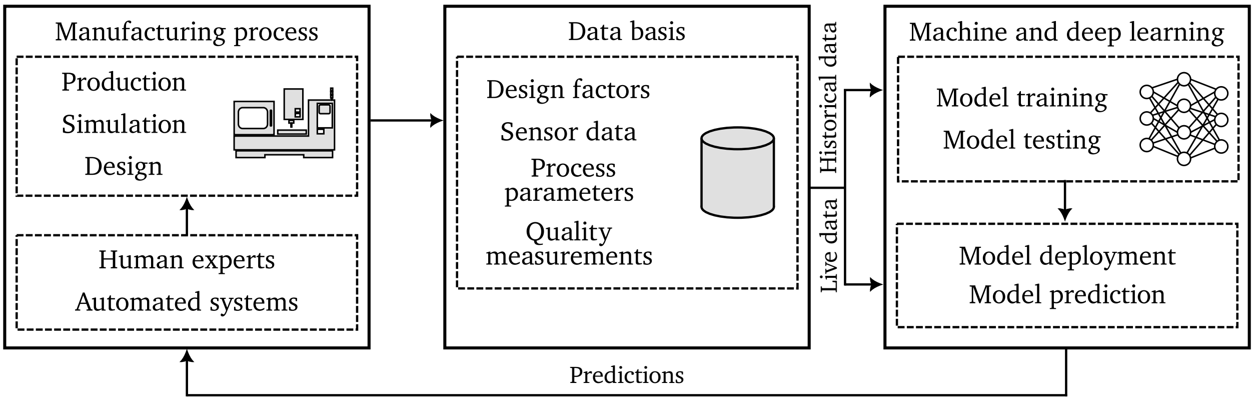


Figure 2.23.: General approach to predictive quality applications as described by [Terc22b]. Own illustration adapted from [Terc22b].

Generally, predictive quality involves three main steps: The collection and aggregation of process and quality data to build a dataset, the training of the ML model, and the deployment and usage of the model for real-time predictions as an addition or replacement for manual QC. Figure 2.23 contains a high level visualization of the general approach to predictive quality.

In literature, there is no clear distinction in terminology between ML-based process monitoring and predictive quality as both are typically concerned with identifying process behavior deviations that lead to quality problems. It is, however, observed that the term predictive quality is more frequently associated with supervised classification tasks, whereas the term process monitoring is often linked to unsupervised anomaly detection. This thesis acknowledges that there are exceptions in both domains, e.g., [Md22], but adheres to this general differentiation for clarity and consistency in discussion. A recent review [Terc22b] into supervised ML-based predictive quality methods shows RF, SVM, kNN as well as CNNs and MLPs as commonly employed algorithms. Common data types used for predictive quality are sensor data, machine parameters as well as (visual) product measurements [Terc22b]. In ML-based process monitoring, both supervised and unsupervised algorithms are used [Amin18]. Prominent methods in the realm of unsupervised process monitoring include variations of AEs, IFs, OC-SVMs as well as CNNs. Datasets for process monitoring and predictive quality are often imbalanced, as the proportion of defective products in a modern manufacturing plant is very low [Krau20; Jour21b; Terc22b; Tnan22a].

2.6. Time series feature extraction

As identified in Section 2.5, a significant share of ML applications in the manufacturing domain as well as the case studies in Chapter 5 rely on time series data from sensors such as accelerometers or internal machine signals. For raw time series data to be processed using the ML algorithms introduced in Section 2.2, feature extraction is performed as a preprocessing step. This involves distilling meaningful patterns, trends and other relevant information from the temporal sequence.

Three main feature families are commonly used for time series signals in industrial applications of ML: **Time domain** features, **frequency domain** features as well as **time-frequency domain** features, *cf.* Figure 2.24 [Delg11; Zhou18]. While the time domain features typically represent simple statistical properties of the time series, such as minimum, maximum, Root Mean Square (RMS) or peak-to-peak, frequency and time-frequency domain features require further preprocessing as explained in the following. The features that are utilized within the case studies in Chapter 5 are summarized in Table B.1 in the Appendix.

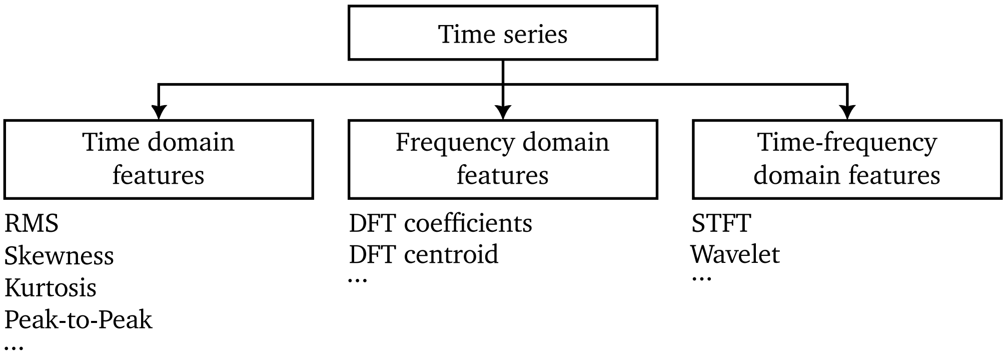


Figure 2.24.: Time series feature types used for ML in the case studies of this thesis. Own illustration adapted from [Preu18; Zhou18]

Frequency and time-frequency features

Three relevant methods for capturing frequency and time-frequency information from time series are utilized in this thesis: Fourier transformation, STFT and CWT. The following explanations draw primarily from the book titled "*Mechanical vibrations and condition monitoring*" by Correa et al. [Corr20].

Fourier transformation The Fourier transformation is a mathematical operation to transform a function of time into a function of frequency. It is a fundamental tool in signal processing and analysis of data originating from sensors such as accelerometers, dynamometers as well as acoustic emission sensors. The transformation enables the identification of characteristic frequencies that indicate normal operation or specific types of faults. In the industrial context, the Discrete Fourier Transformation (DFT) is used for discrete, digital signals that are sampled using sensors, data acquisition devices and Analog-to-Digital Converter (ADC). Mathematically, the DFT for a time series $\{x_i\}_{i=1}^{n}$ is defined as

$$\tilde{x}_k = \sum_{i=1}^{n} x_i \cdot e^{-\frac{2\pi j}{n}(k-1)(i-1)} \quad \text{for} \quad k = 1, 2, \ldots, n, \tag{2.57}$$

where j is the imaginary number, $\tilde{x}_k$ are the Fourier coefficients, representing magnitude $S(f_k) = |\tilde{x}_k|$ and phase at the k-th frequency bin with frequency $f_k = \frac{(k-1)f_s}{n}$ and sampling rate f_s in Hz $(\frac{1}{s})$. By analyzing the values of $\tilde{x}_k$, one can perform spectral analysis of the signal, identifying dominant frequencies, detecting periodicities, and understanding the overall frequency content.

Short-time Fourier transformation While the DFT only offers global information about the time series, the STFT is a powerful tool for analyzing the content of non-stationary signals, where the signal's properties change over time. Given a discrete time series $\{x_i\}_{i=1}^n$ as above, the STFT result $\tilde{X}$ at a discrete time index t and a frequency bin k is calculated as:

$$\tilde{x}_{t,k} = \sum_{i=0}^{l-1} x_{t+i} \cdot w(i) \cdot e^{-\frac{2\pi i j}{l} ki}, \tag{2.58}$$

where l represents the length of the individual windows, t represents the position of the current window in the overall signal and $w(i)$ is the window function value at point i relative to the center of the current window. The window function $w(\cdot)$, which is often a Gaussian or a Hann window, selectively segments the signal into short sections and weighs the signal to attenuate the edges of the section. This windowing is crucial because it minimizes the spectral leakage that would otherwise occur due to segmenting the signal into finite intervals. The STFT is computed by sliding the window function along the time axis, computing the Fourier transformation at each position, which results in a two-dimensional representation of the signal, where one dimension represents time and the other represents frequency. The squared magnitude $|\tilde{X}^2|$ of the STFT can be represented as a **spectrogram**, which is a visual depiction of the signal's frequency spectrum as it varies with time, *cf.* Figure 4.8 for an example. The resolution of the STFT in both time and frequency is determined by the size of the window. A wider window provides better frequency resolution but poorer time resolution, and vice versa, known as the time-frequency resolution trade-off.

Continuous wavelet transformation Unlike the Fourier transformation, which uses sinusoids, and the STFT, which uses a windowed version of the Fourier transformation, the CWT employs wavelets – small waves that grow and decay within a finite period. Mathematically, CWT is defined as the convolution of a signal with a family of wavelets generated from a mother wavelet ψ. These wavelets are scaled and translated versions of the mother wavelet, which is a small wave with limited duration. The CWT of a discrete signal is a computation that produces a two-dimensional representation of the signal, similar to STFT. In contrast to STFT and the resulting spectrograms though, the CWT representation is referred to as **scalogram**, encoding wavelet scale on the ordinate and the position in time on the abscissa, providing insight into the signal's frequency content as it varies over time, *cf.* Figure 4.8. The CWT of a discrete time series $\{x_i\}_{i=1}^n$ is given by:

$$\tilde{x}_{a,b} = \frac{1}{\sqrt{a}} \sum_{i=1}^{n} x_i \cdot \psi^* \left(\frac{i-b}{a} \right), \tag{2.59}$$

also referred to as the wavelet coefficients, where parameter a controls the scale the wavelet is dilated by, parameter b controls its position in time and $\psi^*(\cdot)$ denotes the complex conjugate of the mother wavelet. The scaling factor $\frac{1}{\sqrt{a}}$ is introduced for energy normalization across different scales [Gao11]. The choice of the mother wavelet is an important parameter of the CWT. In the context of vibration analysis for fault identification, process monitoring and condition monitoring, the Morlet wavelet defined as

$$\psi(t) = \frac{1}{\sqrt[4]{\pi}} e^{j\omega_0 t} e^{\frac{-t^2}{2}}, \tag{2.60}$$

has shown high performance in similar studies, *cf.* [Gao11; Bieg23], and is thus adopted for the experiments in this thesis.

2.7. Summary

Chapter 2 introduces the essential theoretical and conceptual foundations for the research in the following chapters of this thesis. Section 2.1 presents the utilized mathematical notation and fundamental paradigms of ML, focussing on supervised and unsupervised learning. Furthermore, metrics for the performance evaluation of classification and anomaly detection models are introduced, including metrics for both discrete classification results as well as continuous score assessment. Section 2.2 gives a concise overview of the theoretical background and properties of the ML algorithms that are utilized. Section 2.3 introduces the concept of MLOps as well as the most common process models used for developing industrial ML applications. Furthermore, the phenomenon of concept drift is introduced. Notably, recent process models such as the CRISP-ML(Q) recognize the issue of concept drift and non-stationarity but do not provide specific advice on how to approach or handle it, which is the fundamental problem that this thesis focuses on. Section 2.4 presents univariate and multivariate two-sample tests which are utilized within the following chapters. Consequently, condition monitoring, process monitoring and predictive quality are introduced in Section 2.5 as the central manufacturing use cases of ML within this thesis. The presented use cases share several similarities regarding general architecture, employed data types as well as ML algorithms. It is shown that ML tasks in this domain often rely on high-frequency sensor data and internal machine signals and pose the problem as either supervised classification – in the use cases condition monitoring and predictive quality – or anomaly detection – in the use case of process monitoring. Commonly used algorithms for supervised classification include RFs, SVCs, kNNs as well as NNs while AEs, IFs, OC-SVMs and AEs are used for anomaly detection problems in this domain. Typically, features from the time domain, frequency domain or time-frequency domain are extracted from the raw time series data to be used as input data for the ML models as presented in Section 2.6. The described types of problems, employed algorithms and data formats are considered throughout the following chapters and case studies of this thesis to increase the practical relevance of the research.

3. Challenges and required action

The preceeding chapter presents a comprehensive overview of relevant ML techniques and their usage in monitoring manufacturing processes as well as the definition of concept drift and related phenomena. Building on these fundamentals, this chapter investigates the general relevance and state-of-the-art approaches for addressing concept drift specifically in the manufacturing context. This is done in two ways: First, a literature review is conducted in Section 3.1 to summarize the outcomes of recent studies on concept drift within ML applications in the manufacturing sector. This review seeks to delineate the primary focuses of these studies and identify potential gaps in the current research landscape. Second, Section 3.2 presents the results of expert interviews conducted with industry practitioners to explore the consequences of concept drift in practice as well as practical needs that exist in the industry. Consequently, the results are aggregated and summarized in Section 3.3.

Overall, this chapter aims at answering the following research question, defined in Chapter 1:

Research Question 1

What impact does concept drift have on ML applications used in process and condition monitoring of manufacturing processes and how is it currently addressed in both research and practical applications?

3.1. Literature on concept drift in machine learning for manufacturing

The literature referenced in the fields of condition monitoring, process monitoring and predictive quality within Sections 2.5.1 and 2.5.2 concentrates on addressing challenges encountered in the manufacturing domain. This includes the identification or classification of special defect types, as well as the development and implementation of novel algorithms uniquely tailored to solve process-specific problems. It is commonly presumed that there is an ample supply of data available for the respective problem, or that a substantial quantity of data can be effectively gathered through experimental means. Less focus is put on the potential practical deployment and usage of the developed methods and models in industrial plants of companies.

In the following part of this section, the relevant literature is presented, beginning with use case-specific publications, followed by use case-independent ones, ordered by year of publication. Literature specifically addressing the detection, management, or adaptation to concept drift in manufacturing applications of ML is limited when compared to the broader ML literature on concept drift and its applications in various domains like recommender systems. One possible reason is that even though ML has a large potential in manufacturing, the number of actually established ML applications running productively in the long term beyond prototype status is still relatively low and the adoption has only picked up in recent years [Mett21].

3.1.1. Condition monitoring

[Zeni19] introduces an approach for concept drift detection through the training of an ML model to predict single features or process measurements based on other features or process measurements that are available to the application. The ML model is thus used as a virtual sensor. Concept drift is assumed to manifest from machine wear or process anomalies and is detected by monitoring the regression error of the aforementioned model. Within the study, this indication is not used to determine whether an ML application requires being updated but rather to indicate necessary maintenance activities of the monitored machinery.

[Lin19] monitor the change rate of the error quantities within the confusion matrix of a classification model used for condition monitoring to detect concept drift. Aging machine components are mentioned as the primary source of concept drift in the described scenario. The employed Linear Four Rates (LFR) method sets thresholds for warning and drift levels based on the calculated rates. If a rate exceeds the warning threshold, it indicates a potential drift, and the system starts preparing for retraining, e.g., by collecting data points. If it exceeds the drift threshold, it confirms a concept drift, prompting the system to adapt by retraining the model with collected data points. The method is evaluated using synthetic datasets.

[Tian19] presents an approach for online anomaly detection under concept drift in structural health monitoring using OC-SVMs. The method distinguishes between concept drift and actual anomalies that should be detected by evaluating the relative relationship between an incoming sample and margin support vectors, error support vectors and reserve vectors of the OC-SVM. The mentioned sources of concept drift include changes in ambient temperature and load factors. In case of detected concept drift, the OC-SVM model is adapted to the changed conditions. Case studies show that the model stays reliable over time.

[Yong20] utilize AEs for concept drift detection in a condition monitoring case study concerning the component health of a hydraulic test bench. The model is trained on a certain subset of process conditions and then subsequently tested on the full set as well as data that contains artificial noise. Concept drift is thus assumed to arise from changing process conditions as well as changing measurement characteristics of the sensors. The authors show that the reconstruction error of the AE increases in case of unseen process conditions as well as when noise is injected to the data, making it a promising basis for detecting concept drift in related applications, although no explicit detection is conducted in the scope of the study.

[Masc20] present a deep learning-based approach for predictive maintenance of turbofan engines, leveraging Elastic Weight Consolidation (EWC) for continual learning. ML models are adapted to different engines. Concept drift is not explicitly detected but assumed to be present when applying the trained model to a different instance of the engine.

[Mole20] presents a study on handling concept drift in predictive modeling for the maintenance of electricity production units in power plants. In the case study, concept drift is caused by varying operating conditions of the plant. Through comparing stationary and adaptive models, the authors demonstrate that adaptive methods significantly improve the accuracy of fault prediction by adjusting the model in response to changes in data properties over time. This research highlights the effectiveness of concept drift detection techniques that rely on detecting changes in algorithm performance metrics. Labels for retraining and drift detection are assumed to be available.

In a similar use case to [Zeni19], [Lour22] use simulation to evaluate a concept drift detector for condition monitoring in the context of identifying clogged or defective screen packs in an extrusion process. The throughput of the extrusion process as well as the screw's rotational speed are monitored for changes to detect concept drift and process anomalies.

[Gori22] use a time series prediction model to identify anomalies of turbo-machinery in a plant. Recurrent NNs are trained to predict the output of a certain sensor based on other sensors on the same machine. A rising error rate indicates faults or anomalies. It is noted that concept drift due to changing environmental conditions and operating modes of the machine is degrading the model performance over time. The ML model

is thus periodically retrained to adapt to changing concepts without explicit detection.
[Pati22] presents a system for condition monitoring of machines by predicting the time when maintenance needs arise. Concept drift is detected by monitoring the model performance for significant changes. Possible sources for concept drift are not explicitly mentioned. Ensemble models are adapted to new data distributions in case of detected drifts. Labels for retraining and drift detection are assumed to be available.
[Jiao23] proposes an approach called drift-aware weight consolidation for adaptive fault diagnosis in IoT applications. The proposed method uses classifier confidences modeled with beta distribution for drift detection and the triggering of retraining. The proposed method is evaluated on a case study focussing on bearing fault detection where concept drift is assumed to arise mainly due to changing operating parameters and load factors.
[Kerm23] addresses the challenge of anomaly detection in industrial collaborative robots (cobots), which are versatile and operate in dynamic environments where concept drift is caused by changing working conditions and robot tasks. The authors propose an unsupervised anomaly detection method that effectively handles concept drift by distinguishing changes due to different working conditions from actual system degradation. Concept drift is detected by monitoring the latent space of a variational AE and comparing it with a reference distribution using the Mahalanobis distance.

3.1.2. Predictive quality and process monitoring

[Mera19] presents Learn++.MIL, an incremental Multiple Instance Learning (MIL) algorithm designed for non-stationary and recurrent target concepts in industrial visual inspection. It effectively handles concept drift by dynamically selecting and weighing ensemble members for the classification of product quality. Changes in the inspection environment and the raw material induce concept drift. Drift adaptation is triggered passively for every incoming batch of data, without explicit detection.
[Soll20] developed a dynamic error prediction system for industrial machines. If the cumulative drift across multiple features exceeds a certain threshold, it is flagged as a potential error, indicating that the machine's behavior is deviating significantly from its expected operational parameters. In a comparable setting to [Zeni19] this information is not used to update another ML model, but to detect process failure conditions.
[Yang21] propose an adaptive learning method for general anomaly detection use cases in IoT data streams. Novel defect types as well as aging and replaced equipment are mentioned as relevant sources of concept drift in this scenario. Retraining of the model is triggered when accuracy drops are observed using a sliding window approach. Labels are assumed to be available for both retraining and drift detection.
In [Seif21], the authors present a method for detecting concept drift in ML models used for manufacturing process monitoring that leverages Shapley Additive Explanations (SHAP) values to cluster data points into groups with similar properties. Concept drift is then detected by tracking the precision of data points per cluster. If the precision of a given cluster drops below a certain threshold, the corresponding predictions are ignored. In addition to the threshold-based examination, the Page-Hinkley test is applied to the precision over time to detect concept drift. Mechanical properties and wear of tools are mentioned as relevant drift sources. Labels are assumed to be available for drift detection.
[Banf22] propose a novel approach for optical defect detection, applied to steel production use cases. The optical defect detection relies on a CNN classification model for images. The feature embeddings from the trained CNN are subsequently used to build a separate outlier detection model. The outlier detection model uses an IF and a threshold on the outlier score to identify samples that show distribution shift, which could lead to unreliable predictions from the defect detector. New defect types, not contained in the training data, are mentioned as a source of concept drift. Experiments show that the concept drift detection method is able to reliably flag classifier performance degradation.

[Terc22a] investigate the use of memory-aware synapse models for continual learning in quality prediction of injection molding processes. The model is adapted to new product variants, without explicitly detecting concept drift with a similar implementation in [Terc19]. Labels are assumed to be available for retraining.
[Sen23] investigate the application of replay-driven continual learning to process monitoring applications in manufacturing. Disturbances such as temperature variations at the tooltip, vibrations depending on the shopfloor layout, and electromagnetic interference are indicated as relevant sources of concept drift. While the exact method of drift detection is not specified, the authors indicate that adaptation of the ML models is performed when low performance levels on the task are observed. Labels are assumed to be available for both retraining and drift detection.
In [Kvak23], the authors use ML models to predict the quality of injection molded parts, indicated as the resulting part weight after the process. The process is influenced by internal and external disturbances such as component wear and material batch fluctuations, inducing concept drift. Multiple existing methods for detecting concept drift are evaluated in experiments that include artificial disturbances. Methods based on monitoring the error rate of the weight prediction model showed the highest performance, requiring the true part weight or the label for the incoming data samples.
In a similar context, [Pasq23] present DARWIN, an adaptive business process monitoring tool that tracks the error rate of the ML model to detect concept drift over time. The ML model is fine-tuned on new data once a drift is detected. Labels are assumed to be available for both the fine-tuning as well as the concept drift detection.

3.1.3. Other industrial use cases

While the previous subsections contain studies that can be categorized in the use case categories introduced in Section 2.5, the following studies are related to general industrial use cases of ML.

[Wang20] propose the monitoring of the FPR of ML models to detect concept drift in general IoT data streams. Labels are assumed to be available for the drift detection. The paper does not explicitly detail a mechanism for adapting the model in response to detected concept drift.
[Tian21] use a special architecture of recurrent NNs called Error-Long Short-Term Memory (LSTM) for online prediction of industrial compressor vibration signals. Concept drift is present due to machine wear and changes on the shopfloor environment. The proposed method enhances accuracy and efficiency by updating the model based on the test error.
[Bach21] discusses the challenges and recommendations for implementing continuous model improvement in smart manufacturing environments. It emphasizes the importance of managing the lifecycle of ML models after deployment. The authors describe different possibilities for detecting concept drift in industrial ML applications, including the monitoring of the model error rate or observable variables and process measurements. It is noted that industrial settings often do not allow monitoring of the error rate as true labels are typically not available in time.
Similar to [Bach21], [Eck22] describes a monitoring framework for deployed ML models in manufacturing and supply chain forecasting applications. Relevant sources of drift include changing consumer habits in supply chain use cases as well as changing sensor operating conditions and units in manufacturing use cases. The suggested concept drift detection method includes monitoring both, univariate features as well as the model error rate, which can be computed in forecasting settings. The emphasis of the study is put on the IT architecture and integration into existing software stacks.
[Kahr22] investigate concept drift aware ML models for predicting energy consumption of industrial machines. Potential concept drifts are identified by the prolonged inactivity of the respective machines. Consequently, the model's prediction error after a period of inactivity is observed to decide whether a given period will be

considered as drifted and if the model shall be retrained.
[Sari23] use continual learning methods, more specifically memory-aware synapses, for the prediction of Numerical Control (NC) signals, such as current and control deviations, in machining processes. Concept drift arises through new process conditions or programs. The exact mechanism of detecting unknown conditions the model shall adapt to is not explained.

3.1.4. Summary

Analyzing the findings of the preceding state-of-the-art literature review, several observations can be made that are summarized in five key statements.

- The body of literature is sparse when compared to the number of studies that target general ML applications in the respective use cases, as can be compared in recent systematic review studies, *cf.* [Seri20; Terc22b; Bret23].

- There is a large variety in the methods for detecting and handling concept drift found in the studies. Two general categories of approaches can be identified. Approaches belonging to the first category actively identify concept drift by testing for significant changes in quantities of interest. Employed methods in this category rely, among others, on monitoring of error rates, e.g., [Lin19; Wang20; Tian21; Pati22], model feature embeddings, e.g., [Yong20; Banf22] and model confidences, e.g., [Jiao23]. The second category does not quantitatively identify concept drift through correlated quantities but rather updates the ML model proactively in equidistant time intervals or through external triggers such as machine state changes, e.g., [Gori22; Kahr22].

- In the context of anomaly detection tasks in condition monitoring and process monitoring, concept drift detection is sometimes used as a proxy for the actual monitoring task, *cf.* [Zeni19; Soll20; Lour22], creating some ambiguity in the terminology. In this sense, detections of concept drift are seen as a deviation from normal operating conditions. Thus, the concept drift detector is not used as a reliability indicator of another ML model but as the primary model itself.

- The majority of reviewed studies target very specific use cases or application scenarios within manufacturing with few exceptions, e.g., [Bach21]. In addition to the specificity of the application scenarios, the applied ML methods for detecting concept drift are commonly very specific too. The majority of studies either propose a new method or investigate the utility of a specific method from ML research for their use case. Thus, it is not straightforward to draw conclusions or guidelines that extend to other, similar use cases in the scope of monitoring manufacturing processes.

- Most studies are conducted using static datasets and/or laboratory setups while only very few studies actually deploy systems into production. Consequently, many studies assume that labeled data for retraining as well as drift detection is available which is oftentimes unrealistic in production [Huye22]. The analyzed studies often do not only consider drift detection but also the consequent model adaptation through techniques such as continual learning or transfer learning, *cf.* Section 2.3.1. Thus, it is assumed that the distribution shifts are caused by intangible changes in the manufacturing environment, rather than analyzing potential causes.

3.2. Practitioner views on concept drift in machine learning for manufacturing

In addition to the literature review, which elucidated the state-of-the-art in terms of concept drift detection in academia, this section presents the results of an expert interview study with an emphasis on monitoring and maintenance of deployed ML applications in practice. This extension is necessitated by findings from the literature review, which indicate that most studies on concept drift in manufacturing employ simulated conditions and static datasets. Consequently, the inclusion of expert interviews with data science practitioners is imperative to analyze the practical implications of deploying ML applications in the manufacturing domain. The methodology and key implications of the interview study are summarized in the following.

3.2.1. Interview methodology and participants

Parts of this section have been previously published as a conference paper titled "Toward the Sustainable Development of Machine Learning Applications in Industry 4.0" which appeared in the proceedings of the European Conference on Information Systems (ECIS) in 2023 [Elle23].

The interviews follow a semi-structured interview guideline in accordance to [Sark13], comprising relevant questions to identify challenges along the phases of the CRISP-ML(Q) process model while allowing interviewees to freely share further insights and experiences. Overall, 15 data science and ML experts of Industry 4.0 start-ups, Small and Medium-sized Enterprises (SMEs) and large companies are interviewed as Interview Partners (IPs), as shown in Table 3.1.

Table 3.1.: Overview of IP symbols and corresponding industry roles. Note that interviewees may be part of multiple categories.

Manufacturing company	**External consultancy / software provider**	
Data scientist: ML service provider	Consultant: Operations and industrial ML	Manager: Industrial ML software provider
IP1, IP2	IP7, IP8, IP9, IP10, IP11, IP12, IP13, IP14, IP15	IP3, IP4, IP5, IP6, IP7, IP8, IP9, IP10

Experts fill both internal corporate roles as managers and data scientists as well as external consulting roles for ML in Industry 4.0, or work as managers of software providers that offer software for ML applications in Industry 4.0. The involvement of interviewees in both the development and deployment phases of ML projects enables a holistic approach to be adopted, analyzing the interdependencies between ML system design and its long-term deployment in the industry. Even though most ML applications in this sector have only emerged in the past decade, seven ML experts with more than 15 years of experience are interviewed. Moreover, 13 interviewees have at least five years of experience in ML development for Industry 4.0. All of the participants have already provided expert knowledge to various ML development teams prior to this study. In addition to general experience in ML development in industry 4.0, the majority of the participants (IP1,2,4,5,10,11,13,14) possess expertise in manufacturing quality control in the automotive and aerospace sectors. IP8 and IP9 also bring extensive experience in chemical product manufacturing to the study. IP3 and IP12 primarily gained experience in condition monitoring of machinery plants.

After addressing interviewee-related questions to gain insights into their prior experience, current position, and the industries they have been working in, the participants were provided with an overview of the CRISP-DM and CRISP-ML(Q) process models. It was consequently inquired whether the presented process models fit the interviewee's current approach of structuring ML projects within their respective companies. Subsequently, in the main part of the expert interviews, the interviewees were walked through the CRISP-ML(Q) phases and specifically asked how these phases are usually implemented in their projects. In each phase, it was investigated whether there is already consideration regarding reliability and long-term deployment and which specific challenges arose in past projects in this regard. Interviews were conducted from February to April 2022 via video call and lasted 52 minutes on average. After mutual agreement, all interviews were recorded and transcribed using the software Amberscript [Ambe23]. During the analysis of the last three interviews, theoretical saturation was reached, meaning that no further challenge was mentioned by the remaining interviewees [Flic04].

3.2.2. Interview results

In this section, the interview results are aggregated and presented. The section focuses on results that are directly related to the topic of reliability or that were mentioned in the context of the monitoring and maintenance phase of the CRISP-ML(Q) process model. For the full interview results the interested reader is referred to the corresponding publication [Elle23]. Four major themes emerged from the interviews: *active industry recognition*, *lack of standards and best-practices*, *handling unlabeled data* and *consideration along the lifecycle* which are used to structure the interview results in the following.

Active industry recognition A large majority of interviewees recognized the issue of changing environmental conditions or machine states and their influence on the post-deployment model performance as significant and recognized in their application domains although some were not familiar with the terminology concept drift (IP1,2,3,4,5,6,9,10,11,12,13,14,15). Most interviewees agreed in this context that monitoring and maintenance are paramount for long-term deployment. One interviewee described: "*ML models only provide value in situations that are shown in the dataset, [. . .] they are only ever meta-stable*" (IP12). Five practically experienced sources of change were repeatedly mentioned in the interviews: Sensor drift or general sensor and data quality problems (IP1,4,5,6), seasonalities or time dependencies not identified during development (IP5,11,15), changes in the machine or product configuration (IP1,3,12,13), network or hardware problems that render their respective data sources invalid or unavailable (IP6,12), and changes in material properties (IP4,5,6,15). "*Data drift is a significant issue, especially for long-running and old machinery with retrofits*" (IP14). In connection to concept drift, the importance of testing ML systems even shortly after deployment to surface problems that cannot be revealed using a static test set was highlighted by (IP4,5,6,11). One of the interviewees stated: "*For us, it is crucial to evaluate the model before and after deployment [. . .] we were surprised how many problems can only be revealed after deployment*" (IP6). Multiple interviewees (IP2,6,11) mentioned that it is critical to identify a set of environmental conditions as well as operating modes of the machine or production line which the ML application needs to handle, already in the early stages of development as this choice strongly influences the requirements on the training and testing data. Exiting this defined set after deployment necessitates model adaptation with current data. In this context, the integration of domain knowledge is seen as crucial. Workshops with the stakeholders and domain experts should be conducted for clarification, which is viewed as a possible approach to get an initial set of conditions and modes (IP2,8,10,15). Nevertheless, it was mentioned in the same context that this is "*very hard*" (IP2) and unlikely to result in a complete set, making continuous model monitoring and checks mandatory (IP1,6,9,10,11,12,13,14).

One interviewee explained, *“Whatever challenge we face, in the end, it’s always about data. Before development, we often don’t know what we need or we do not have the data needed and in the end, data drifts and other changes kind of force us to reconsider if our trained models are still useful”* (IP5). Multiple interviewees stated that the robustness and sustainable long-term use of ML applications have only recently emerged as a major focus of their work, as the productive usage of ML applications is only slowly becoming a reality in the last years and the number of deployed models is still relatively small (IP1,7,12,14,15).

Lack of standards and best practices A number of interviewees mentioned a lack of standards / best-practice solutions for monitoring and maintenance of ML applications in industrial environments (IP1,2,6,12,14,15) and emphasized the difficulties faced due to this void. The majority of interviewees saw the CRISP-ML(Q) process model as relatively close to the way they internally structure ML projects, even though, at the same time, most of the interviewees have only heard about the predecessor CRISP-DM prior to the interview. In addition, a number of interviewees (IP1,4,8,12,13,14,15) mentioned that CRISP-ML(Q), compared to CRISP-DM, fits real projects better, as the additional monitoring and maintenance phase is crucial for practical deployment. Interviewees described both process models as abstract and high level though, providing only structural guidance rather than concrete implementation advice. The majority of interviewees that had experienced the issue of post-deployment performance degradation firsthand, described that the accompanying issues were commonly only fixed after they already impacted the respective production process and thus retrospectively rather than proactively through concept drift detection. "*We usually are in a ’wait and see’ mode for obvious signs of problems that we would then try to diagnose and fix*" (IP14). Corresponding root causes did not always prove to be related to general drift or seasonalities in the data but also to faulty equipment or network problems (IP4,6,14). *“Our ML development pretty much follows the CRISP-DM process model. But if I am honest, we are primarily concerned with delivering models, and processes become – let’s say - less standardized after deployment. So we really need something like a monitoring and maintenance phase, but right now I would say we rather intervene in emergencies only*" (IP9). While some interviewees have successfully implemented concept drift detection systems, their approaches underline the absence of a unified methodology in addressing concept drift within manufacturing as they are catered to individual use case configurations. Interviewees who did implement monitoring systems described that the utilized approaches often involve monitoring the properties of the input data (IP2,6,10) or a set of conditions for the usage of the model (IP6,14). Statistical measures can be used to capture properties of the training data that are then compared to the live data during operation. Suitable measures include distribution distance metrics or simple thresholds like SPC. It was mentioned that the complexity of the monitoring task rises with the number of data sources (IP2,6). In addition to the input data, it was described that the model errors can be monitored (IP1,2,11,14) although this approach is seen as difficult to execute systematically as labels are required for calculating the model error quantitatively. Oftentimes only obvious prediction errors catch the attention of employees (IP1,14). In this context, it was mentioned that there is uncertainty around the correct approach to monitoring (IP1,2,6,12,14,15).

Handling unlabeled data A fundamental and often encountered problem during both development and post-deployment monitoring is missing meta-data in the form of data annotations and labels related to the target variable, e.g., maintenance activities, faulty products or machine breakdowns (IP1,3,4,5,6,8,12,13). A possible remedy is the usage of semi- or unsupervised ML algorithms, which do not require explicit labels but are not suitable in all use case scenarios. One interviewee mentioned, *“We almost exclusively use unsupervised approaches in manufacturing projects, because labeled data is rarely available in sufficient quality and quantity, and target variables such as fault types are expensive to obtain and may change over time”* (IP4). An interviewee (IP8) noted further, "*It is not just about the data or the algorithms; it is also about how well our teams understand the manufacturing processes and can infer useful labels from their knowledge*".

With respect to post-deployment concept drift detection, one interviewee stated that "*Differentiating between real model problems and normal process fluctuations without labels is challenging.*" (IP3). Without access to ground truth labels during operation, data scientists cannot monitor the actual performance of the model in terms of ML Key Performance Indicators (KPIs). In other application domains, such as recommender systems or forecasting models, this is different. Extensive logging during operation is thus important to quickly analyze model errors and find their root causes (IP1,11,14). As articulated by (IP14), "*Systematically maintaining the accuracy and relevance of ML models over time is difficult without consistent, labeled data [...] we are often reliant on indirect measures to assess system health*".

Consideration along the lifecycle A number of interviewees stated that drift-related concerns such as post-deployment monitoring and maintenance activities should be considered along the whole model lifecycle, including the initial planning and project setup. One interviewee (IP13) stated *"Never change a running system – that is something I hear quite often from our customers in this context and it is really slowing us down. I am convinced that we will provide services to maintain ML systems in the future, but to do that we need to reduce the complexity of these systems. [. . .] So I advocate for future projects to keep ML system maintenance in mind from the beginning and consider that in, yeah, pretty much every future system design"*. Post-deployment considerations significantly influence the planning and development stages of ML models. These considerations include a high degree of automation through DevOps and MLOps pipelines, as referenced in Section 2.3. This automation supports data access, automated monitoring, and notifications. Additionally, where possible, it allows for automated retraining and model deployment when deviations in data distributions are detected (IP2,8,15). Moreover, thinking about the model's lifecycle affects early development decisions, like selecting reliable operating conditions and choosing the right algorithms. This forward-thinking approach is crucial for reducing future maintenance burdens and ensuring the sustainability of ML systems (IP2,8,10,15). Moreover, the integration of lifecycle consideration into ML projects has implications for business models and contractual frameworks within the industry. Although challenging, monitoring and maintenance of ML models is seen as a viable addition to the business model of ML solution providers as mentioned both by manufacturing company internal and external interviewees (IP1,6,8,10,11,12,13,14,15) as it *"provides a constant revenue stream whereas, you know, for this prior development process, we usually agree on project-based fixed-term work and payment"* (IP15). At the same time, this may provide another challenge, as monitoring and maintenance activities are often not covered by the initial development contracts (IP15).

3.3. Summary

In this section, the agreements as well as the identified differences between the literature review in Section 3.1 and the expert interview study in Section 3.2 are summarized, answering the first research question:

Research Question 1

What impact does concept drift have on ML applications used in process and condition monitoring of manufacturing processes and how is it currently addressed in both research and practical applications?

Common findings

- **Recognition of concept drift as a significant challenge:** Both, the expert interview as well as the literature review have established that a central challenge for the practical implementation of ML models for condition monitoring, process monitoring and predictive quality in real-world manufacturing environments is the fact that production processes are subject to continuous change and variation. In both sections, similar sources of concept drift were mentioned, including changing operating parameters, environment conditions, sensor issues and general data quality problems. This often leads to performance degradation due to concept drift which must be properly addressed.
- **Variety in detection methods:** The literature review and expert insights both highlight the diversity of methods for detecting concept drift. The two overarching categories identified in the literature review – active detection vs periodically or externally triggered model updates – are also noted in the discussion within the expert interviews.
- **Gap in standardized frameworks for detection:** The literature review and expert feedback collectively highlight the absence of standardized frameworks and best practices for detecting concept drift in manufacturing ML systems. While the literature indicates a diverse array of specific strategies for concept drift detection without a common framework, expert insights stress the essential need for uniform guidelines to direct the lifecycle management of ML models, including development, monitoring, and updating processes. This consensus points to the critical demand for the creation and implementation of standardized protocols to enhance the robustness and longevity of ML applications against concept drift, starting with effective monitoring of deployed ML models.

Diverging findings

- **Specificity vs general applicability:** The literature review points out that most studies focus on highly specific use cases and application scenarios, often proposing novel or tailored methods for those scenarios. In contrast, expert interviews express a desire for more generalized solutions and standards that can be applied across various manufacturing contexts, indicating a gap between academic research's specificity and the industry's need for broader applicability.
- **Assumption of labeled data:** A significant portion of the literature assumes the availability of labeled data for retraining and drift detection. However, expert interviews reveal that in real-world manufacturing settings, labeled data is often scarce or unavailable, highlighting a disconnect between academic assumptions and industrial reality.
- **Real-world validation:** Experts emphasize the importance of testing and validating ML models in actual production environments, pointing out that many problems only become apparent post-deployment. The literature, however, tends to focus on theoretical models and controlled experiments, with less emphasis on real-world deployment and the practical challenges encountered therein.
- **Differences in addressing root causes:** Literature often presumes that responding to concept drift necessitates updating or adapting the ML model. However, expert interviews suggest a broader view, indicating that drifts may also stem from external factors such as connectivity or sensor issues, not just intangible changes in data distribution. This divergence highlights the literature's focus on algorithmic solutions, whereas practical insights reveal the importance of diagnosing and rectifying hardware or environmental issues as part of drift management.

In both the literature review as well the expert interviews, it has been shown that a multitude of heterogeneous possibilities for detecting and adapting to concept drift exists. This finding is supported by the preceding analysis of process models and paradigms for structuring the lifecycle of ML models, *cf.* Section 2.3. It is shown that recent process models such as CRISP-ML(Q) acknowledge the need to continuously check for concept drift and related performance problems after model deployment but do not provide guidance on how this should be implemented for a given use case.

4. Framework development

This chapter encompasses the development of a framework that provides decision support and guidance for the implementation of appropriate methods for detecting concept drift in ML applications for the monitoring of production processes.
The chapter is divided into six sections. First, Section 4.1 presents the Design Science Research (DSR) approach as the research methodology used in this thesis, along with the framwork's scope and objectives. Consequently, Section 4.2 introduces the framework on a high level. Section 4.3 and Section 4.4, respectively, present the derived structure of active and passive approaches to concept drift detection in detail. Importantly, Section 4.3 furthermore introduces Localized Reference Drift Detection (LRDD) as a novel method for active drift detection that is specifically tailored to the manufacturing applications targeted by this thesis.
Following the introduction of active and passive drift detection as the main components of the framework, Section 4.5 discusses decision factors, requirements and practical considerations regarding the implementation of the introduced methods. The main findings are summarized in Section 4.6.

4.1. Research design

The research process in this thesis follows the DSR approach as proposed in [Hevn04; Peff07; Vais07]. DSR sees widespread use in the development and validation of artifacts such as algorithms, human-computer interfaces and process models. In contrast to explanatory science, DSR aims at academic research objectives that target practical engineering problems in the real world through the development of novel methods and frameworks that enable professionals of the discipline to design solutions for their field problems [Hevn04; Peff07].

While multiple instantiations or models for the DSR approach exist, this thesis utilizes the methodology proposed by [Peff07], which is visualized in Figure 4.1 and divides the research process into six phases, also referred to as activities. Following [Peff07], the six activities are defined: In the first activity, *problem identification & motivation*, the specific research problem is identified and the value of a solution justified. Within this thesis, this is done through the expert interviews and literature review conducted in Chapter 3.
Consequently, objectives or requirements for the solution that is to be designed are defined in the second activity, *objectives of solution*, along with the scope that the solution shall be applicable to. The main RO for this thesis is consequently defined and introduced in the following, along with four derived sub-objectives (RO.a – RO.d):

Research Objective (RO)

Development of a framework for detecting concept drift in ML applications used in process and condition monitoring of manufacturing processes.

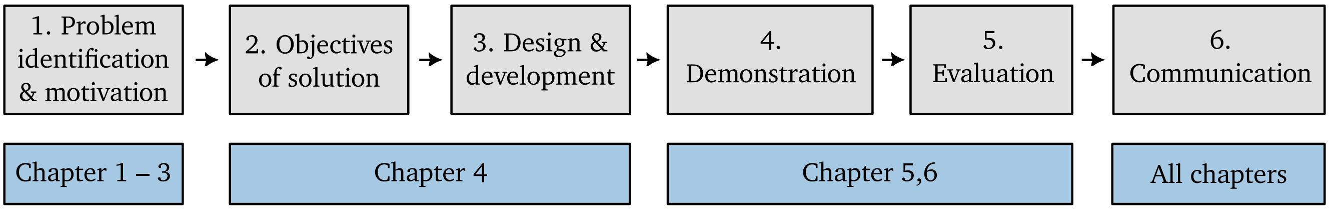


Figure 4.1.: DSR approach used in this thesis along with the relevant chapters of the document. Own illustration following [Peff07].

The main requirement of this framework is to provide methods that are able to detect any concept drifts that would jeopardize the reliability of the respective ML applications, resulting in the objective of a **high recall of the drift detection (RO.a)**. Simultaneously, normal operating conditions should not be erroneously flagged as drifted, thus requiring a **high precision of the drift detection (RO.b)**. The target scope of the solution follows the conclusions of Chapter 2, particularly Section 2.5 along with the findings of Chapter 3. The framework should therefore **be applicable to ML-based condition monitoring, process monitoring and predictive quality scenarios in the manufacturing domain (RO.c)**. It should cover use cases that rely on high-frequency, multivariate time series data that is used in the supervised and unsupervised ML models for classification and anomaly detection identified in Section 2.5.The use cases are chosen as they are seeing strong adoption in companies [Chui21; Mett21] and rely on similar ML architectures as elaborated in Section 2.5.

In summary, the framework serves as a concretization of the corresponding *monitoring & maintenance* phase of the CRISP-ML(Q) model specific to the aforementioned use cases. Importantly, the solution scope is limited to the detection of concept drift, excluding follow-up activities such as root cause analysis or ML model retraining. Findings from the expert interviews detailed in Section 3.2 highlight the diversity in potential sources of concept drift and suitable follow-up activities as it is often unconscious errors, physical defects or network issues that lead to model performance deterioration, necessitating targeted interventions at the root cause level rather than indiscriminate model updates. Other sources of concept drift may be transient, such as network problems or ongoing maintenance work, and thus do not substantiate the training of new models. Automatic model updates are typically unsuitable in commercial manufacturing settings due to the significant costs associated with the frequent acquisition of new labels [Cobb23]. For the same reason, drift detection methods that are relying on ML performance metrics and, in turn, true labels are unsuitable as well. Therefore, the final sub-objective encompasses that **no true labels are required for drift detection (RO.d)**.

The third activity, *design & development*, encompasses the actual creation of the solution approach. In the context of this thesis, a framework for concept drift detection in ML applications used to monitor manufacturing processes is developed within the following sections of this chapter. To achieve this, existing methodologies for addressing concept drift in general ML literature are reviewed to identify which methods are suitable for the target use cases, which requirements and decision criteria exist, and if use case-specific extensions are required.

The developed framework defines guidelines for the detection of concept drift in the targeted ML applications in manufacturing and thus provides a Level 2 contribution in the classification of DSR contribution types, referring to, e.g., constructs, methods, models and design principles and technological rules [Greg13].

The fourth and fifth activities, *demonstration* and *evaluation*, are concerned with the exemplary usage of the designed solution approach in experimentation, simulation or other appropriate activities and the consequent evaluation of whether the intended purpose is fulfilled and the objectives, defined in the second phase, are reached. Within this thesis, the demonstration and evaluation are performed through three case studies in Chapter 5 that show diverse instantiations of condition monitoring, process monitoring as well as predictive quality applications in the manufacturing domain:

- Case study 1 investigates concept drift detection in an accelerometer-based tool condition monitoring application in milling using an experimental setup and dataset that are specifically created for this thesis.
- Case study 2 investigates concept drift detection in a predictive quality and process monitoring application in milling that relies on internal NC signals of the milling machine.
- Case study 3 investigates concept drift detection in an accelerometer-based condition monitoring application for pigment sieving developed together with an industry partner.

Lastly, the details of the developed solution as well as the results of the demonstration and evaluation are disseminated in the final activity, *communication*. In the scope of this thesis, this is done through the thesis itself as well as through the published articles that relate to the topic which are listed in the first section of the Bibliography.

4.2. Framework overview

The proposed framework is visualized on a high level in Figure 4.2 and encompasses the monitoring part of the monitoring & maintenance phase of the CRISP-ML(Q) process model.

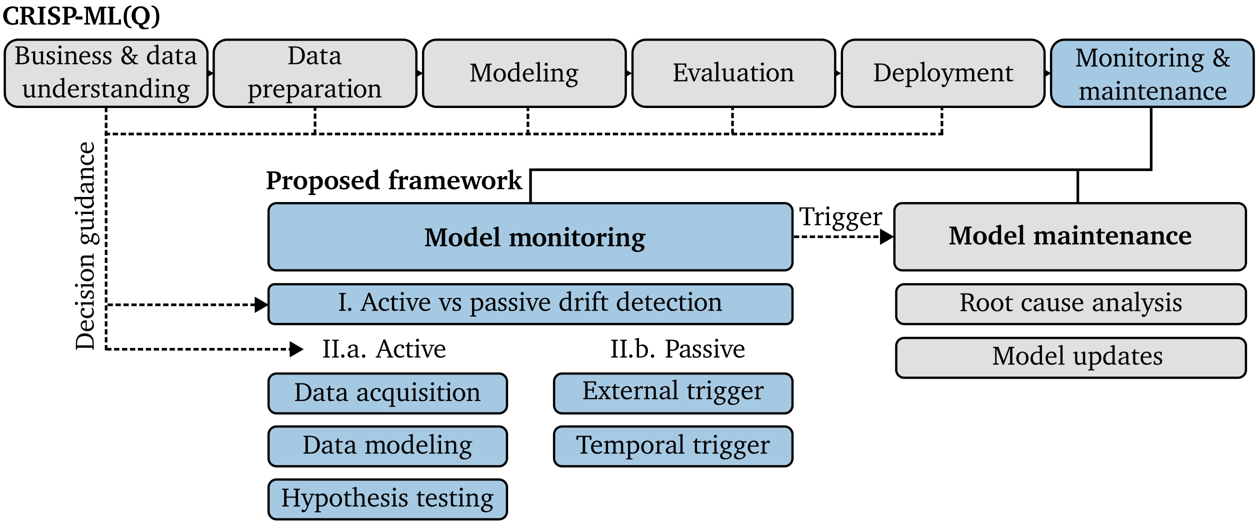


Figure 4.2.: Proposed framework for monitoring of deployed ML models in the specific application scope of this thesis as an implementation of the corresponding CRISP-ML(Q) phase. Own illustration.

Model monitoring aims at identifying situations that may cause performance degradation of a deployed ML model in the manufacturing use cases described in Section 2.5. The output of the model monitoring component is a trigger for the **model maintenance** component.

As identified in the preceding chapter through the literature review and expert interviews, practical drift detection methods for the target use cases in manufacturing can be broadly categorized into two families, **active drift detection** and **passive drift detection** which are defined for the scope of the proposed framework in Definitions 3 and 4.

Definition 3 (Active drift detection) *Active drift detection constitutes a proactive and continuous strategy, systematically assessing specific quantities of interest that are related to the ML model's performance, such as error rates, prediction confidences and input data for significant changes. This approach employs real-time analysis and distributional comparisons to promptly detect deviations caused by concept drift and provide notifications to operators accordingly.*

Definition 4 (Passive drift detection) *Passive drift detection adopts a reactive strategy, characterized by waiting for predefined intervals (temporal triggers) or external events to trigger model maintenance actions. Thus, this approach does not continuously scan for changes in data streams related to the ML model's performance but rather responds to predetermined stimuli. In turn, passive drift detection triggers model maintenance in case of the specified trigger conditions, regardless of whether actual concept drift is present.*

While the distinction in Definitions 3 and 4 is based on the practical considerations in Chapter 3, other definitions involving the wording *active* and *passive* have been proposed in general ML literature. Ditzler et al. (2015) define active approaches towards concept drift in a way that closely resembles the previously mentioned approach, but conceptualize passive monitoring as an online learning scenario, *cf.* Section 2.1.1, where the model is continuously adjusted to accommodate every new data point [Ditz15]. A similar classification has been proposed in [Han22].

In fact, a large portion of the literature concerning concept drift detection and adaptation assumes online learning or sequential learning scenarios, where data points and corresponding labels arrive continuously and new models can be trained or adapted quickly, e.g., [Gama04; Baen06; Bife07; Gama14]. In these studies, the detection of concept drift is mainly utilized to control which data is used for the sequential or incremental training of the model. If a drift is detected, only the data points arriving after the detected change point are used to train a new model [Gama04].

In contrast, Definition 4 does not assume updating the model with each new data point as this is not practical in the addressed scenarios. The analysis in this study is thus constrained to an offline, or batch learning scenario, *cf.* Section 2.1.1, where ML models are deployed after training and evaluation, and only updated or generally maintained based on triggers as visualized Figure 4.2.

Both active and passive drift detection approaches are further defined in the following, answering the second research question:

Research Question 2

What methods for the detection of concept drift in ML applications exist in the literature?

While the decision between active and passive drift detection is a major consideration within the proposed framework – I. in Figure 4.2 – several further design decisions are involved when implementing active and passive drift detection – II.a. and II.b. in Figure 4.2 which are elaborated upon within Sections 4.3 and 4.4. The findings are consequently summarized and suggestions for implementation derived in Section 4.5, aiming to answer the third research question:

Research Question 3

How should the detection of concept drift be implemented for ML applications used in process and condition monitoring of manufacturing processes?

4.3. Active drift detection

Several research articles exist in application-independent, general ML literature concerning methods to actively detect concept drift in deployed ML applications. To investigate the mechanisms of existing methodologies, a systematic literature review is conducted in this section. Consequently, a general architecture of active drift detection systems is derived, along with design methods and respective considerations for implementation.

4.3.1. Systematic literature review

Methodology

The literature review follows the methodology of Kitchenham and Charters [Kitc07] which splits the review process into three phases: planning the review, conducting the review and reporting the review results. Initially, research questions are defined and a review protocol is developed to guide the review process, including study selection criteria and data extraction methods. In the conducting phase, studies are identified and selected according to specific criteria and consequently evaluated for quality. Key data from these studies are collected and combined to address the research questions. In the final reporting phase, findings are presented and discussed, highlighting implications with respect to the defined research questions and future work.
Two Sub Research Questions (SRQs) are formulated to guide the analysis of the primary studies and answer the aforementioned research question:

- **SRQ1:** How can the identified approaches be structured and categorized?
- **SRQ2:** Which advantages, disadvantages and requirements do the identified approaches have?

The search was conducted using the ACM Digital Library[1], IEEE Xplore[2], and ScienceDirect[3] as databases. The following search string is adapted to fit each of the search engines of the publication databases: (*(drift OR shift) AND detect**) AND (*machine learning* OR *deep learning* OR *artificial intelligence*). The search is constrained to articles published in the past 15 years (2009-2023), to focus on contemporary research and recent advancements in addressing the challenge of concept drift.

Exclusion criteria The initial query yielded a total of 785 publications, combined from the three databases. After merging the search results from each database, duplicates were removed based on paper titles and Digital Object Identifiers (DOIs). For every of the remaining candidate papers, the paper title and, if required, the abstract was reviewed. Reasons for the exclusion of candidate publications include:

- publication is not directly concerned with concept drift detection (n=628),

[1] ACM Digital Library: *https://dl.acm.org/*
[2] IEEE Xplore: *https://ieeexplore.ieee.org/Xplore/home.jsp*
[3] ScienceDirect: *https://www.sciencedirect.com/*

- publication is not written in English (n=4),
- publication does not provide technical and implementation details of the proposed solution (n=32),
- publication is a survey or literature review (n=9),
- publication proposes a solution that is highly specific to a certain domain or data type or is not applicable to the type of problems this thesis focuses on (n=18),
- publication is concerned with online or sequential learning or the publication proposes a solution that depends on ground truth labels during operation (n=49).

While most of the exclusion criteria are self-explanatory, the last one requires additional consideration. As stated before, online or sequential learning is not relevant to the majority of applications in the considered use cases within manufacturing, thus the respective studies are excluded. While online learning requires ground truth labels during operation for continually training the ML model, the search additionally revealed a large number of methods for concept drift detection that assume the availability of ground truth labels for the detection of drift itself. Methods that rely on the availability of ground truth labels typically calculate performance KPIs such as the accuracy during model operation and flag significant drops as concept drifts. These techniques are referred to as *supervised* or *explicit* concept drift detection methods while techniques that do not rely on ground truth labels are referred to as *unsupervised* or *implicit* [Seth15; Seth17; Ab G20]. The assumption of label availability is not realistic in manufacturing scenarios, thus studies concerning supervised/explicit methods are excluded as well, *cf.* RO.d.
After the first review, 660 publications were excluded, primarily due to a bad content fit. Consequently, the full texts of the remaining 125 publications were skimmed and 45 publications were selected for the analysis. Within this round of exclusions, the majority of studies were excluded due to being focused on online learning or requiring true labels (n=49). Lastly, snowballing was conducted through the references in the selected studies, to identify additional relevant studies that were not revealed through the keyword search in databases, adding 13 additional studies for a total of 58. The final list of studies is included in Table A.1 within the Appendix and the findings are presented along the derived framework in the following sections.

Framework for active drift detection

Based on the identified studies, concept drift detection based on monitoring quantities of interest such as the input data is typically achieved in three stages: **Stage 1: data acquisition & windowing**, **stage 2: data modeling** and **stage 3: hypothesis testing**, e.g., [Lu18], as visualized in Figure 4.3.

In stage 1, a window of the most recent data points $\mathcal{D}_{\text{live}}$ as well as a window of reference data points $\mathcal{D}_{\text{ref}}$ are acquired to be compared in the subsequent stages. Even though concept drift detection methods that rely on model performance KPIs are excluded, Figure 4.3 shows both the input data $\boldsymbol{x}_t$ as well as the labels y_t for completeness. Both data windows are then abstracted in stage 2, typically involving dimensionality reduction or feature extraction and consequently compared in stage 3 using univariate or multivariate two-sample tests to determine whether the two windows represent the same data distribution at some chosen significance level α. The three stages are elaborated upon in detail in the following sections, along with the corresponding findings of the literature review and practical considerations regarding the targeted use cases in the manufacturing domain.

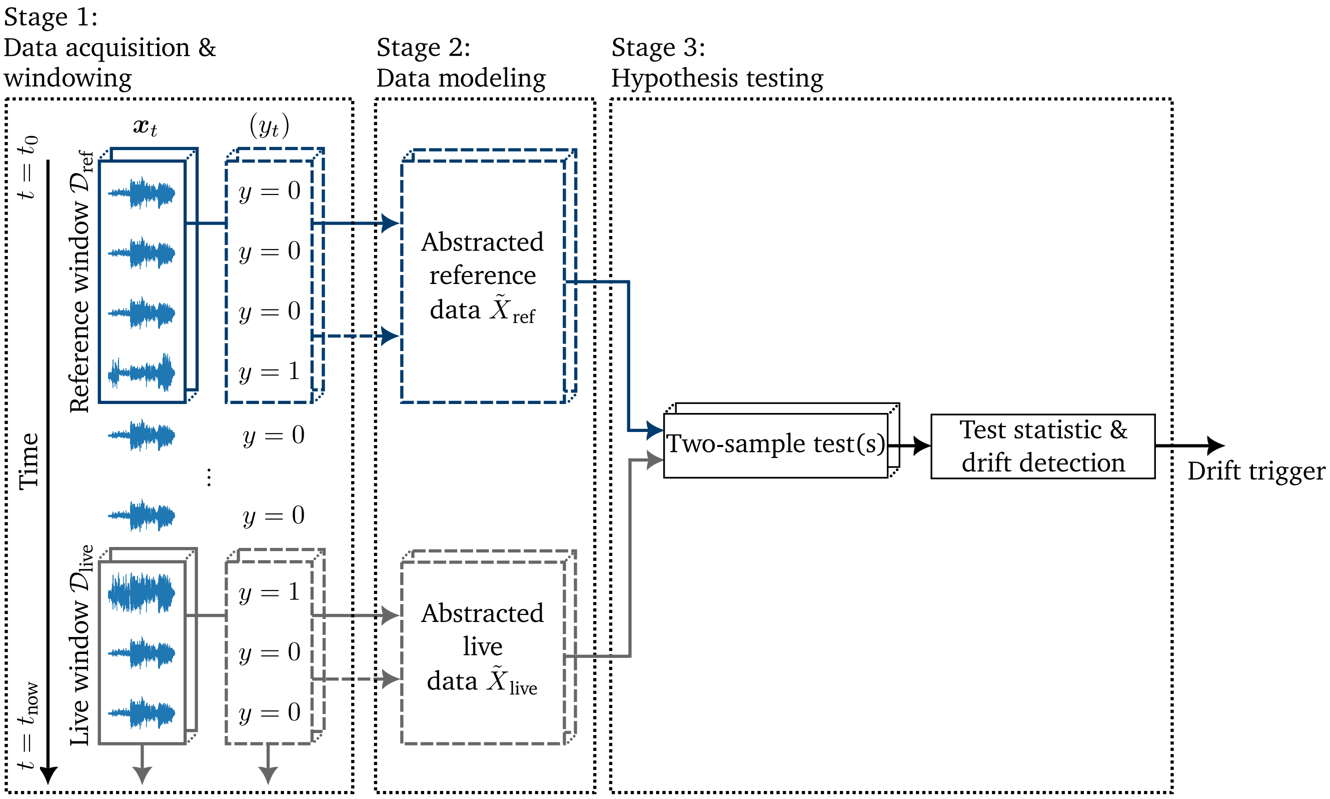


Figure 4.3.: Framework for active drift detection, divided into three stages: data acquisition & windowing, data modeling and hypothesis testing. Data is streaming in from top to bottom of the figure, divided into two windows (stage 1), abstracted (stage 2) and consequently used for hypothesis testing (stage 3). Own illustration.

Stage 1: Data acquisition and windowing

In this first stage, data is typically divided into two sets or windows, e.g., [Lind13; Pina15; Baie23]. The first window $\mathcal{D}_{\text{ref}}$ corresponds to the reference data that the model is expected to reliably work upon, thus characterizing a stable concept [Seth17; Lu18]. In the identified literature, the training or evaluation data that was acquired during model development in the earlier phases of the CRISP-ML(Q) process is often used for $\mathcal{D}_{\text{ref}}$ [Dos 16; Jawo17; Seth17]. While publications exist that also treat $\mathcal{D}_{\text{ref}}$ as a sliding window, e.g., [Žlio10a], a static reference set is more suitable for the target use cases as the ML model and thus its known concepts are typically static in manufacturing.

The live window $\mathcal{D}_{\text{live}}$ represents the data distribution that is currently processed by the model in operation. Thus, a sliding window is commonly used that shows the last $|\mathcal{D}_{\text{live}}|$ data points. The size $|\mathcal{D}_{\text{live}}|$ of the live window is an important hyperparameter of the active drift detection framework. The window should be large enough to capture the current data distribution and have sufficient robustness to noise, while small enough to be reactive to changes without a strong detection lag [Žlio10a; Gözü19].

Hyperparameters in drift detectors are problematic in practice as there is typically no data available to systematically tune them [Žlio10a], a fact that is often ignored in scientific studies, e.g., [Gözü19]. While the reference distribution is well known, the possible data distributions after a concept drift has occurred are unknown, thus making it impossible to quantitively tune parameters for detection. This stands in contrast to hyperparameters of the actual ML models, *cf.* Section 2.2, that can be tuned using validation datasets or CV.

In general, a window size should be chosen that corresponds to the expected reaction time of the application.

Most of the studies identified in the literature review are designed to operate on tabular data. However, the application scenarios relevant to this thesis and thereby also the case studies work on raw sensor data streams from sensors such as accelerometers and internal machine control signals. Thus, in contrast to general ML literature, a *data point* for the ML model typically corresponds to a segment of the recorded sensor data in time as visualized for $\boldsymbol{x}_t$ in Figure 4.3 rather than referring to a single measurement. The segment may contain univariate data of a single sensor or multivariate data of multiple sensors or other signals, such as internal machine control signals that have to be partitioned with respect to the use case on hand. The segmentation of the individual signals is visualized Figure 4.4.

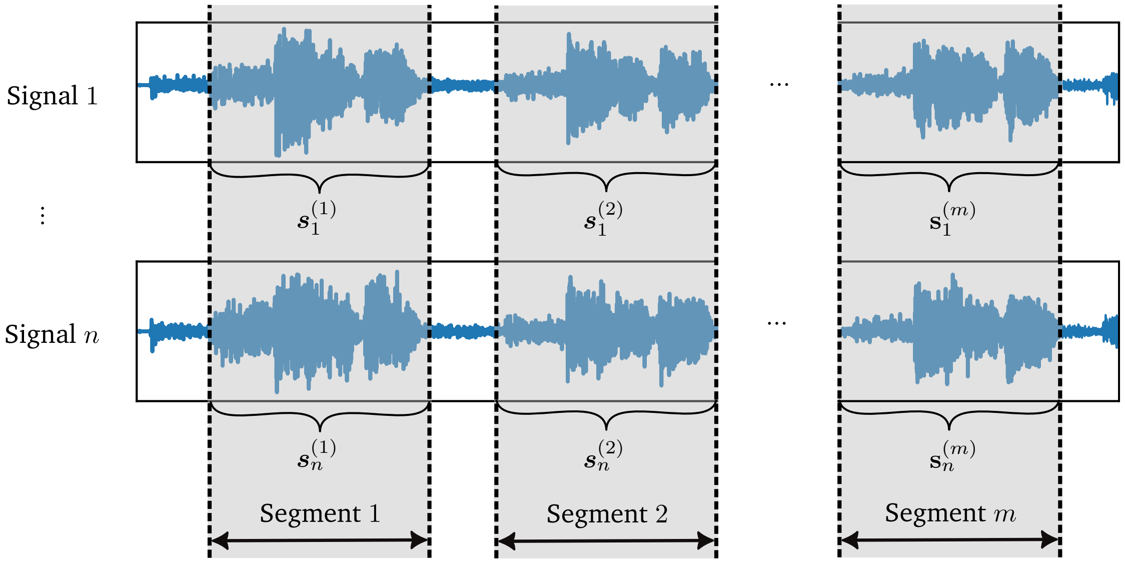


Figure 4.4.: Time series contextualization and partitioning. The multivariate time series with n signals is split into m semantic segments that are either equidistant or represent timespans of interest such as tool operations or specific sub-geometries of the manufactured part. Own illustration.

Segments are defined either by metadata such as Programmable Logic Controller (PLC) signals which allow semantic contextualization in time or alternatively by setting constant segment lengths and optional pauses in between considered segments. In the first option, semantic segments may be defined, e.g., as the time frame in which a specific geometric feature of the workpiece was machined as can be identified from the state of the NC program in time. The state of the NC program can typically be acquired from the PLC machine controller, *cf.* [Fert23]. Depending on the use case configuration, the semantic segments may be further divided into sub-windows of equal length. The start and end timestamps of the identified segments are applied to partition all individual signals within the multivariate time series, independent of the sampling rate. Generally, the segmentation method chosen for the concept drift detection should correspond to the windowing for the actual ML model. If, e.g., a tool condition monitoring application is developed for a milling tool, only data that corresponds to the active operation of this tool should be used for both the tool condition monitoring as well as the concept drift detection.

Following data acquisition & windowing, data modeling is applied in the second stage which is explained in the next section.

Stage 2: Data modeling

Data modeling involves dimensionality reduction of the data points in $\mathcal{D}_{\text{ref}}$ and $\mathcal{D}_{\text{live}}$ to extract quantities that meaningfully summarize the characteristics of the data distributions before the two-sample testing referred to as $\tilde{X}_{\text{ref}}$ and $\tilde{X}_{\text{live}}$ in Figure 4.3. This represents an important intermediate step as the raw data often has a high dimensionality or format that can render two-sample testing computationally infeasible [Raba19; Pian22]. There are strong differences within the implementation of this stage in the literature review studies with four major categories emerging that are visualized in Figure 4.5.

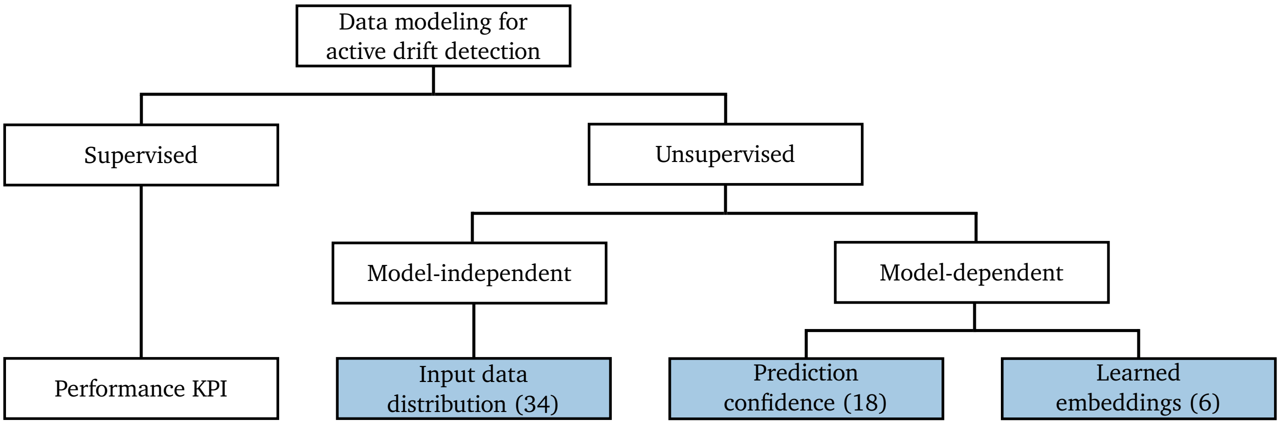

Data modeling for active drift detection
Supervised
Unsupervised
Model-independent
Model-dependent
Performance KPI
Input data distribution (34)
Prediction confidence (18)
Learned embeddings (6)

Figure 4.5.: Overview and categorization of the major options for data modeling with the number of respective studies in parentheses. Supervised techniques were excluded in the review but are visualized for completeness. Own illustration.

The first distinction can be made between supervised and unsupervised concept drift detectors. Supervised drift detection methods rely on true labels that can be compared with the model predictions to continuously compute performance KPIs or error rates. A significant increase in the error rate would indicate concept drifts. As supervised concept drift detection mechanisms are excluded from the review and the framework (RO.d) though, only unsupervised methods are elaborated further. Within unsupervised methods, a further distinction can be made between model-independent methods and model-dependent methods. While model-independent methods solely rely on the data that is processed by the ML model, model-dependent methods take the particular ML model that is used within the application into account as visualized for an example application in Figure 4.6.

Model-inpendent: Input data distribution The first category concerns methods for concept drift detection that utilize the input data of the ML model for concept drift detection, with or without additional processing before the two-sample testing. This is based on the assumption that most forms of concept drift, *cf.* Section 2.3.1, affect the input data distribution $P(X)$ and are thus detectable by monitoring $P(X)$ for significant changes. The majority of identified studies (34) fall into this category which can be attributed to the high diversity of approaches as well as to the large applicability of this category, as it can be applied regardless of the actual ML model that is used. Several studies in this category resort to direct monitoring of the input data without further data modeling techniques, e.g., [Dos 16; Raba19; Porw22], which is the simplest option. In the scenarios that are targeted within this thesis, this option is not advisable as elaborated for the first stage. In contrast, there are multiple studies that perform dimensionality reduction via feature extraction in the data modeling step, e.g., [Ditz11; Lee12; Cava16; Dos 16; Souz20a].

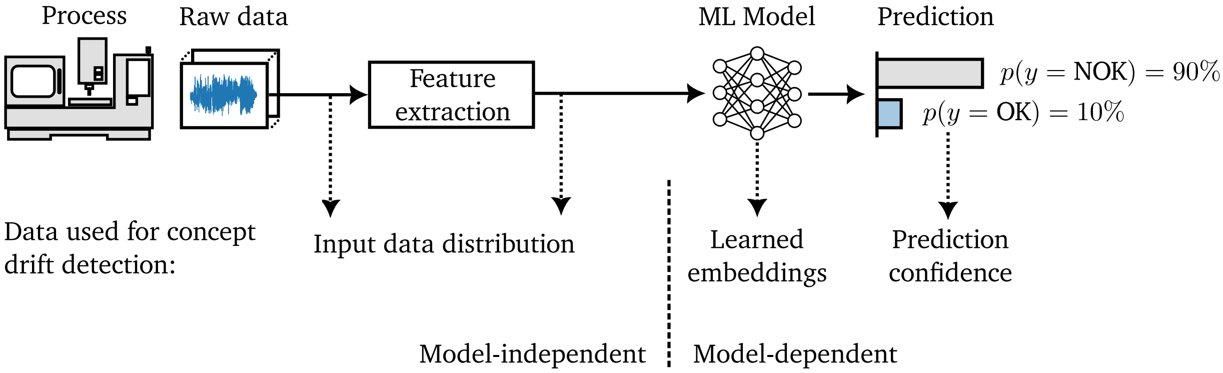


Figure 4.6.: Data modeling options for unsupervised active drift detection visualized in a predictive quality example application. Model-dependent options rely on quantities computed by the ML model – learned embeddings or prediction confidences – while model-independent options rely solely on the input data. Own illustration.

Typically, the same features are used for concept drift detection which are also used for the prediction of the actual ML application. Based on the raw time series data and the transformations introduced in Section 2.6, two categories of features are identified, which are suitable for different families of ML models that are commonly utilized in the target use cases: scalar features and image-like features.
In case of scalar features, feature calculators are used, which compute specific attributes for each window in a univariate time series as depicted in Figure 4.7. For a given window, multiple features – l in Figure 4.7 – may be calculated for each individual univariate time series. These are consequently concatenated with other features as well as features calculated on the other time series belonging to the same window, resulting in a feature vector $\boldsymbol{x}_i$ for a given time window i, describing the properties of each univariate signal in it. This feature vector is generated for each window in the dataset, thus resulting in a feature matrix $X \in \mathbb{R}^{m \times nl}$, with the m windows in its rows and $n \cdot l$ combinations of l feature calculators and n signals in its columns, corresponding to the descriptions in Section 2.1.1. Table B.1 in the Appendix contains a list of all time, frequency and time-frequency domain features that are utilized within the case studies of this thesis.

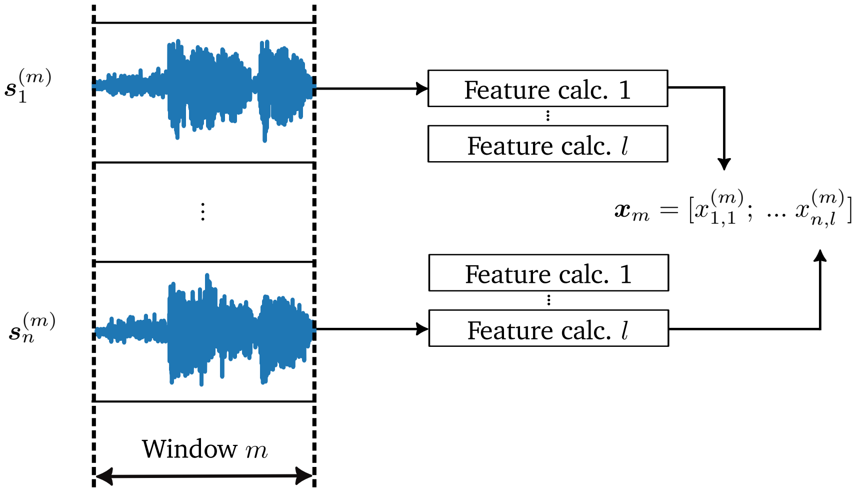


Figure 4.7.: Feature extraction from multivariate time series windows. The features, including statistical and frequency-based ones, are calculated separately for each signal and then concatenated in one vector $\boldsymbol{x}$ per window m. Own illustration.

The second category for extracting features from the raw time series data involves generating image-like spectrograms or scalograms using STFT and CWT respectively, as described in Section 2.6 and depicted in Figure 4.8. These two-dimensional representations may be treated as images that can be processed by suitable NN architectures such as CNNs and CAEs. The usage of CNNs on time-frequency representations has been shown to yield high performance in a number of recent publications, *cf.* [Vers17; Garc22; Bieg23]. While the utility of processing time series data this way has been proven empirically, there are differences between actual images vs spectrograms and scalograms, including the fixed semantic meaning of the ordinate as well as the not necessarily inherent hierarchical structure in spectrograms and scalograms. Both STFT and CWT are applied individually to each signal in a window, resulting in a matrix $X_i^{(j)}, i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}$ for every signal and window. For a single window, the resulting n matrices can be stacked to form a tensor $\boldsymbol{X}_m$, representing a multi-channel image.

In contrast to scalar features, image-like features cannot be easily monitored as they have very high dimensionality. Thus, if the ML model uses image-like features, the dimensionality should either be further reduced by techniques such as PCA or the prediction confidences of the ML should be used for concept drift detection instead, which is explained in the following sections.

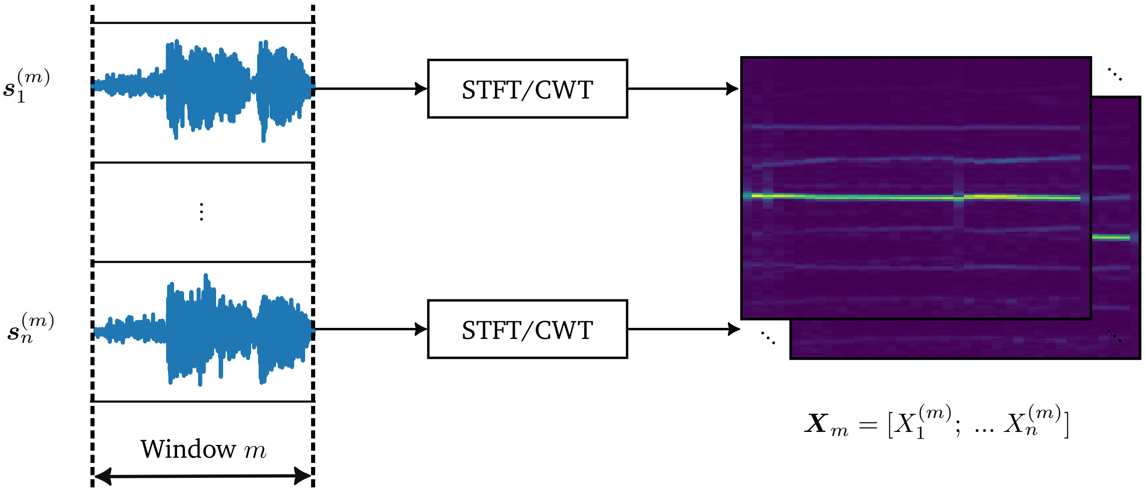


Figure 4.8.: For usage with CNNs, spectrograms or scalograms are generated using STFT or CWT respectively, resulting in an image-like matrix per signal. Own illustration.

While it is practical to utilize the features that are extracted for the ML application already in case of scalar features, approaches exist that utilize techniques such as singular value decomposition or PCA for dimensionality reduction and feature extraction, e.g., [Shan17]. More recently, another subcategory of methods based on the input data distribution emerged, which uses additional ML models for the purpose of detecting distribution shifts. These methods often resort to models that are commonly used for anomaly detection, such as the reconstruction error of AEs or restricted bolzmann machines, e.g., [Jawo17; Raba19; Kory21; Kami22]. These models are consequently trained on $\mathcal{D}_{\text{ref}}$ and the resulting reconstruction error is used as the abstracted reference set. Consequently, the reconstruction error is calculated for the live windows as well. A significant change in the reconstruction errors over time signalizes potential concept drift.

Another variant of using additional models is proposed in [Gözü19] and evaluates the capability of a model to distinguish between the reference data and the current live data. If the model can do so with high performance, there is likely a significant difference in the data distributions existing.

While having the advantage of being applicable regardless of the ML model chosen, model-independent methods have the commonly mentioned disadvantage of often being oversensitive and prone to false alarms [Seth17; King21; Baie23]. This downside can be explained by the fact that they are testing for all significant distribution deviations between $\mathcal{D}_{\text{ref}}$ and $\mathcal{D}_{\text{live}}$, not differentiating if the identified deviations will influence the model predictions and in turn the model performance. Additionally, some data formats cannot easily be used for two-sample tests without additional preprocessing, such as camera images, scalograms and spectrograms.

Model-dependent: Prediction confidence The category with the second-largest amount of studies encompasses model-dependent approaches that rely on the prediction confidences of the ML model that is being used in the application as visualized in Figure 4.6. Of the identified studies, 18 belong to this category. These methods are based on the assumption that the confidence and thus the uncertainty of the model in its predictions will significantly change if the data distributions change due to concept drift, e.g., [Lind13].
While some methods in this category are only applicable to certain types of models such as SVMs, e.g., [Seth15], or NNs, e.g., [Baie23], others are applicable, independently of the algorithm, as long as a confidence score is produced, e.g., [Žlio10a; Lind13; Lipt18; Raba19; King21].
ML models commonly compute univariate scores for binary classification problems and multivariate scores in case of multiclass classification problems as models produce one score per class option, e.g., via the softmax function in NNs, *cf.* Section 2.2. Multivariate scores can be reduced to univariate values via the Shannon entropy, *cf.* Equation (2.17) in Chapter 2, e.g., [Baie23]. Most anomaly detection models produce scores that can be interpreted similarly to confidences, *cf.* Section 2.2, and can thus also be used in this category of concept drift detectors.

Approaches in this category have the advantage of only being sensitive to changes in the data distribution that influence the model output, thus alleviating this specific disadvantage of model-independent methods. Furthermore, they strongly reduce the dimensionality for the two-sample testing, thereby lowering the computational effort and runtime of the drift detector and are not limited by the input format. Thus, model-dependent methods also allow concept drift detection when the input format is not easily usable for classic feature extraction, such as with camera images, spectrograms or scalograms.
The applicability of the techniques in this category depends on the type of ML model used as not all models output meaningful scores alongside their predictions as elaborated in Section 2.2. In existing studies, ensemble models such as RFs have shown superior performance in terms of sensitivity of the confidence scores to concept drift when compared to, e.g., NNs [Jour21c]. As several methods for enhancing the confidence estimation of NNs were proposed in recent years, a comprehensive comparison of the suitability of uncertainty estimation methods for usage in concept drift detection was performed and is presented in Appendix C. In essence, the study indicates that there is no significant performance gain in terms of concept drift detection when using seemingly more advanced methods for uncertainty estimation in NNs compared to using the *standard* softmax confidences introduced in Section 2.2. Ensembles are not exclusive to DTs and other models can be used in ensembles as well, e.g., [Seth17]. Alternatively, approaches exist that attempt to learn meaningful decision boundaries and confidence scores for models that do not have the ability, e.g., [King21].

Model-dependent: Learned embeddings More recently, with the first study published in 2020, *cf.* Table A.1, an additional category of model-dependent concept drift detection methods has emerged in the literature. This category uses embeddings for concept drift detection, which are the feature values in intermediate layers of deep NNs and CNNs as visualized in Figure 4.6. In the LeNet example architecture in Figure 2.12 in Chapter 2, embeddings may be taken from either of the last two layers before the output activation.

Six of the studies identified in the literature review belong to this category. The majority of these studies utilize the concept drift detectors in image classification use cases, e.g., [Banf22; Pian22].
Typically, the embeddings produced by the NNs are multi-dimensional but some authors propose to reduce the dimensionality by averaging the embedding vectors, e.g., [Acke20a].
The methods that belong to this category are only applicable to deep NNs and CNNs as only these models produce embeddings, in contrast to *classic* ML models such as RFs or SVMs. They share the advantages of confidence-based methods, specifically being more robust to drifts that do not influence the performance as well as being input-format independent. Additionally, they retain more information when compared to confidence-based methods, due to the commonly higher dimensionality of the embeddings when compared to confidences or the entropy.

Following data modeling, hypothesis testing is applied to compare the abstracted data windows which is explained in the next section.

Stage 3: Hypothesis testing

One or multiple two-sample hypothesis tests, *cf.* Section 2.4, are used within the final stage of the active drift detection framework to test whether the data distribution P_{live} of the current live data window after the modeling stage ($\tilde{X}_{\text{live}}$) originates from the distribution P_{ref} of the reference data window after the modeling stage ($\tilde{X}_{\text{ref}}$).

Therefore, the hypotheses used in the two-sample test are as follows:

$$H_0 : P_{\text{live}} = P_{\text{ref}} \Rightarrow \text{no concept drift}$$

$$H_1 : P_{\text{live}} \neq P_{\text{ref}} \Rightarrow \text{concept drift}$$

The null hypothesis H_0 describes the case of **no concept drift**, where the ML model is making predictions within its known domain and data distribution. Thus, the model performance should be comparable to the test set performance that was evaluated during development. The alternative hypothesis H_1 indicates **concept drift** if a significant difference between the current live data window and the reference data was observed. Thus, the model is operating outside of the reference data distribution and the prediction performance will likely be degraded. This corresponds to the *trigger* connection between model monitoring and model maintenance in Figure 4.2.

Two-sample tests involve calculating a test statistic, *cf.* Section 2.4, which quantitatively measures the discrepancy between the two distributions P_{live} and P_{ref}. This statistic serves as the basis for comparing the observed data against the expectations under the null hypothesis. The p-value, derived from the test statistic, quantifies the probability of observing data at least as extreme as the current findings, assuming H_0 is true [Lehm86; Chat18]. A decision on the hypothesis is made by comparing the p-value to a predetermined significance level α:

$$\text{Decision} = \begin{cases} \text{Reject } H_0 \text{ in favor of } H_1, & \text{if } p \leq \alpha \\ \text{Retain } H_0, & \text{if } p > \alpha \end{cases} \tag{4.1}$$

The studies analyzed within the literature review utilize various hypothesis tests, of which a popular selection, the KS test, the MMD test and the Chi-squared test are introduced in Section 2.4. Nonparametric tests like these are preferable for concept drift detection as oftentimes no assumptions about the distribution of the data can be made [Dos 16; Van 19]. The adequate choice of a hypothesis test primarily depends on the dimensionality and type of the data after the data modeling stage. For a univariate data modeling strategy,

such as using the prediction confidences, the KS test is often employed, e.g., [Žlio10b; Dos 16; Raba19]. While the KS test is designed for continuous values, the Chi-squared test is suitable for categorical values [Lehm86]. If the data modeling stage yields multivariate data, such as learned embeddings or features derived from the input data distribution, multiple univariate tests can be conducted on the individual feature dimensions, or multivariate hypothesis tests can be used. The MMD test is a commonly used nonparametric multivariate test employed in several analyzed publications, e.g., [Raba19; Cobb22]. If, instead, multiple univariate tests are used, the p-values of the individual univariate tests must be properly combined to prevent accumulation of the α-errors [Lehm86]. As suggested for similar scenarios in [Raba19], the Bonferroni correction [Blan95] can be used, which adjusts the critical α value to $\frac{\alpha}{n}$ where n is the number of individual two-sample tests that are performed. H_0 is thus rejected if the minimum p-value among all tests is less than $\frac{\alpha}{n}$.
Lastly, multivariate tests such as the MMD test have the theoretical advantage of being able to detect divergences that only manifest as unusual combinations of feature values, while a combination of univariate tests only checks the distributions of each feature individually. Thus, a combination of univariate tests may be unable to detect certain drifts.

The targeted use cases present certain challenges when applying two-sample hypothesis tests, which will be elaborated in the following section.

4.3.2. Challenges regarding hypothesis testing

Parts of this section have been previously published as a conference paper titled "A Nearest Neighbor-Based Concept Drift Detection Strategy for Reliable Tool Condition Monitoring" which appeared in the proceedings of the International Workshop on Software Engineering and AI for Data Quality in Cyber-Physical Systems in 2023 [Jour23b] and which was additionally reviewed and presented at the Neural Information Processing Systems (NeurIPS) 2023 Workshop on Distribution Shifts [Jour23c].

All of the introduced state-of-the-art hypothesis testing methodologies for active concept drift detection rely on the assumption that the samples in the reference and live windows are Independent and Identically Distributed (i.i.d.) under non-drift conditions. Thus, H_0 assumes that the data points in the windows are independently drawn from the same population. This assumption is problematic in the targeted use cases, especially in the case of condition monitoring, for multiple reasons:

- **Temporal correlation:** Even if a large live window is chosen, in use cases related to condition monitoring it will typically only show a certain sub-population of the full data distribution such as a certain wear state of a tool or component that is being monitored as wear states change slowly [Tora15]. The data points within the live window will be highly correlated, and, if the size of the live window is chosen small, even distances due to different times of day might be flagged as significant deviations in two-sample tests.

- **Sample selection bias:** During the acquisition or recording of training data for an ML model, it can be necessary to stage defects or provoke certain situations that show phenomena like strong wear or product defects as they are typically rare in actual production [Wues16]. This can lead to a class distribution in the training and reference data, e.g., the proportion of OK vs NOK parts, that does not correspond to the class distribution that is observed during the operation of the model, affecting all targeted use cases.

Thus, it can generally not be assumed that batches of recent deployment data are i.i.d. even if no concept drift is present. This is problematic as the i.i.d. assumption is essential for the two-sample tests used for drift detection. Concept drift detectors without further measures might thus raise a high number of false alarms and thus only provide low precision [Cobb22; Jour23b] which is problematic with respect to RO.b.
This issue is visualized for an exemplary synthetic condition monitoring dataset in Figure 4.9. The synthetic reference dataset is generated using two features in a diagonal ellipsis with three classes that show progressing wear states from the bottom left to the top right in three shades of grey.

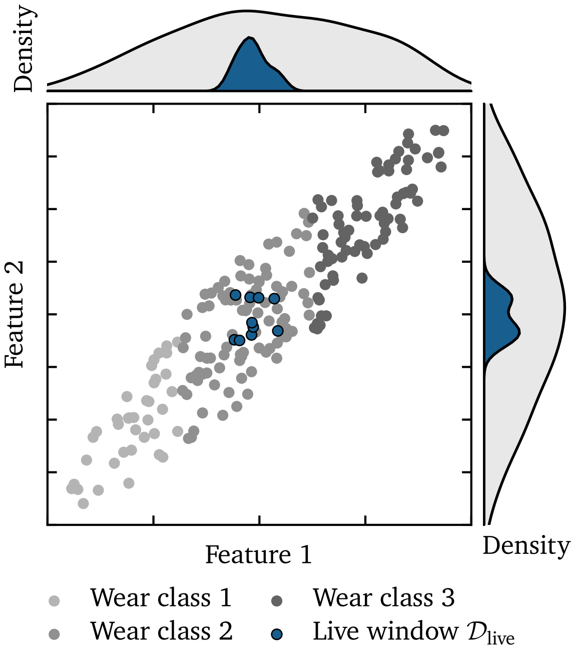
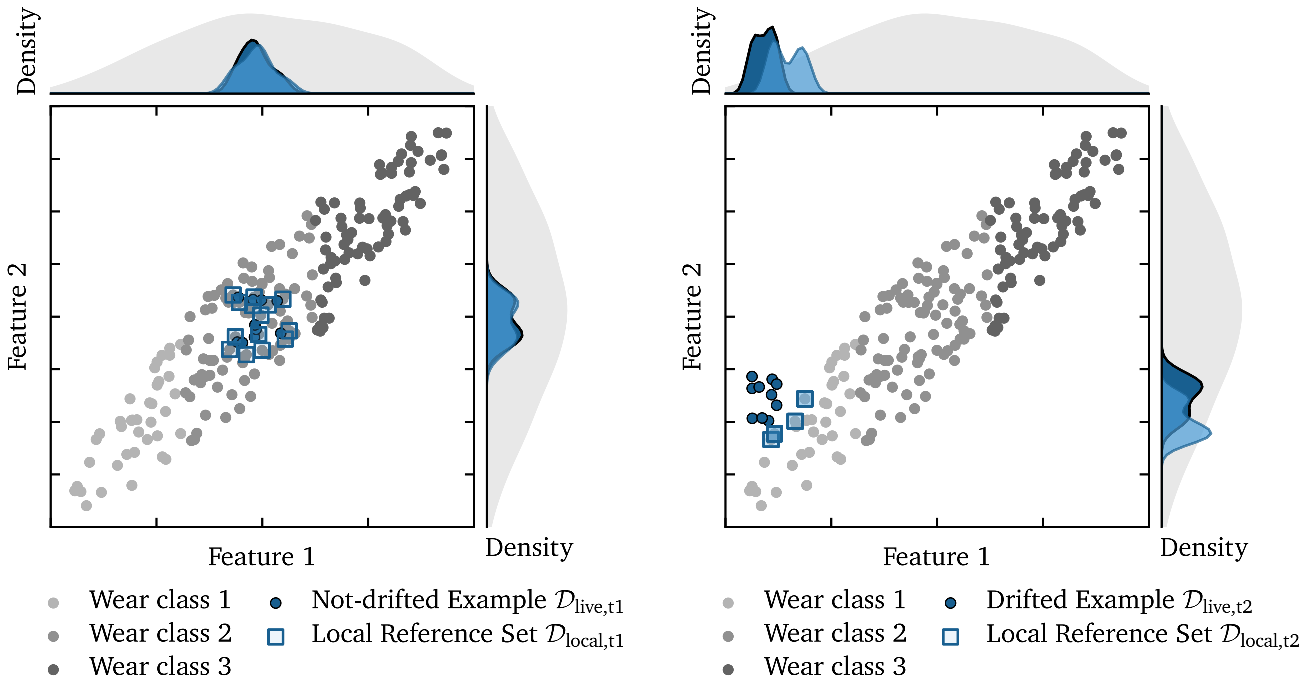


Figure 4.9.: Synthetic condition monitoring dataset with two features and three classes of progressive wear visualized in shades of grey as the reference window for drift detection. A live window is visualized in blue, clearly being within the bounds of the known feature space. Visibly, the densities of both features are strongly different and would thus yield a false positive concept drift detection in a two-sample test as this would compare the blue and grey densities. Own illustration.

This systemic issue is recognized in a recent publication that proposes *Context-Aware Drift Detection* [Cobb22]. In their approach, the authors extend a multivariate MMD-based two-sample testing approach for drift detection by a context variable. This context variable can be, among others, the classifier prediction or the time of day, and is consequently used to weigh the reference set and increase the relevance of data that corresponds to a context that is close to the current deployment context. The approach by [Cobb22] will be referred to as ContextMMD in the following.
Even if the classifier prediction is used as context, the intra-class variance may still yield false positives in use cases like condition monitoring as visible in Figure 4.9. Thus, a novel approach is developed in this thesis as a simple and computationally cheap, yet effective alternative: Localized Reference Drift Detection (LRDD). By choosing an adaptive local reference set to the current deployment data through nearest neighbor search, this approach aims to reduce false alarms for concept drift without the need to explicitly define a context variable. In this way, the approach produces a hybrid solution between outlier detection and distributional two-sample testing. The LRDD algorithm pseudocode is given in Algorithm 1 and visualized in Figure 4.10.

Algorithm 1 Localized Reference Drift Detection (LRDD)

```
Input: Reference dataset $\mathcal{D}_{\text{ref}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n_{\text{ref}}}$
Input: Current data window $\mathcal{D}_{\text{live,t}} = \{(\boldsymbol{x}_i, \hat{y}_i)\}_{i=1}^{n_{\text{live}}}$
Input: Number of neighbors $k$
Input: Significance level $\alpha$
Output: Drift detection result
Standardize $\mathcal{D}_{\text{ref}}$ and $\mathcal{D}_{\text{live,t}}$
Initialize $k$NN with $\mathcal{D}_{\text{ref}}$
Initialize an empty set of indices $I_{\text{NN}}$
for $\boldsymbol{x}_i, \hat{y}_i$ in $\mathcal{D}_{\text{live,t}}$ do
    Get $k$-nearest neighbors of $\boldsymbol{x}_i$ in $\mathcal{D}_{\text{ref}}$ where $y_i = \hat{y}_i$
    Update $I_{\text{NN}}$ with the indices of these neighbors
end for
Extract the refined reference dataset $\mathcal{D}_{\text{local,t}}$ from $\mathcal{D}_{\text{ref}}$ using the unique indices in $I_{\text{NN}}$
Compute $p$-value using a two-sample test
if $p \leq \alpha$ then return Drift detected
elsereturn No drift detected
end if
```



(a) Example for data distributions with a live window that is not drifted.

(b) Example for data distributions with a live window that is drifted.

Figure 4.10.: Exemplary visualization of the LRDD concept using a synthetic two-feature dataset with three classes that are distributed along an ellipsis shape. Two scenarios are shown: (a) a live window that is not drifted. The local reference set shows overlapping distributions. (b) a live window that is significantly drifted. The local reference set shows strongly distinct distributions, thereby enabling effective detection. Own illustration.

LRDD consists of the following procedure: After training of the ML model, a nearest neighbor model is fitted to the test split $\mathcal{D}_{\text{ref}}$ of the available dataset. During the operation of the ML model at time t, a data window $\mathcal{D}_{\text{live,t}} = \{(\boldsymbol{x}_i, \hat{y}_i)\}_{i=1}^{n_{\text{live}}}$ that contains the n_{live} most recent input data points $\boldsymbol{x}$ as well as the corresponding model predictions $\hat{y}$ up until time t is used to detect concept drift. Therefore, the k-nearest neighbors of the data points in $\mathcal{D}_{\text{live,t}}$ are queried from the nearest neighbor model, only considering data points in $\mathcal{D}_{\text{ref}}$ that have class labels y that correspond to the ML models estimates $\hat{y}$, similar to [Cobb22]. Following this step, duplicates in the resulting k-neighbor data points are removed. The resulting set of unique k-neighbors forms the localized reference set $\mathcal{D}_{\text{local,t}}$, which can be compared against $\mathcal{D}_{\text{live,t}}$ to check for concept drift.
This approach allows a refined comparison of the current data distribution with a sub-distribution of the reference set that is closest in feature space, therefore considering the correlation between the data points in $\mathcal{D}_{\text{live,t}}$ and preventing false positive detections when $\mathcal{D}_{\text{live,t}}$ is not i.i.d. and does not contain samples from all classes or feature regions in $\mathcal{D}_{\text{ref}}$. In case of drift, the samples in $\mathcal{D}_{\text{local,t}}$ are expected to be far away in feature space from the samples in $\mathcal{D}_{\text{live,t}}$ which can be detected through two-sample testing.
The two-sample test can be performed with any existing method for checking distributional equality such as permutation testing with MMD or a combination of univariate KS tests for all feature dimensions as explained before. The proposed approach has been evaluated using a tool condition monitoring dataset that was released in the scope of the 2010 challenge of the PHM society [PHM 10]. The evaluation has shown that LRDD significantly enhanced the precision in detecting drifts by minimizing false alarms for active concept drift detection based on the ML models input features. For the full details of this evaluation, the interested reader is referred to the respective publication ([Jour23b]). The approach is further validated in the first case study of Chapter 5.

4.4. Passive drift detection

In contrast to active drift detection, the passive approach, as stated in Definition 4, does not scan the data stream for changes that indicate concept drift directly but rather indirectly detects drift by assuming concept drift is linked to external triggers such as a change of production variants or predefined time intervals. Passive drift detection thus has the major caveat of potentially triggering model maintenance actions, *cf.* Figure 4.2, even if no actual concept drift occurred and, conversely, not being able to directly detect problems such as sensor defects when they appear. It is thus less precise than active concept drift detection and generally only preferable if active concept drift detection is not feasible or if the reasons for concept drift are isolated and well-known.

The primary reason for using passive drift detection is the ambiguity between concept drift detection and general anomaly detection as identified in Section 3.1.4. In an example application where a specific machine component or production process is monitored for defects using anomaly detection algorithms such as OC-SVM, the changes in the observed data from sensors that manifest from a persistent component defect that sporadically or gradually manifests are not distinguishable from concept drift that is caused by, e.g., a drifting sensor that does not negatively influence the production process which is monitored. In this scenario, false positive drift detections may arise from the process or condition monitoring task itself.
Thus, a decision rule can be derived: In the case of anomaly detection applications where anomalies are not transient but persistent, *cf.* Figure 4.11, only passive concept drift detection is applicable. The model maintenance phase is thereby either triggered periodically or by predefined triggers. Active concept drift detection would only be possible in a supervised way, using true label feedback which is typically unavailable in anomaly detection use cases.

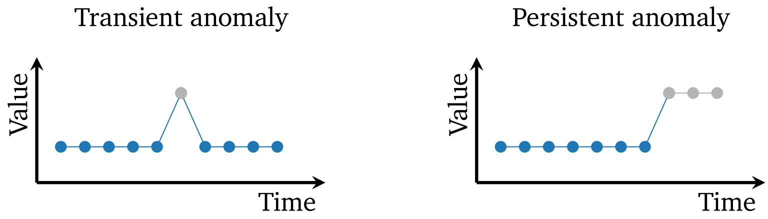


Figure 4.11.: Visualization of transient anomalies vs persistent anomalies. Persistent anomalies caused, e.g., by a component defect are not easily distinguishable from concept drift. Own illustration.

A secondary reason for passive drift detection is given if the reasons for concept drift are isolated and well-known, e.g., as identified using historical data.

Manufacturing processes and their output depend on a multitude of internal and external influence factors that can be suitable as triggers for model maintenance. Figure 4.12 visualizes exemplary influence factors in an Ishikawa diagram, a tool often used in QC to identify the root cause of a problem [Ishi90]. Ishikawa diagrams and the included factors are commonly structured using the so-called 5M model, considering Manpower (people), Material, Machine (equipment), Medium (environment) and Method (process).

ML applications utilizing passive concept drift detection are implemented in [Gori22] in the use case of anomaly detection for turbomachinery and in [Kahr22] in the use case of energy consumption prediction using periodic model updates and external triggers for updates, respectively. Furthermore, this path of the framework is validated in case study 3 in Chapter 5.

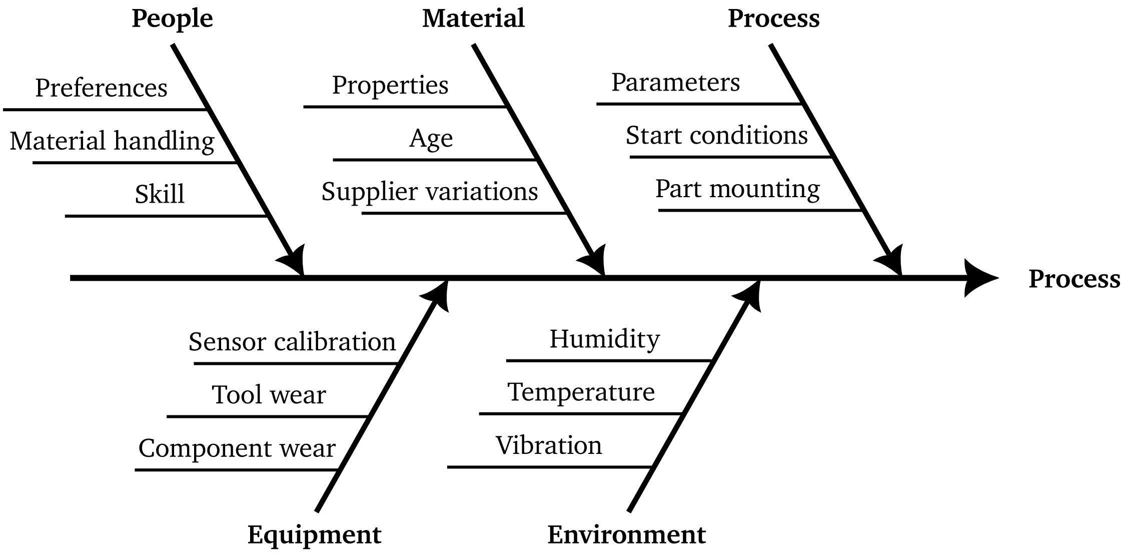


Figure 4.12.: Factors of influence on the manufacturing process visualized as an Ishikawa diagram [Ishi90]. Most of the factors are not constant over time, necessitating adaption of ML models that are deployed within the process. The 5M model can be used to identify potential concept drift triggers. Own illustration.

4.5. Implementation

Within this section, the decision between the presented approaches for concept drift detection is discussed, followed by practical aspects of the implementation.

The introduced approaches for concept drift detection are summarized together with their respective key requirements from the preceding sections in Table 4.1. On a high level, the table shows approaches with increasing detection capabilities from top to bottom. In parallel, the application requirements are increasing as well, as indicated in the rightmost column. Model performance KPIs are included for completeness, they are, if applicable, the most direct measure of concept drift in terms of practical model degradation. However, in the targeted real-world use cases, the requirement of continuous access to true labels typically cannot be fulfilled as it would make the ML use case itself obsolete.
The decision between an active or passive concept drift detection approach or the utilization of a combination of both is the first major decision a practitioner has to take when implementing the framework in Figure 4.2. As described, a passive approach should only be chosen if active detection is not possible or if external triggers are well-known and accessible, e.g., via an Manufacturing Execution System (MES) or via Open Plattform Communications Unified Architecture (OPC UA). In use cases that utilize ML for anomaly detection with permanent anomalies, only passive concept drift detection is applicable, with an exemplary implementation in case study 3 in Chapter 5.

Table 4.1.: Overview of strategies for concept drift detection with submethods and primary requirements. Model performance KPIs are included for completeness.

Category	Method	Submethod / Data modeling	Requirements
Passive	(a) Fixed trigger	(a.1) Time interval	R.a.1 Time interval can be determined with historic data
		(a.2) External trigger	R.a.2 Meaningful triggers are known and accessible
Active	(b) Model-independent monitoring	(b.1) Raw data	R.b.1 Model input data is accessible and meaningful
		(b.2) With preprocessing	R.b.2 Suitable preprocessing methods are available
	(c) Model-dependent monitoring	(c.1) Prediction confidences	R.c.1 Model is based on an architecture that produces meaningful confidences or anomaly scores, e.g., ensembles
		(c.2) Learned embeddings	R.c.2 Deep learning is used and embeddings are accessible
		(c.3) Model performance KPIs	R.c.3 Continuous access to true labels is available

Increasing detection capability ↓

Increasing implementation requirements ↓

If the operating conditions of the ML application have been well-defined in the early stages of the CRISP-ML(Q) process, a combination of both techniques can be applicable, where model maintenance is triggered both actively, through the described methods, and passively, through known factors such as a change of the product variant.
With regard to active concept drift detection, ML model-dependent concept drift detection approaches are generally preferable, as they are less susceptible to false alarms but have specific requirements concerning the suitability of the utilized ML model as outlined in Table 4.1. If, e.g., a DT is used in the application which provides neither prediction confidences nor embeddings, only a model-independent method can be used.

The reference window $\mathcal{D}_{\text{ref}}$ should ideally be filled with data that was acquired during actual production or with a held-out test dataset. Especially with ML model-dependent approaches, the training dataset should not be used, as the distribution of prediction confidences and embeddings may differ due to the training process [Sun19; Van 19]. Furthermore, it is shown in Section 4.3.2 that standard two-sample tests rely on the i.i.d. assumption, which is often violated in the targeted use cases. Thus, data modeling techniques that produce multivariate outputs should utilize the proposed LRDD approach.
Lastly, a significant challenge in configuring concept drift detection methods lies in the lack of testing data for drifted conditions as these, per definition, are unknown. Thus, parameters such as the significance threshold α cannot be precisely tuned beforehand. Instead, a first estimate needs to be made and then tuned iteratively. If, for example, an initial significance level of $\alpha = 0.05$ is chosen but regular false drift alarms (FP) are observed, the significance level should be iteratively lowered over time.

4.6. Summary

Chapter 4 introduces a framework for concept drift detection in ML applications used in process and condition monitoring of manufacturing processes.
First, Section 4.1 presented the DSR approach as the research process in this thesis. Importantly, four objectives for the framework have been defined, including high precision (RO.a) and recall (RO.b) of the concept drift detection, applicability to the targeted use cases in manufacturing and their respective common architectures (RO.c), and lastly, the objective that concept drift should be detected without access to labels after deployment (unsupervised) (RO.d). In Section 4.2, an overview of the framework and its intended role in concretizing the *monitoring* phase of the CRISP-ML(Q) process model is presented. The framework identifies and defines active and passive concept drift detection. Active concept drift detection is further structured and elaborated upon in Section 4.3. Active concept drift detection methods track changes in the distribution of the data that is consumed by the ML application or in the values that are computed by the ML model itself. A three-staged approach is derived from a systematic literature review that can be configured to fit the application requirements of a given ML use case in manufacturing. In the first stage, two windows are defined, containing data from the known reference concept ($\mathcal{D}_{\text{ref}}$) as well as a sliding window that shows the most recent data points ($\mathcal{D}_{\text{live}}$). In the second stage, data modeling, dimensionality reduction is performed to extract meaningful values for comparison ($\tilde{X}_{\text{ref}}, \tilde{X}_{\text{live}}$). In this stage, three major options are identified and discussed. In the last stage, hypothesis testing, it is assessed whether there are statistically significant differences between the two windows, which would indicate concept drift. Importantly, it is identified that standard two-sample tests yield false positive drift detections in the target use cases, as the live window contains highly correlated data points that do not cover the full value distribution, even if no concept drift occurred. LRDD is proposed as a method to refine $\mathcal{D}_{\text{ref}}$ for this scenario.
Section 4.4 introduced passive concept drift as an alternative method if active concept drift detection is not applicable. Passive concept drift detection triggers model maintenance using predefined external conditions or time intervals. The Ishikawa diagram is shown as a potential method for identifying influencing factors that may be used as triggers.
Lastly, Section 4.5 provides a holistic view of all presented approaches as decision guidance for practitioners.

5. Case studies and validation

This chapter presents the validation of the proposed framework for detecting concept drift in ML-based monitoring of manufacturing processes through its application in case studies. In three exemplary use cases, listed in Table 5.1, it is investigated whether the framework is applicable and whether it allows the effective design and implementation of concept drift detection mechanisms with respect to the ROs defined in Section 4.1, drawing conclusions and recommendations for practitioners in the process.
Section 5.1 presents the first case study which involves a tool condition monitoring use case applied to a CNC milling process on a testbed machine. Varying process conditions such as altered feed rates are used to create diverse datasets with and without drift for testing. Active drift detection is implemented and evaluated for different scenarios as recommended by the proposed framework. The second case study is presented within Section 5.2 and is subdivided into a process monitoring and a predictive quality use case. Two experiments are conducted under equal conditions on a testbed machine but with multiple months in between, introducing concept drift due to aging equipment. Active drift detection is implemented and evaluated within this case study as well. Section 5.3 presents the third case study which encompasses a condition monitoring use case implemented at the site of an industry partner. Here, an anomaly detection application concerned with permanent anomalies is developed. Thus, passive drift detection is employed and evaluated. Lastly, Section 5.4 discusses the overall findings of the case studies and derives practical recommendations.

Table 5.1.: Overview of case studies and their respective configuration.

Case study	Use case	Process	ML application	Drift detection
Case study 1	Condition monitoring	CNC milling	Supervised classification	Active
Case study 2	Predictive quality Process monitoring	CNC milling	Supervised classification Anomaly detection	Active
Case study 3	Condition monitoring	Pigment sieving	Anomaly detection	Passive

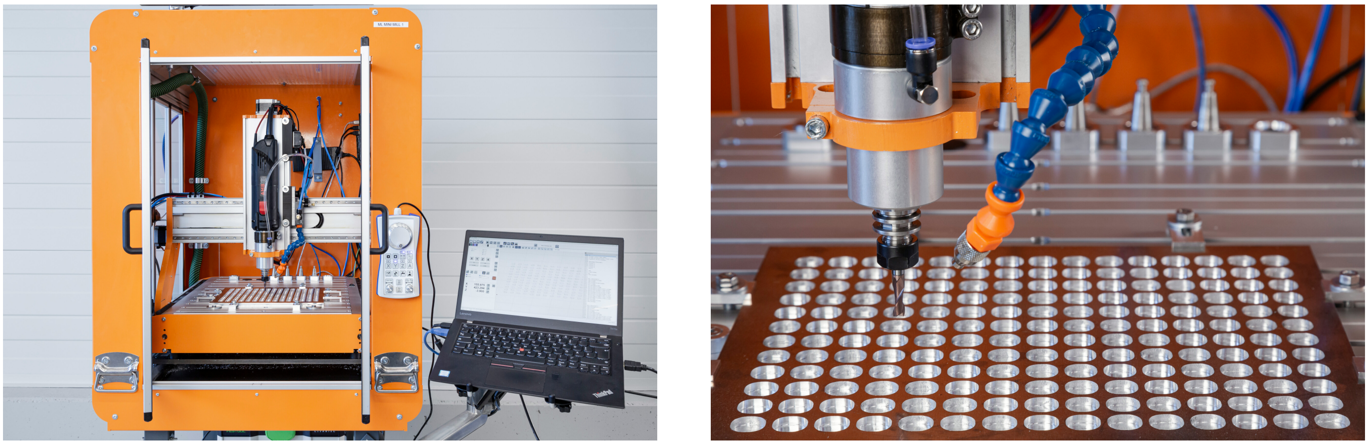
5.1. Case study 1: Tool condition monitoring in a CNC milling process

The first case study is concerned with an accelerometer-based tool condition monitoring system within a CNC milling process. Tool condition monitoring in milling processes is a popular research field with high practical relevance as presented in Section 2.5.1. This case study serves the purpose of validating the proposed framework and evaluating the performance of the proposed LRDD method as well as the underlying hypothesis that the i.i.d. assumption will be violated in condition monitoring use cases, even if no concept drift occurs. Multiple datasets are generated under specified reference operating conditions and various deviations, leading to the introduction of concept drift. Consequently, the developed framework for concept drift detection is applied to the use case, considering different scenarios and ML application designs.

5.1.1. Use case description and machine learning application design

Experimental setup and equipment

The case study incorporates a 3-axis table milling machine that is located in the FlowFactory learning factory at the Technical University of Darmstadt, Germany. Specifically, the study utilizes a Stepcraft M.500 3-axis gantry milling machine, *cf.* Figure 5.1 that is purposefully selected, configured and retrofitted for the experiments. This machine type is chosen for its accessibility and possibilities of reconfiguration due to its relatively small size as well as its relative ease of usage concerning NC programming and control software usage. In contrast to the stock configuration, the machine is retrofitted with high-frequency accelerometers and equipped with a minimal quantity lubrication system to prevent edge build-up on the milling tools. Furthermore, mounting points are installed on the table, as to provide a constant position of the workpiece relative to the machine axis. Both the lubrication system as well as the mounting points are used to ensure that the experiment conditions are repeatable and non-deliberate distribution shifts are avoided. As the implementation of the sensors did not involve structural changes to the machine, the presented system can be transferred to a wide range of similar machine tools and processes.



(a) CNC machine used for acquiring the datasets with the attached laptop running the NC control software WINPC-NC.

(b) Detail view of the milling tool above the aluminum plate.

Figure 5.1.: Stepcraft M.500 3-axis CNC milling machine used in the tool condition monitoring case study. In contrast to the stock configuration, a chassis, automatic lubrication, automatic tool exchanger, accelerometers, as well as a chip extraction system, were retrofitted to enable the experiments. Own illustration.

The machine is retrofitted with a Bosch CISS multi-sensor. The CISS multi-sensor incorporates an array of Micro-Electro-Mechanical System (MEMS) sensors, including a triaxial accelerometer (BMA280), gyroscope (BMG160) and acoustic emission sensor (AKU340). For this case study, only the accelerometer is utilized. The properties of the accelerometer and the inbuilt data acquisition hardware are summarized in Table 5.2. The sensor is mounted on the Z-axis of the machine, close to the spindle mount on the gantry. The mount point is chosen so that the sensor does not exhibit relative movement to the spindle and thus the milling tool, ensuring relatively constant vibration characteristics across the different positions of the machine. The CNC machine is controlled by the software *WINPC-NC* [Lewe24] which interprets G-code that is generated by Computer-aided Manufacturing (CAM) software from technical drawings of the desired workpiece geometry.

WINPC-NC enables the export of controller-internal signals to the host PC with a sampling rate of $5\,\mathrm{Hz}$ via the Windows registry. Relevant exported signals include:

- absolute axes positions on X, Y and Z,
- controller status, allowing the distinction between regular drive, tool changes, referencing and idle states, as well as,
- the currently processed program line in the NC program.

Table 5.2.: Properties of the accelerometer used in the Bosch CISS sensor for the tool condition monitoring case study.

Sensor property	**Value/Description**
Sensor name	Bosch BMA280
Measurement axis	X,Y,Z
Measurement range	± 16 g
Sensor type	MEMS
Data acquisition	Onboard via USB
Sampling rate f_s	$2000\,\mathrm{Hz}$
ADC resolution	14 bit

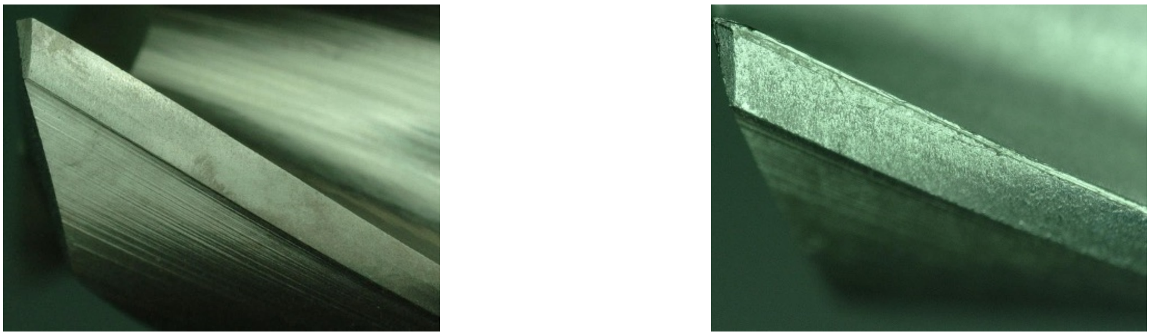
As this case study aims to investigate concept drift in tool condition monitoring applications, the general experiment setup is configured to allow for the generation of comprehensive datasets under varying operating conditions, while minimizing unnecessary complexity or external disturbances. The machines are used to mill patterns of pocket geometries into cold-cured aluminum (AlCuMg1) plates with a thickness of $8\,\mathrm{mm}$ and a size of $300\,\mathrm{mm}\times 250\,\mathrm{mm}$. During the experiments, 15×12 pockets are milled in a grid shape into the plate, *cf.* Figure 5.1b, in a single pass. The machining parameters are summarized in Table 5.3. The pocket geometry is chosen as it represents a common geometric feature in various industries [Fert23], while being simple enough to be machined using a single tool.

Table 5.3.: Tool condition monitoring case study cutting parameters.

Parameter	**Value/Description**
Spindle speed	$16\,000\,\mathrm{min}^{-1}$
Feed rate horizontal	$120\,\mathrm{mm\,min}^{-1}$
Feed rate vertical	$30\,\mathrm{mm\,min}^{-1}$
Depth of cut	$4\,\mathrm{mm}$
Tool flutes	2
Tool type	Solid carbide
Tool diameter	$6\,\mathrm{mm}$
Material	Aluminum 7075
Number of passes	1

Tool wear states

The tool condition monitoring use case considers two different tool conditions. The first condition represents relatively unworn tools that are able to produce the desired quality outcomes of a typical machining process while the second condition represents tools with high wear leading to artifacts such as poor surface finish and burrs [Zhou18] which can be observed on the aluminum plates used in the experiments. The wear of milling tools can be quantified by optically measuring the distance from the cutting edge to the end of the abrasive wear on the flank faces of the tool, defined as Verschleißmarkenbreite (VB) in [DIN 6583]. The maximum observed value, VB_{max}, is commonly used as the wear indicator in publications targeting tool condition monitoring in milling processes [Sagl03; Zhou18].



(a) Milling tool in new condition.

(b) Milling tool in worn condition.

Figure 5.2.: Two-flute solid carbide milling tool used in the tool condition monitoring experiments in new (a) and worn (b) states. Images are taken with a Keyence VHX 5000 digital microscope that is also utilized to optically measure the flank wear. Own illustration.

To ensure comparable magnitudes of the actual wear in the experiments, a special contraption was designed that drives a diamond-coated file along the cutting edges of the tools, while applying a constant spring-driven normal force and optically observing the wear level VB_{max} through a microscope. Through this mechanism, it is ensured that both wear classes have low intra-class variance, to minimize the influence of stochastic wear effects on the results. Similar setups have been proposed in recent tool condition monitoring publications such as [Fari20]. ATORN $6\,\text{mm}$ two-flute and three-flute solid carbide tools are used for the trials. Solid carbide tools are chosen to further reduce intra-class variance as the tools are expected to not experience significant wear throughout the trials except for the controlled wear that is introduced through the aforementioned methodology.

In the experiments, healthy tools are defined as tools with a $\text{VB}_{\text{max}} < 30\,\mu\text{m}$ while worn tools are defined as tools with a $\text{VB}_{\text{max}} \geq 30\,\mu\text{m}$. On average, healthy tools had a measured VB_{max} of $9.7\,\mu\text{m}$ while worn tools had a measured VB_{max} of $85.7\,\mu\text{m}$. Importantly, the magnitude of the tool wear is oftentimes considered still acceptable, depending on the use case. While stronger tool wear would likely be easier to detect in the accelerometer signals, this level is chosen to render the problem more challenging from an ML perspective.

Machine learning application design

A dataset representing the reference operating conditions for the use case is recorded using five tools in both healthy and worn conditions. The reference condition dataset is split into a training/validation and test dataset. The training/validation dataset is generated while milling a full aluminum plate using a set of three tools whereas the test dataset is generated using two additional tools on a different aluminum plate. Thus, differing instances of the plates, differing instances of the used tools as well as potentially slightly differing mounting points are assumed to be the expected variation in operating conditions during normal operation. In summary, two full aluminum plates are used to generate the reference condition training/validation and testing set, respectively containing 180 milled pockets. An optimized tool-to-pocket assignment is computed that ensures that the tools and their respective wear conditions are used evenly across the potential positions on the aluminum plate. An example image of the NC tool paths is visualized in Figure B.1 in the Appendix. This is done to prevent unintentional drift caused by varying vibration levels depending on the pocket position as well as an undesired correlation between pocket position and tool condition.

The host computer running the NC control software is recording two data streams during operation: low-frequency internal variables of the NC control software as well as the high-frequency 3-axis accelerometer data. Both are visualized for part of the reference condition dataset in Figure 5.3. The two data sources are synchronized using the clock of the host computer. Using the NC program state and the current machine coordinates, windows are extracted that contain the horizontal pocket milling processes, highlighted red in Figure 5.3. Furthermore, an NC program parser is developed to associate NC program line numbers to the currently active tool. To enable live predictions of the tool condition during machining, the identified time windows are further divided into $10\,\mathrm{s}$ windows that are treated as individual data points in the experiments. Consequently, features are extracted from the accelerometer data of the individual windows. Two types of features are extracted, as outlined in Section 4.3.1: scalar features, *cf.* Table B.1 in the Appendix, and image-like features as visualized in Figure 4.8. The fundamentals of the specific features are described in more detail within Section 2.6. Although there are various other possible features for processing raw data, these features have proven to be useful in comparable studies, *cf.* [Cai20]. The scalar features are used to train shallow models, including all non-deep learning models as well as MLPs.

As identified in Section 2.5.1, tool condition monitoring use cases are typically posed as supervised ML problems, employing a variety of models, including RFs, SVCs, kNNs, DTs as well as deep learning-based architectures with wear states or specific fault types as classes. For this experiment, a binary classification between tools in healthy conditions ($y = 0$) and tools with strong wear ($y = 1$) is adopted as introduced before, comparable to recent studies such as [Hess19]. To enable a comprehensive analysis, all aforementioned algorithms are trained and compared within the case study.
Five tools are used on two aluminum plates to generate the training and testing data for the reference conditions. Three of the tools are used for training and validation of the models, while two tools are used for testing the finalized models. Hyperparameter optimization is done using 5-fold CV on the data from the training tools and the final hyperparameter values for all models are documented in Table B.2 in the Appendix. The CNNs use a relatively simple architecture, inspired by LeNet [LeCu98], which is reported in Table B.3 in the Appendix.
The tool condition classification performance of all the ML models is listed in Table 5.4. Accuracy can be used to assess performance in this case study as the dataset has almost perfectly balanced class ratios.

Table 5.4.: ML model performance on the reference condition test dataset of the tool condition monitoring case study.

Metric	**RF**	**SVC**	***k*NN**	**DT**	**MLP**	**CNN (STFT)**	**CNN (CWT)**
Accuracy	0.991	0.992	0.970	0.954	0.993	1.000	0.999
F_1-Score	0.990	0.992	0.970	0.953	0.993	1.000	0.999

All models show high performance in classifying the tool condition with an average accuracy of 98.53%, even though training and testing data have been recorded using distinct instances of tools of the same type, indicating that all models can be used for tool condition monitoring under the defined reference conditions. The CNN using STFT features reaches perfect accuracy on the reference condition test set.

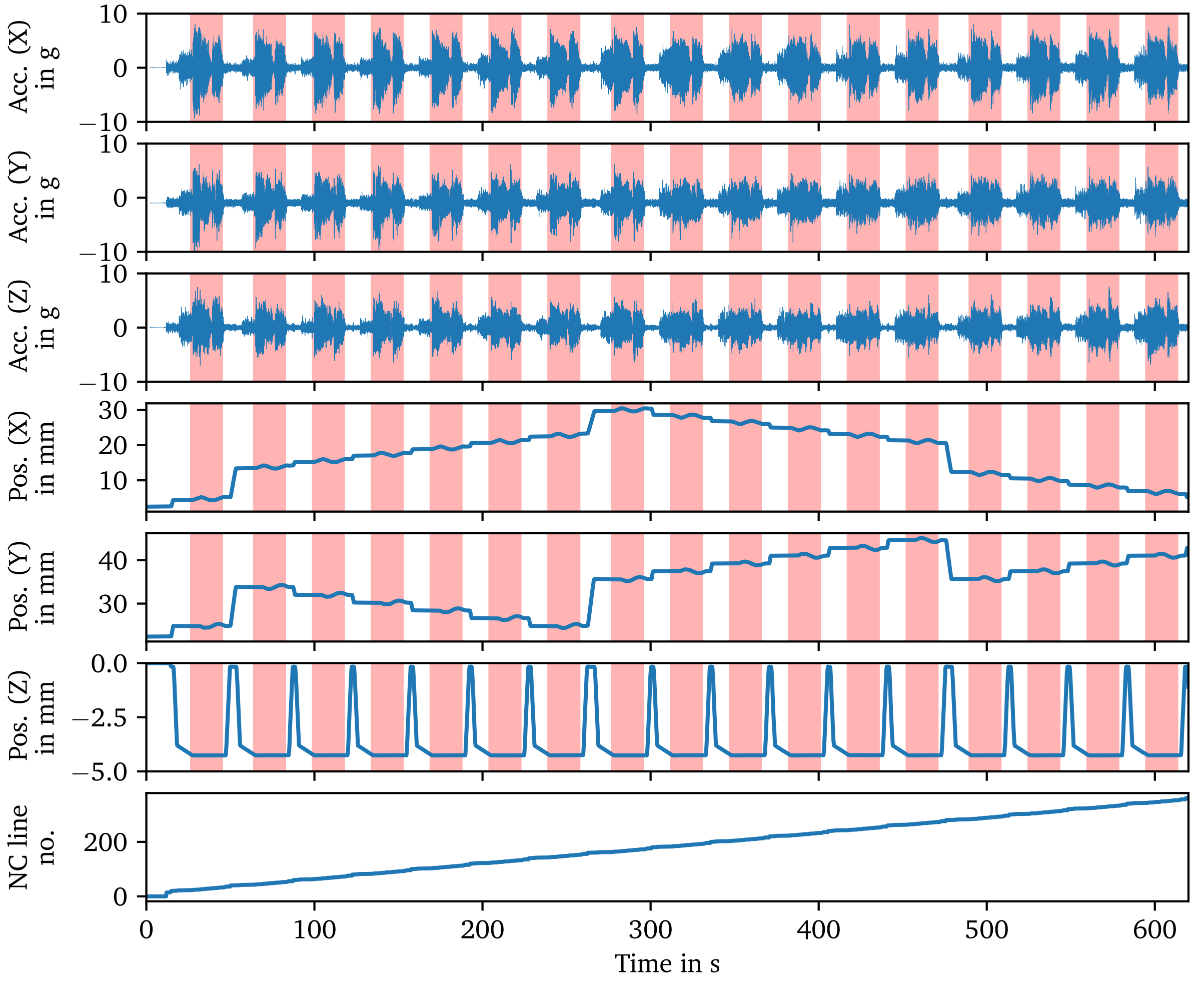


Figure 5.3.: Synchronized accelerometer and NC control data of the reference experiment in the tool condition monitoring case study. Identified windows of horizontal milling are shaded in red. One window corresponds to a completed pocket in Figure 5.1b. Not all recorded signals are shown. Own illustration.

5.1.2. Data collection with concept drift

After establishing the ML application on reference conditions with high classification accuracy, further datasets are generated which are used to evaluate the influence of concept drift on the application and consequently evaluate the applicability of the framework proposed in Chapter 4.

Four additional experiments are performed to generate a comprehensive dataset under varying operating conditions. In each of the experiments, data is again recorded for tools in healthy and worn conditions with different changes to the process. The process changes are assumed to be examples of realistic variations that might occur over longer terms of application usage. To the equipment operator, it is not obvious if the ML model is robust against these changes or if the performance will be degraded. The different conditions that are used in the experiments are detailed within Table 5.5. All additional configurations use a single aluminum plate each, thus yielding 180 pockets per configuration and 1080 pockets in total.

The introduced ML models are evaluated on the data recorded with the designated testing tools to get realistic performance estimates. Importantly, the models are only trained on the data from the designated training tools under reference conditions. Therefore they have not been exposed to data of the drifted conditions during training. The performance of the models on the altered operating conditions are reported in Table 5.6.

Table 5.5.: Reference and drift conditions in the tool condition monitoring case study.

Name	**Expected influence**	**Description**
Reference conditions	-	Default milling conditions and parameters as described in Table 5.3.
Spindle change	Low	Exchanged spindle motor with $1.65\,\mathrm{kW}$ instead of $1.4\,\mathrm{kW}$ in the original motor. Process parameters are unchanged with respect to Table 5.3.
Crooked mounting	Low	The aluminum plate is partially mounted on a $0.2\,\mathrm{mm}$ thick piece of copper, thereby introducing crookedness and leading to a slightly larger depth of cut.
Three-flute cutter	High	Exchanged cutting tools to three-flute tools out of the same material. Cutting parameters are unchanged with respect to Table 5.3.
Parameter drift	High	Feed rates both horizontally and vertically are increased by 10% with respect to Table 5.3.

Table 5.6.: Performance degradation of the tool condition monitoring application with respect to the introduced process changes. All models are trained on a subset of the reference conditions and consequently tested on the test split of the reference and the altered conditions.

Metric	**Configuration**	**RF**	**SVC**	***k*NN**	**DT**	**MLP**	**CNN (STFT)**	**CNN (CWT)**	**Avg.**
Accuracy	*Reference conditions*	**0.991**	**0.992**	**0.970**	**0.954**	**0.993**	**1.000**	**0.999**	**0.985**
	Spindle change	0.869	0.778	0.802	0.874	0.804	0.829	0.881	0.834
	Crooked mounting	0.961	0.963	0.958	0.926	0.965	0.963	0.967	0.958
	Three-flute cutter	0.524	0.774	0.440	0.557	0.616	0.451	0.496	0.551
	Parameter drift	0.624	0.452	0.487	0.660	0.506	0.478	0.553	0.537
F_1-Score	*Reference conditions*	**0.990**	**0.992**	**0.970**	**0.953**	**0.993**	**1.000**	**0.999**	**0.985**
	Spindle change	0.891	0.837	0.817	0.900	0.824	0.845	0.900	0.859
	Crooked mounting	0.942	0.945	0.938	0.894	0.948	0.945	0.951	0.937
	Three-flute cutter	0.390	0.809	0.354	0.503	0.612	0.122	0.455	0.464
	Parameter drift	0.502	0.472	0.057	0.582	0.144	0.099	0.345	0.314

Two important observations can be made from the results in Table 5.6. First, the introduced process variations lead to a significant decrease in performance. Crooked mounting introduces the lowest performance decrease which is expected as the slightly offset mounting will not influence the vibrations as strongly as the other variations.

Second, the performance degradation of the evaluated algorithms has a large variance but none of the algorithms are robust against the introduced process variations. Interestingly, the feature type – statistical vs image-like – does not seem to have a systematic influence on the robustness of the algorithms.

5.1.3. Concept drift detection

Summarizing the preceding section, the developed tool condition monitoring application is performing well under the defined operating conditions but is, at the same time, clearly vulnerable to concept drift in the form of operating conditions deviating from the reference. If, e.g., the NC program is updated with new process parameters or the tool type is changed, the performance will strongly degrade, which is not obvious to the machine operator. Other changes such as different mounting have less of an influence, *cf.* Table 5.6, making it difficult to specify exactly beforehand, under which circumstances the ML model needs to be checked or updated. Thus, the framework introduced in Chapter 4 is applied in this section.

Active vs passive drift detection

The presented ML application architecture uses supervised classification. Thus, the main reason for resorting to passive drift detection in case of unsupervised ML with permanent anomalies is not given and active drift detection can be used. Furthermore, the results of the reference condition experiments, *cf.* Table 5.6 and Figure 5.4, show that the models are able to classify the tool condition accurately on the test dataset, which has been created using different tools and aluminum plates than those used to train the models. Thus, there is no significant concept drift present in the operating condition variance that is observed for normal operation of the system and active drift detection is generally preferable in this case study.

In the next sections, the evaluation methodology for the active drift detection experiments is introduced, followed by a scenario analysis and the respective results.

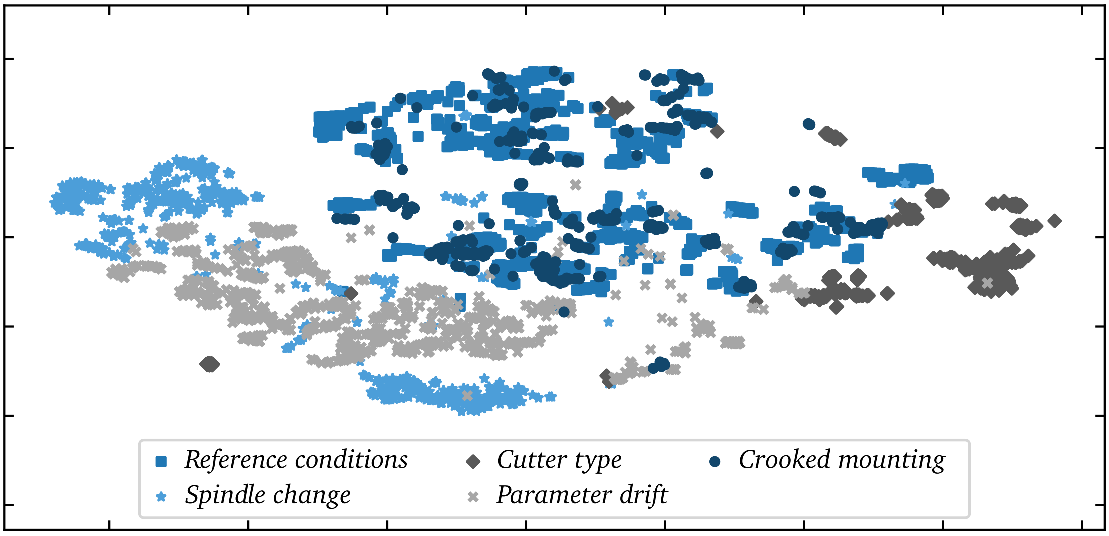


Figure 5.4.: t-SNE visualization of experiments conducted in the tool condition monitoring case study. Visualization is based on the features listed in Table B.1. Own illustration.

Experimental evaluation

The experimental evaluation aims to assess the capability of the active drift detectors to reliably detect significant distribution shifts that may lead to degradation of classification performance while not raising false alarms under unshifted conditions. Thus, this experiment is evaluated using the classification metrics introduced in Section 2.1.2. In this context, the positive – drifted – class is defined as a live window $\mathcal{D}_{\text{live}}$ that contains data points that do not belong to the reference conditions, e.g., the *spindle change* dataset, while the negative class is defined as a live window that contains data points that do belong to the reference conditions. The data distributions are visualized in a two-dimensional t-SNE plot in Figure 5.4. Most of the introduced process changes introduce strong drift of the feature distributions. Only the data points of the *crooked mounting* dataset do not exhibit a significant change in the feature space, mostly overlapping with the data points of the reference conditions.

For quantitative evaluation, the dataset corresponding to the training portion of the reference conditions is further split into a training set (50%) and validation set (50%). The reference condition validation set is used to initialize all drift detectors and thus corresponds to $\mathcal{D}_{\text{ref}}$ in Figure 4.3. Consequently, $n = 100$ sequences of length $|\mathcal{D}_{\text{live}}|$ are randomly chosen from the reference condition testing set as well as the datasets with changed process conditions and shown to the initialized detectors. Importantly, only the sequence starting points are randomly selected and the sequences thus contain data points in their original temporal sequence which are not randomly shuffled. This renders concept drift detection significantly harder but is more closely aligned to the real deployment of such a system as explained in Section 4.3.2.
The detectors produce p-values concerning the null hypothesis, *cf.* Section 4.3.1. Assuming a significance level α, the elements of the confusion matrix (TP, FP, TN, and FN) are computed for the concept drift detection performance. Based on these, precision, recall, and the F_1-score are reported.

Active drift detection configuration

For active drift detection, the three stages, data acquisition & windowing, data modeling and hypothesis testing need to be configured as visualized in Figure 4.3.
The tool condition monitoring ML application uses $10\,\text{s}$ data windows for feature extraction and prediction. Thus, these sample lengths are adopted for the drift detection framework as well. A live window size of $|\mathcal{D}_{\text{live}}| = 50$ is configured, to strike a trade-off between timely alarms in case of emerging concept drift and robustness against single outliers.
The recorded datasets, paired with the large variety of implemented ML models allow the investigation of different scenarios for data modeling. Therefore, three scenarios out of Table 4.1 are evaluated that are deemed to be of practical relevance in this use case and assume different levels of availabilities and configurations of the ML models as will be explained in the following:

- model-independent drift detection with preprocessing, b.2 in Table 4.1,
- model-dependent drift detection using prediction confidences, c.1 in Table 4.1, as well as,
- model-dependent drift detection using learned embeddings, c.2 in Table 4.1.

Scenario b.2: Model-independent drift detection with preprocessing In this scenario, it is assumed that there is either no direct access to the ML model for drift detection purposes or that the utilized model does not produce meaningful prediction confidences or learned embeddings, which is the case for a significant portion of models that are evaluated in this case study. Thus, the drifted operating conditions shall be detected by monitoring the distribution of the sensor data. Drift detection based on raw data – b.1 in Table 4.1 – is not seen as a viable alternative for two reasons: First, the high sampling rate of the sensor would lead to extremely long runtimes and computational costs of the drift detection algorithms. Secondly, existing literature shows that spectral and statistical features are required to capture the information regarding tool wear from the accelerometer time series data [Cai20]. Thus, the feature values that are used for prediction by the models, *cf.* Table B.1, are utilized for active drift detection as well. Compared to further preprocessing or dimensionality reduction techniques, this has the benefit of being computationally cheap as the features need to be calculated either way, and it further increases the relevance of the detections as the drift detection uses the same features as the models used for tool wear prediction. Importantly, this scenario only applies to the scalar features and cannot easily be applied to spectrograms or scalograms. In those cases, the subsequent scenarios – c.1 and c.2 – are applicable.

Four methods are consequently implemented and evaluated for two-sample testing as introduced in Section 4.3. All methods operate on the 48-dimensional data points containing the features listed in Table B.1.

- **KS**: Univariate KS tests are used on all feature dimensions individually aggregated using the Bonferroni correction as implemented in *alibi-detect* [Van 19].
- **MMD**: A multivariate MMD test is used on all feature dimensions using the implementation in *alibi-detect* [Van 19].
- **ContextMMD**: Contextualized version of MMDDrift as introduced in [Cobb22], *cf.* Section 4.3.2, using the implementation in *alibi-detect* [Van 19]. The model predictions are used as context variables.
- **LRDD**: The kNN-based reference set refinement method proposed in Section 4.3.2 applied as a preprocessing method before the actual two-sample testing. A parameter study is conducted concerning the optimal choice of two-sample test and number of neighbors k in the proposed LRDD approach with the results reported in Table B.4 in Appendix B.2. Notably, $k = 1$ with MMD testing is the best-performing approach and this configuration is consequently adopted for the further evaluation of the case study.

The F_1-scores of this scenario are reported in Table 5.7. Additionally, precision and recall of this experiment are listed within Tables B.5 and B.6 in the Appendix. From the results in Table 5.7, three main conclusions can be drawn. First, the low F_1-scores for KS and MMD confirm the hypothesis of Section 4.3.2 that *simple* two-sample tests are not suited for drift detection in condition monitoring applications. This can be explained by the high correlation of the data points in the live window over time and the fact that they only show a certain subset of the full reference conditions at a given time. Further analysis shows that the low F_1-scores for KS and MMD detectors are caused by the detector flagging almost all data samples as drifted, independent of whether they belong to the reference conditions or the drifted ones. This results in a very low precision, while the recall is very high, *cf.* Tables B.5 and B.6 in the Appendix. Second, using the proposed LRDD method or alternatively ContextMMD [Cobb22], the drift detectors reach significantly higher scores. Thus, the drifted conditions are detectable in all three datasets using the proposed methods in this scenario. In this context, it is notable that lower α values lead to higher performance in terms of F_1-scores as the number of false positive detections is lowered, consequently increasing the precision. Lastly, even the detectors that perform well for other datasets show low performance on the *crooked mounting* dataset. This is a desirable result, as crooked mounting does not decrease the performance of the ML model significantly as presented in Table 5.6. Thus, it should not be distinguishable from the reference operating conditions.

Table 5.7.: Drift detection performance in scenario b.2 of the tool condition monitoring case study. Mean F_1-scores and their standard deviations are reported over 10 experiment runs. *Crooked mounting** is marked with an asterisk as it did not degrade the performance of the application as listed in Table 5.6 and thus does not constitute concept drift.

Configuration	Data modeling	Detector	F_1-Scores $\alpha = 0.005$	 $\alpha = 0.01$	 $\alpha = 0.05$
Spindle change	Features	KS	0.667 ± 0.000	0.667 ± 0.000	0.667 ± 0.000
		MMD	0.668 ± 0.002	0.667 ± 0.001	0.667 ± 0.001
		ContextMMD	0.888 ± 0.008	0.868 ± 0.004	0.782 ± 0.019
		LRDD	$\mathbf{0.923 \pm 0.014}$	$\mathbf{0.902 \pm 0.011}$	$\mathbf{0.880 \pm 0.013}$
*Crooked mounting**	Features	KS	0.667 ± 0.000	0.667 ± 0.000	0.667 ± 0.000
		MMD	0.668 ± 0.002	0.667 ± 0.001	0.667 ± 0.001
		ContextMMD	0.266 ± 0.056	0.318 ± 0.035	0.471 ± 0.021
		LRDD	0.050 ± 0.022	0.098 ± 0.035	0.179 ± 0.050
Three-flute cutter	Features	KS	0.667 ± 0.000	0.667 ± 0.000	0.667 ± 0.000
		MMD	0.668 ± 0.002	0.667 ± 0.001	0.667 ± 0.001
		ContextMMD	0.888 ± 0.008	0.868 ± 0.004	0.782 ± 0.019
		LRDD	$\mathbf{0.922 \pm 0.014}$	$\mathbf{0.902 \pm 0.012}$	$\mathbf{0.879 \pm 0.013}$
Parameter drift	Features	KS	0.667 ± 0.000	0.667 ± 0.000	0.667 ± 0.000
		MMD	0.668 ± 0.002	0.667 ± 0.001	0.667 ± 0.001
		ContextMMD	0.881 ± 0.004	0.865 ± 0.003	0.782 ± 0.019
		LRDD	$\mathbf{0.920 \pm 0.015}$	$\mathbf{0.901 \pm 0.011}$	$\mathbf{0.880 \pm 0.013}$

Scenario c.1: Model-dependent drift detection using prediction confidences In this scenario, it is assumed that there is direct access to the ML model's prediction confidences, thus enabling model-dependent drift detection. The confidence monitoring is implemented using KS-tests on the (univariate) prediction entropy $H(p(y|\boldsymbol{x}))$ of each of the trained models. All models except the CNN are implemented using the *scikit-learn* python library [Pedr11] and use the corresponding implementations for confidence estimation, as introduced in Section 2.2. As SVCs do not possess an in-built mechanism for estimating prediction confidences, the confidences are estimated using isotonic regression [Plat99].

The evaluation results of this scenario are reported in Table 5.8. Again, precision and recall values for this experiment are listed in Tables B.7 and B.8 in the Appendix. The results in Table 5.8 indicate that only the monitoring based on the RF confidences provides satisfactory detection performance. While the RF confidence-based detection even outperforms some of the approaches in scenario b.2, all other models perform poorly in this scenario. This follows previous studies, indicating that ensemble algorithms like the RF provide more meaningful confidence estimates for data unseen during training than other algorithms [Diet00; Laks17; Jour21c]. The experiment results further indicate that if univariate confidences or entropies are used, the i.i.d. problem described in Section 4.3.2 and indicated in scenario b.2 does not arise and unmodified two-sample tests show high drift detection performance. This is most likely due to the confidence not depending on the current class as much as the feature distributions. The additional usage of the proposed LRDD method did not show performance increases in this scenario.

In conclusion, prediction confidence-based drift detection in conjunction with a univariate KS-test can be utilized effectively if models are used that provide well-calibrated confidences.

Table 5.8.: Drift detection performance in scenario c.1 of the tool condition monitoring case study. Mean F_1-scores and their standard deviations are reported over 10 experiment runs. *Crooked mounting** is marked with an asterisk as it did not degrade the performance of the application as listed in Table 5.6 and thus does not constitute concept drift.

				F_1-Scores	
Configuration	**Data modeling**	**Detector**	$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.05$
Spindle change	Entropy	KS (RF)	$\mathbf{0.920 \pm 0.018}$	$\mathbf{0.922 \pm 0.016}$	$\mathbf{0.886 \pm 0.020}$
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.655 ± 0.005	0.660 ± 0.005	0.668 ± 0.003
		KS (kNN)	0.281 ± 0.060	0.373 ± 0.073	0.460 ± 0.054
		KS (MLP)	0.616 ± 0.035	0.618 ± 0.036	0.651 ± 0.018
		KS (CNN)	0.637 ± 0.005	0.641 ± 0.005	0.658 ± 0.005
*Crooked mounting**	Entropy	KS (RF)	0.223 ± 0.048	0.285 ± 0.037	0.410 ± 0.028
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.665 ± 0.008	0.666 ± 0.006	0.667 ± 0.004
		KS (kNN)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (MLP)	0.510 ± 0.025	0.552 ± 0.021	0.610 ± 0.009
		KS (CNN)	0.670 ± 0.001	0.670 ± 0.001	0.670 ± 0.003
Three-flute cutter	Entropy	KS (RF)	$\mathbf{0.935 \pm 0.015}$	$\mathbf{0.932 \pm 0.015}$	$\mathbf{0.887 \pm 0.020}$
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.615 ± 0.013	0.639 ± 0.012	0.664 ± 0.006
		KS (kNN)	0.408 ± 0.038	0.463 ± 0.032	0.541 ± 0.026
		KS (MLP)	0.734 ± 0.022	0.735 ± 0.023	0.749 ± 0.014
		KS (CNN)	0.497 ± 0.002	0.502 ± 0.003	0.499 ± 0.004
Parameter drift	Entropy	KS (RF)	$\mathbf{0.930 \pm 0.015}$	$\mathbf{0.928 \pm 0.014}$	$\mathbf{0.887 \pm 0.020}$
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.670 ± 0.005	0.670 ± 0.004	0.670 ± 0.003
		KS (kNN)	0.031 ± 0.022	0.048 ± 0.027	0.071 ± 0.026
		KS (MLP)	0.665 ± 0.038	0.676 ± 0.024	0.696 ± 0.018
		KS (CNN)	0.638 ± 0.003	0.663 ± 0.007	0.670 ± 0.003

Scenario c.2: Model-dependent drift detection using learned embeddings In this scenario, it is assumed that deep learning models are used that provide learned embedding values that are accessible. This scenario is partially similar to the first one, mainly exchanging the *handcrafted* features in Table B.1 by features which are learned and computed by the model itself. For the experiments, the 32-dimensional output of the first fully connected layer within the CNN architecture is used, *cf.* Table B.3. The same four detector variants as in scenario b.2 are used. The evaluation results of this scenario are reported in Table 5.9 with precision and recall values reported within Tables B.9 and B.10 in the Appendix.

The most notable result of the evaluation in this scenario is the similarity of the results to scenario b.2. The same problem of a high number of false positives and thus a low precision arises with the *simple* two-sample tests (KS and MMD). Again, this problem is effectively mitigated using the proposed LRDD method or ContextMMD [Cobb22]. Importantly, the results for configurations such as *parameter drift* show, on average, the highest F_1-scores of the three analyzed scenarios. This is explainable by the fact that the CNN embeddings are both model-specific, as they represent the learned features of the model, and are still fine-grained when compared to the model confidences.

Table 5.9.: Drift detection performance in scenario c.2 of the tool condition monitoring case study. Mean F_1-scores and their standard deviations are reported over 10 experiment runs. *Crooked mounting** is marked with an asterisk as it did not degrade the performance of the application as listed in Table 5.6 and thus does not constitute concept drift.

				F_1-Scores	
Configuration	**Data modeling**	**Detector**	$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.05$
Spindle change	Embeddings (CNN)	KS	0.670 ± 0.001	0.670 ± 0.001	0.670 ± 0.001
		MMD	0.670 ± 0.001	0.670 ± 0.001	0.667 ± 0.001
		ContextMMD	$\mathbf{0.898 \pm 0.004}$	$\mathbf{0.890 \pm 0.005}$	0.856 ± 0.005
		LRDD	0.841 ± 0.023	0.870 ± 0.009	$\mathbf{0.901 \pm 0.005}$
*Crooked mounting**	Embeddings (CNN)	KS	0.670 ± 0.001	0.670 ± 0.001	0.670 ± 0.001
		MMD	0.670 ± 0.001	0.670 ± 0.001	0.667 ± 0.001
		ContextMMD	0.385 ± 0.055	0.442 ± 0.075	0.523 ± 0.081
		LRDD	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Three-flute cutter	Embeddings (CNN)	KS	0.670 ± 0.001	0.670 ± 0.001	0.670 ± 0.001
		MMD	0.670 ± 0.001	0.670 ± 0.001	0.667 ± 0.001
		ContextMMD	$\mathbf{0.898 \pm 0.004}$	0.890 ± 0.005	0.856 ± 0.005
		LRDD	0.895 ± 0.004	$\mathbf{0.891 \pm 0.005}$	$\mathbf{0.923 \pm 0.007}$
Parameter drift	Embeddings (CNN)	KS	0.670 ± 0.001	0.670 ± 0.001	0.670 ± 0.001
		MMD	0.670 ± 0.001	0.670 ± 0.001	0.667 ± 0.001
		ContextMMD	0.898 ± 0.004	0.890 ± 0.005	0.856 ± 0.005
		LRDD	$\mathbf{0.949 \pm 0.023}$	$\mathbf{0.974 \pm 0.016}$	$\mathbf{0.993 \pm 0.002}$

5.1.4. Interim summary

This case study shows an exemplary implementation of the developed framework in a real-world tool condition monitoring use case. A diverse dataset is generated that represents both reference operating conditions as well as multiple drifted conditions recorded under realistic variations of the process parameters and the general setup. Several ML models are trained on the reference conditions and tested on both reference conditions and the drifted conditions. It is shown that the performance of all models remains stable in reference conditions but degrades strongly in the drifted conditions, highlighting the necessity of continuously checking deployed ML applications in manufacturing for concept drift.

An active drift detection approach is selected and implemented in three different scenarios, showing that the drifted operating conditions can be reliably detected in all of them using the identified methodology, achieving objectives RO.a-RO.d for this case study.
Importantly, it is shown in two scenarios, specifically for scenarios b.2 and c.2 that *simple* two-sample testing fails in condition monitoring scenarios within manufacturing as even for the reference conditions, the live data only shows a subset of the whole data distribution and the data points are highly correlated over time, violating the i.i.d. assumption. In the evaluated cases, the KS and MMD tests yield a high number of false positives, thereby flagging even the reference conditions as drifted, confirming the assumptions stated in Section 4.3.2. Consequently, the proposed LRDD approach is validated in these two scenarios, showing strongly improved detection rates.
Furthermore, the results validate the assumption that model-dependent drift detection techniques are to be typically preferred over model-independent drift detection methods, as they achieve slightly higher performance values throughout the experiments.

Lastly, the experiments show that choosing a suitable significance level α presents a major challenge in implementing a drift detection framework, as no single α value is optimal in each experiment. Thus, the significance value should be adjusted throughout the application lifetime.

5.2. Case study 2: Predictive quality and process monitoring in a CNC milling process

The second case study is concerned with predictive quality and process monitoring in a CNC milling process based on NC control-internal signals of the CNC machine. Process monitoring based on internal machine signals is highly relevant for commercial applications as machine tools are increasingly equipped with sophisticated edge computing solutions that enhance data availability while not relying on external sensors [Fert22a; Fert22b].

This case study serves the purpose of validating the proposed framework for additional use cases and data types when compared to the previous one. Importantly, process monitoring often involves anomaly detection ML models, which were not used in the first case study. Two datasets are used that were originally presented in [Fert22b] and which have been recorded in equal settings but 1.5 years apart. Consequently, the influence on ML model performance is analyzed and the developed framework for concept drift detection is applied to the use case, again considering different scenarios and application designs.

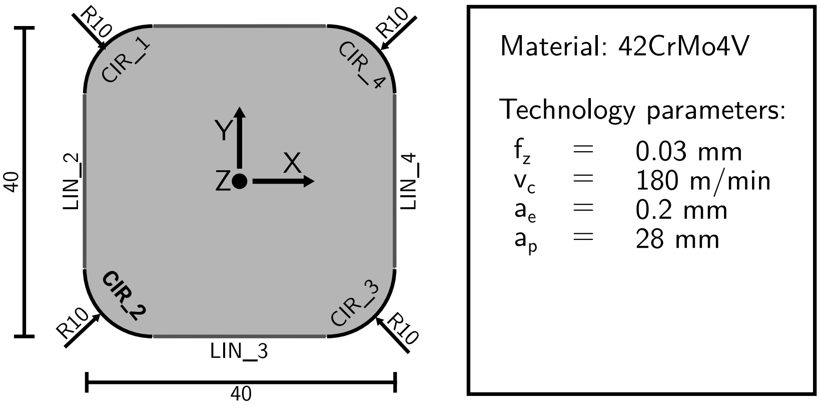


Figure 5.5.: Schematic representation of the part geometry and quality-relevant features in the predictive quality case study. The binary quality prediction of CIR_2 is analyzed within the case study. Figure corrected and extracted from [Fert22b].

5.2.1. Use case description and machine learning application design

Experimental setup and equipment

As this dataset has been previously published, only a concise summary is presented in this section and the interested reader is referred to the respective publications, *cf.* [Fert22a; Fert22b], for the full details regarding the dataset, the experiment setup and the data segmentation process. The dataset for this case study was generated in the TEC-Lab at the Technical University of Darmstadt, Germany, and captures the milling process of pocket geometries, *cf.* Figure 5.5. In the analyzed process, a total of 392 pockets are milled into eight plates of 42CrMo4V steel in a 7×7 grid pattern. The part geometry and machining parameters are summarized in Figure 5.5. Each pocket consists of seven quality-relevant geometric elements with defined tolerances.

The parts are manufactured in the summer of 2020 on a DMG MORI 3-axis DMC 850 V machining center, which is equipped with a Sinumerik 840D sl control system. Additionally, an edge computing solution was installed that provides access to NC control-internal signals at a frequency of $500\,\mathrm{Hz}$. The recorded signals of this dataset are summarized in Table 5.10. After the production, the quality-relevant geometric features were measured in a coordinate measurement machine. The test series served the purpose of investigating process-parallel quality predictions under near-production conditions. Thus, various perturbations were introduced during the milling process, which partially resulted in quality deviations.

Table 5.10.: Available NC controller signals in the predictive quality case study. All signals are recorded at $500\,\mathrm{Hz}$.

Signal name	**Axis**
CMD_SPEED	X, Y, Z, Spindle
CTRL_DIFF	X, Y, Z
CURRENT	X, Y, Z, Spindle
linENC_POS	X, Y, Z
ENC_POS	Spindle
POWER	X, Y, Z, Spindle
TORQUE	X, Y, Z, Spindle

Machine learning application design

The quality measurements are used to assign a binary label $y \in \{\text{OK}, \text{NOK}\}$ to each quality-relevant feature, which is the target variable for the ML task. In this case study, the classification regarding the CIR_2 feature, *cf.* Figure 5.5, being in (OK) or out of tolerance (NOK) is exemplarily considered.
In contrast to the authors of the original publications [Fert22a; Fert22b], this thesis defines the positive class as the (rarer) NOK class as the dataset shows a strong class imbalance and is biased towards the majority (OK) class making up 78% of the pockets. This is done to enhance the meaningfulness of the metrics as especially precision and recall strongly depend on the class balance. Most metrics assume that the positive class is the class of interest, *i.e.* that should be detected, while the negative class is the more common, general case [Murp12; Agga17].

The machine control signals are partitioned into time segments each corresponding to the milling process of the geometric feature CIR_2 in an individual pocket. The partitioning is achieved based on the NC controller state using the slicing algorithm presented in [Fert22a]. Based on the segments, the scalar statistical and spectral features listed in Table B.1 in the Appendix are extracted.[1] Due to a large number of features (23 signals $\times$ 16 features $= 368$) and a low number of samples (392 pockets), an additional feature selection is performed using the python package *tsfresh* [Chri18]. The 10 features with the overall highest correlation to the target variable on the training set are kept, which are listed in Table B.11 in the Appendix.
As identified in Section 2.5.2, quality-related ML use cases are commonly posed as either supervised classification tasks in the context of quality prediction or unsupervised anomaly detection tasks in the context

[1] In this case study, no image-like features such as spectrograms and scalograms are considered as those did not yield satisfactory performance levels with this type of input data, consistent with earlier analysis [Fert22b]. This may be caused by the considerably lower sampling rate compared to the other case studies.

of process monitoring. Thus, both categories of ML models are analyzed. The dataset is split into training and testing data and hyperparameter optimization is done using 5-fold CV on the training data. The final hyperparameter values of all models are documented in Table B.12 in the Appendix.
The accuracy and the F_1-Scores for all models are summarized in Table 5.11.

Table 5.11.: ML model performance on the test dataset of the predictive quality and process monitoring case study.

	Supervised classification					**Anomaly detection**		
Metric	**RF**	**SVC**	***k*NN**	**DT**	**MLP**	**IF**	**LOF**	**OC-SVM**
Accuracy	0.992	0.975	0.992	0.975	0.992	0.898	0.941	0.763
F_1-Score	0.980	0.943	0.980	0.941	0.980	0.806	0.877	0.641

All supervised classification models reach high levels of performance on the test dataset. On average, the classification models achieve significantly higher performance than the anomaly detection models. This is most likely caused by an overlap of the defect types between training and testing data – as the supervised classification models are trained with examples of the NOK data, they can identify it easier when compared to anomaly detection models which are only exposed to OK data during training.

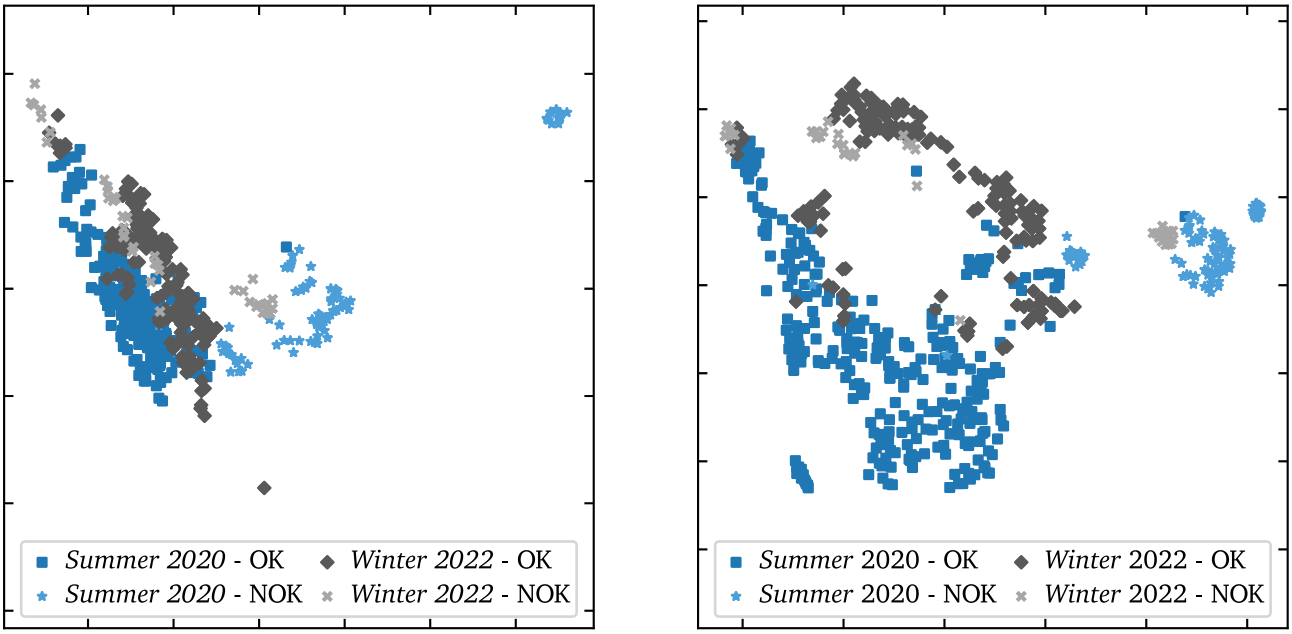
5.2.2. Data collection with concept drift

To study the influence of concept drift on this type of ML application, another 196 pockets, distributed over four plates were produced in a second run in the winter of 2022 using the same experimental design, machine and process perturbations as introduced earlier. The class imbalance is almost identical to the aforementioned dataset collected in 2022, with 79% of pockets having the CIR_2 feature within the given quality tolerances (OK). Similar to the first case study, the models trained on the training data of the first trial (summer 2020) are consequently evaluated on the testing data of the second trial (winter 2022) and the results are listed in Table 5.12.

Table 5.12.: Performance degradation of the predictive quality and process monitoring application concerning the recording dates of the data used for testing. All models are trained on the training split of the *Summer 2020* data and consequently tested on the test split of both experiment runs.

		Supervised classification					**Anomaly detection**			
Metric	**Dataset**	**RF**	**SVC**	***k*NN**	**DT**	**MLP**	**IF**	**LOF**	**OC-SVM**	**Avg.**
Accuracy	*Summer 2020*	**0.992**	**0.975**	**0.992**	**0.975**	**0.992**	**0.898**	**0.941**	**0.763**	**0.941**
	Winter 2022	0.718	0.633	0.706	0.718	0.689	0.220	0.271	0.215	0.521
F_1-Score	*Summer 2020*	**0.980**	**0.943**	**0.980**	**0.941**	**0.980**	**0.806**	**0.877**	**0.641**	**0.894**
	Winter 2022	0.324	0.270	0.316	0.324	0.304	0.317	0.287	0.322	0.308

Similar to the first case study, there is strong performance degradation in all of the ML models. Again, the performance decrease is relatively uniform across the model categories, with the anomaly detection models suffering a stronger decay than supervised classification models. The degradation of the F_1-Score is higher compared to the accuracy which is caused by the class imbalance. In contrast to the first case study, this degradation cannot be directly attributed to a change in the process parameters but rather is caused by cumulative changes, wear, the change of seasons and aging that happened in the 1.5 years that passed between the experiment runs, highlighting the vulnerability of ML models in manufacturing operations. The distribution of the feature values is visualized in Figure 5.6 – notably, both the OK and NOK classes appear shifted between the experiment runs, explaining the performance degradation.



(a) Results of 2D PCA, colored by trial and class.

(b) Results of 2D t-SNE, colored by trial and class.

Figure 5.6.: PCA and t-SNE visualizations of the experiments in the predictive quality case study. Visualizations are based on statistical and spectral features, *cf.* Table B.11 in the Appendix. One sample corresponds to the milling process data of the CIR_2 geometric feature in one pocket. Own illustration.

5.2.3. Concept drift detection

Summarizing the preceding section, the developed predictive quality and process monitoring application shows high performance under the defined operating conditions but suffers strong performance degradation in the second experiment run that is likely caused by subtle changes that happened in the 1.5 years between the trials, which would not be obvious to the machine operator and may only be noticed when the false predictions of the ML model cause problems downstream. Thus, the framework introduced in Chapter 4 is applied in this section.

Active vs passive drift detection

The presented ML application uses two categories of ML models: supervised classification, which is similar to the first case study, and anomaly detection. As anomalies in terms of NOK parts are expected to be rather sporadic and not due to permanent state changes of the production process, active drift detection is preferable, as described in Section 4.5.

Experimental evaluation

The evaluation largely follows the structure that is introduced in the first case study, *cf.* Section 5.1.3. Again, multiple scenarios are evaluated that depend on the configuration of the use case. Importantly, three major differences exist compared to the first case study.
First, the dataset is limited in the way it was recorded: The defect types are not introduced sporadically but rather one after the other. Thus, it is required to randomly shuffle the data to get representative training and testing splits, different to the first case study, in which there were separate tools used for testing. Through the random shuffling, the temporal correlation is not given anymore and it is thus not necessary to use the introduced LRDD method. Importantly, this only applies to this evaluation, in practical usage it would still be required.
Second, as mentioned in the previous section, deep learning approaches based on spectrograms and scalograms did not yield satisfactory prediction results in this case study. Thus, learned embedding-based approaches, c.2 in Table 4.1, are not evaluated here, as those approaches are only compatible with deep learning models.
Lastly, the process monitoring use case utilizes anomaly detection models that output anomaly scores, similar to the prediction confidences of supervised classification models. Thus, they are utilized in the same way.

As the remaining scenarios overlap with the scenarios of the first case study, they are briefly introduced:

- Scenario b.2: Model-independent drift detection with preprocessing – equivalent to scenario b.2 in the first case study.

- Scenario c.1.1: Model-dependent drift detection using prediction confidences – equivalent to scenario c.1 in the first case study.

- Scenario c.1.2: Model-dependent drift detection using anomaly scores – equivalent to scenario c.1 in the first case study, but using anomaly scores instead of prediction confidences.

The F_1-scores of all scenarios are reported in Table 5.13. Additionally, precision and recall of this experiment are listed within Tables B.13 and B.14 in the Appendix.

The results in Table 5.13 show that the drifted data is detectable in all three scenarios. Interestingly, the KS-based detector in scenario b.2 is performing poorly on this dataset. This may be caused by the data structure, as the signals used as input data, *cf.* Table 5.10, are likely less correlated than the signals used in the first case study, which all originated from the same sensor. The multivariate MMD-based detection method performed very well in comparison.
In scenario c.1.1, only the RF confidences allow effective drift detection, similar to the previous case study, closely followed by the MLP.
In scenario c.1.2, all anomaly detection model scores are showing very high performance in detecting the drifted operating conditions, surpassing the results of the other two scenarios in certain configurations. This highlights the utility of using model-dependent drift detection methods, especially if anomaly detection models are used.

Table 5.13.: Drift detection performance in the predictive quality and process monitoring case study. Mean F_1-scores and their standard deviations are reported over 10 experiment runs.

				$\boldsymbol{F_1}$**-Scores**	
Dataset	**Scenario**	**Detector**	$\boldsymbol{\alpha = 0.005}$	$\boldsymbol{\alpha = 0.01}$	$\boldsymbol{\alpha = 0.05}$
Winter 2022	Scenario b.2	KS	0.183 ± 0.060	0.233 ± 0.066	0.704 ± 0.032
		MMD	$\mathbf{0.971} \pm 0.013$	$\mathbf{0.991} \pm 0.004$	$\mathbf{0.978} \pm 0.008$
	Scenario c.1.1	KS(RF)	$\mathbf{0.880} \pm 0.031$	$\mathbf{0.923} \pm 0.031$	$\mathbf{0.983} \pm 0.010$
		KS(DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS(SVC)	0.182 ± 0.055	0.228 ± 0.075	0.539 ± 0.056
		KS(kNN)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS(MLP)	0.830 ± 0.021	0.861 ± 0.021	0.931 ± 0.021
	Scenario c.1.2	KS(IF)	$\mathbf{1.000} \pm 0.000$	0.986 ± 0.006	0.952 ± 0.013
		KS(OC-SVM)	$\mathbf{1.000} \pm 0.000$	$\mathbf{1.000} \pm 0.000$	$\mathbf{1.000} \pm 0.000$
		KS(LOF)	$\mathbf{1.000} \pm 0.000$	$\mathbf{1.000} \pm 0.000$	0.957 ± 0.010

5.2.4. Interim summary

This case study shows an additional implementation of the developed framework in a predictive quality and process monitoring use case. Two datasets are generated that capture the same milling process at different times and seasons. It is shown that ML models trained with data of the first experiment run have significantly degraded performance on the later experiment run. Thus, concept drift has occurred in between the experiments, which may be caused by aging equipment and the differing seasons.

An active drift detection approach is again selected and implemented in three different scenarios, showing that the drifted operating conditions can be reliably detected in all of them using the identified methodology, achieving objectives RO.a-RO.d for this case study. In addition to the scenarios that were evaluated in the first case study, it is shown that the anomaly score of anomaly detection models in the context of process monitoring is a strong method for data modeling in the respective scenario.

5.3. Case study 3: Condition monitoring in a pigment sieving process

The third case study is concerned with an accelerometer-based condition monitoring system for ultrasonic sieves within a pigment production process chain. The work presented within this case study is done in the context of a research project conducted with a company in the chemical sector. The produced pigments are used, among others, in cosmetics, food products and the automotive industry. The case study investigates the special case, identified in Section 4.4, of anomaly detection use cases with permanent anomalies. In this situation, concept drift cannot be distinguished from the permanent anomalies the ML application is designed to detect. Thus a passive approach to drift detection is implemented. Through multiple trial runs on a test machine, the magnitude of concept drift between production runs is evaluated and a suitable ML application design is derived by applying the developed framework.

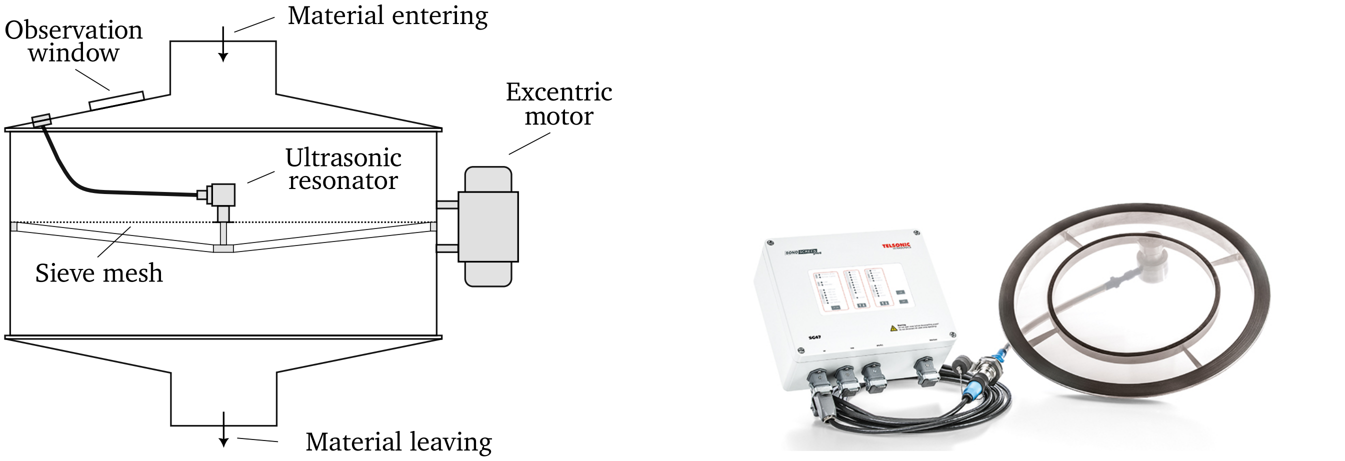
5.3.1. Use case description

The pigment production is conducted in batches of product variants which are produced in three continuous shifts over multiple days or sometimes weeks.

The pigment production process starts with raw mica as the starting material. The raw materials are stored in a barrel located on the top floor of the production building and then flow down into the coating barrel. In this stage, coloration occurs as the raw material particles receive a coating. Following this, the intermediates are pumped back up from the intermediate vessel to a filtration system. This system contains a rotating cylinder equipped with a filtering mesh, designed to remove foreign and coarse particles. The next step involves dehydrating the purified product solution in a drying facility. At this point, the product is in a powdered form and is placed on a flat conveyor belt for calcination. During this process, impurities and extraneous substances are evaporated. Additionally, for some products, thermal decomposition occurs due to the high temperatures, facilitating the formation of the final product.

The last operation in the production process is the sieving of the final product to remove impurities and pigment particles with diameters that exceed the specification. The pigment material that passes through the sieve is immediately packaged in the final packaging. That way, the sieving has a critical influence on the final product quality.

The sieving process involves sieving machines from different suppliers, *cf.* Figure 5.7 that all operate in similar ways. The sieving machines consist of metal enclosures in which a metal mesh with varying mesh sizes depending on the pigment specification is installed. The machines are excited in two ways to facilitate the material flow rate: First, an excentric electrical motor shakes the whole sieving assembly at low frequencies ($f_{\text{excentric}} \approx 100\,\mathrm{Hz}$) but high amplitudes. Second, the mesh itself is excited by an ultrasonic resonator at $f_{\text{ultrasonic}} \approx 35\,\mathrm{kHz}$.



(a) Schematic side cut-out drawing of the sieving apparatus with both sources of excitation. Own illustration.

(b) Example of sieving mesh with ultrasonic excitation source. Extracted from [Ultr24].

Figure 5.7.: Components of the sieving machine used in the pigment production case study.

A major problem in pigment production is that the sieve mesh sporadically develops tears as shown in Figure 5.8. Tears lead to particles and other impurities entering the final product that do not adhere to the specifications, affecting the product's quality.

As the tears appear highly sporadically, the currently applied maintenance strategy is corrective which oftentimes involves a significant delay. Employees regularly check the mesh surface optically through an observation window in two-hour intervals, which is a non-value-adding activity. Oftentimes, tears are not noticed for prolonged times as optical observation is inhibited by the dusty air inside the sieve. Additionally, samples of the sieved pigments are checked in the laboratory. While the laboratory method exhibits a high accuracy, it also introduces a delay of hours to days until a problem is noticed. As the batches that were sieved with tears in the mesh have to be processed again or discarded, this leads to high costs for the company. Thus, the case study's goal is to investigate how a condition monitoring system can be designed to detect starting tears early on and consequently notify employees autonomously.

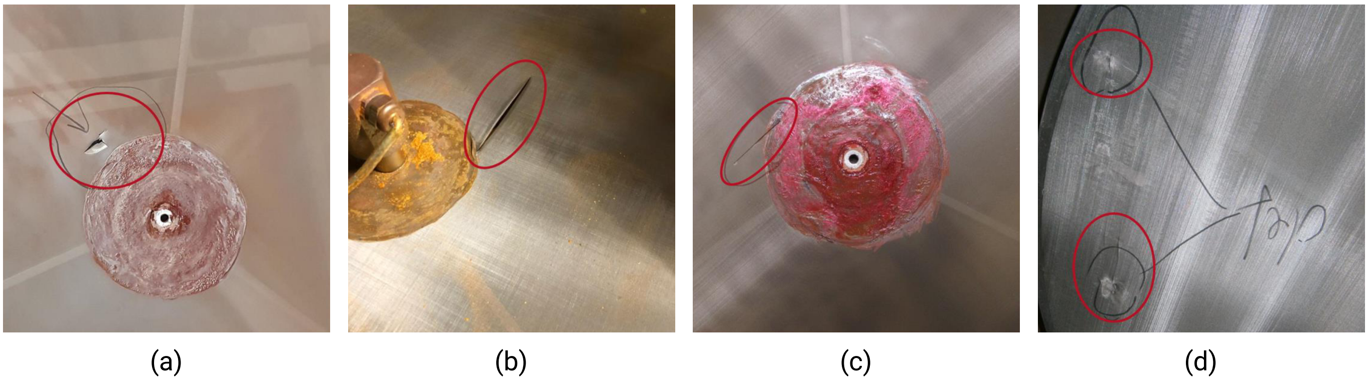


Figure 5.8.: Examples of sieve mesh tears in the pigment production case study. Own illustration.

5.3.2. Experiment setup and data exploration

While several patents exist on approaches to detect tears in industrial sieving machines, *cf.* [Zhu22], no commercial solution was found that is applicable to the requirements of the pigment production process. A master's thesis supervised by the thesis author selected anomaly detection within the sieve's vibration response to the ultrasonic excitation as the most promising approach for condition monitoring among several competing concepts [Zhu22]. A sampling rate of $f_s \geq 2 \cdot f_{\text{ultrasonic}}$ should be chosen according to the theorem of Nyquist to prevent aliasing [Lyon97] and thus properly record the vibration response of the sieve to the excitation. Consequently, an ADC and sensor are chosen that allow sampling of the acceleration at $f_s = 102.4\,\text{kHz}$. Two uniaxial accelerometers are attached to the outer hull of the lower cone of the designated test sieving machine, *cf.* Figure 5.7a for the purposes of the experiment. The properties of the accelerometers are summarized in Table 5.14.

Table 5.14.: Properties of the accelerometer used in the pigment production case study.

Sensor property	**Value/Description**
Sensor name	HBK 4397-a
Measurement axis	Z
Measurement range	± 500 g
Sensor type	IEPE
Data acquisition	NI 9250 DAQ
Sampling rate f_s	$102\,400\,\text{Hz}$
ADC resolution	24 bit

To analyze the viability of the data acquisition setup and the detectability of sieve tears through the vibration response, four experiment runs have been performed with a test sieving machine under near-production conditions. In the experiments, the sieve is left running, and after a certain time, a tear is introduced manually through a sharp object, *cf.* [Zhu22]. The STFT spectrograms of two experiments are exemplarily visualized in Figure 5.9. Visibly, there is a strong amplitude at the excitation frequency $f_{\text{ultrasonic}} \approx 35\,\text{kHz}$ and possible harmonics at approximately half the excitation frequency.

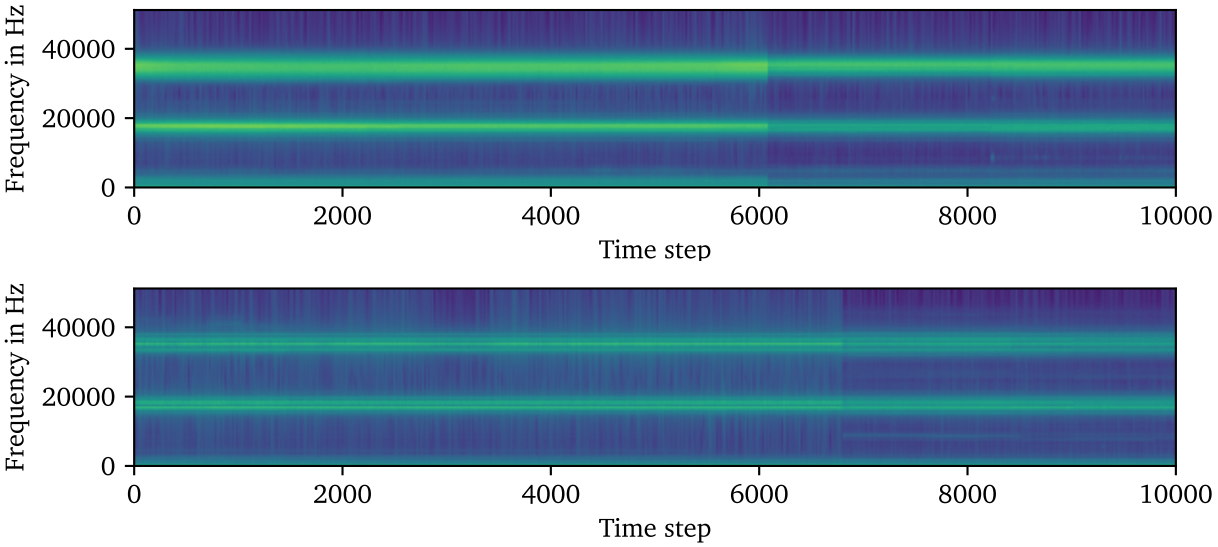
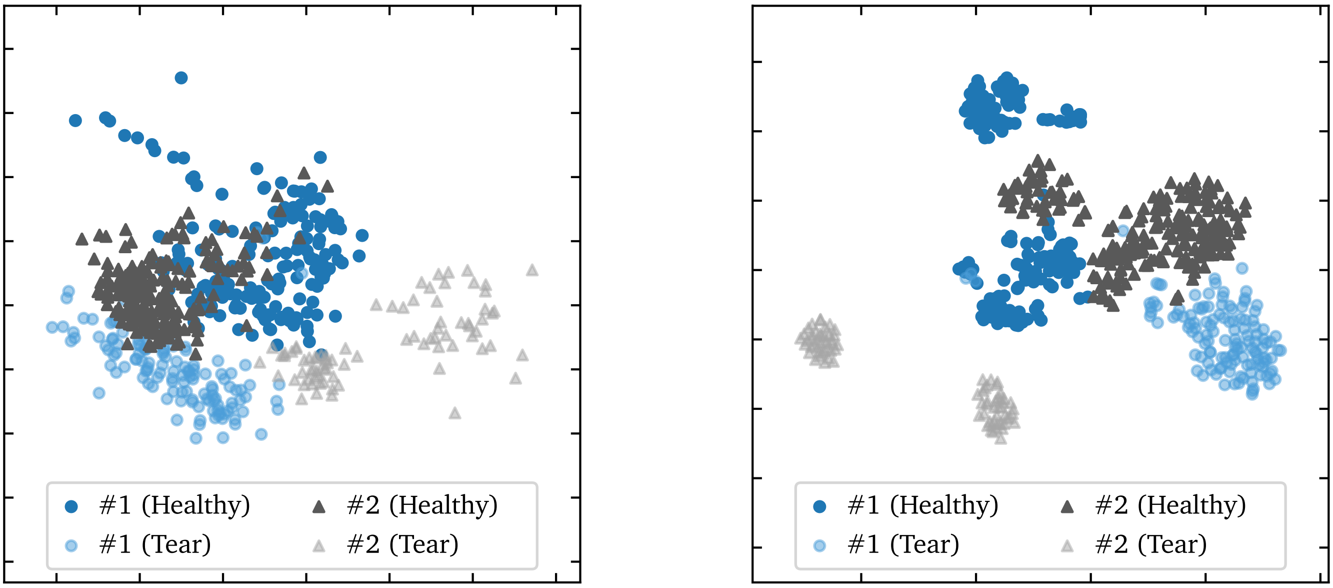


Figure 5.9.: Spectrograms of two full experiment runs in the piegment sieving case study. Brighter colors signalize higher amplitudes at the respective frequencies. The artificial tears are introduced at approximately time step 6100 in the first run and 6800 in the second run. The experiment duration is approximately $166\,\text{min}$ in each run. Only one sensor signal is visualized for each run. Own illustration.

Two important findings can be derived from the experiments and the visualization in Figure 5.9: First, the vibration response after the sieve tear is visibly different from the vibration response before the tear, indicating the general viability of the measurement concept. Secondly, the vibration response differs strongly between the two visualized experiments and also between the other two conducted experiments. This includes both, the vibration response prior to the tear as well as the vibration response after the tear. This variance can be attributed to concept drift between the experiments. The sieve meshes were exchanged in between the experiments, which involved variations in sieve mesh size and mounting position as there is no fixed reference for the orientation of the sieve, the excitation source and the respective cable connection. This variation is only expected to increase in real production as this involves the added variation of different pigment types flowing through the apparatus.

The ML application in this case study is tasked with predicting if a given data point belongs to an intact sieve surface which corresponds the negative class ($y = 0$), or a torn sieve which corresponds to the positive class ($y = 1$). In contrast to the previous two case studies, no metadata exists to semantically segment the data in time within this case study as the process is continuous. Thus, the data is divided into $2\,\text{s}$ segments for feature extraction. Based on the segments, two feature types are calculated for different kinds of ML models. First, the scalar statistical and spectral features listed in Table B.1 in the Appendix are computed for each of the two sensor signals. Additionally, spectrograms and scalograms are computed for each segment and signal as input for CNN-based models. Based on the extracted features, the data distributions of the experiments shown in Figure 5.9 are visualized as two-dimensional plots using PCA and t-SNE in Figure 5.10. This plot supports the visual findings from Figure 5.9 as it shows that both the healthy and tear classes are visibly different between the two experiment runs. Notably, the classes within the trials are separable in both the PCA and t-SNE plots. The class clusters of different trials are not overlapping though, signalizing significant concept drift in between the experiment runs.



(a) Results of 2D PCA, colored by trial and class.

(b) Results of 2D t-SNE, colored by trial and class.

Figure 5.10.: PCA and t-SNE visualizations of two experiments in the pigment production case study. Visualizations are based on statistical and spectral features, *cf.* Table B.1. One sample corresponds to a one-time window with a length of $2\,\mathrm{s}$. Own illustration.

5.3.3. Machine learning application design and concept drift detection

As the sieve tears can appear at any location on the sieve surface with various sizes and orientations, *cf.* Figure 5.8, the positive class has an expectedly large variance. Therefore, anomaly detection is preferable for this application scenario in contrast to supervised classification [Zhu22]. The introduced ML models for anomaly detection are implemented and evaluated for this case study. In addition to the three models used in the previous case study (IF, LOF and OC-SVM), a CAE is implemented that utilizes either the spectrogram (STFT) or scalogram (CWT) features as introduced in Section 2.2. The architecture of the CAE is listed in Table B.16 in the Appendix. All models are trained on the first 20% of each trial, which contains only data corresponding to a healthy sieve, and consequently tested on the remaining 80%, which contains data of the healthy and damaged sieve as visualized in Figure 5.11. The average performance of the models in classifying between healthy ($y = 0$) and torn ($y = 1$) sieve over all four experiment runs is listed in Table 5.15. The hyperparameters are optimized using CV between the experiment runs and documented in Table B.15. Importantly, each model is trained and tested on the same experiment run.

The results in Table 5.15 indicate that the anomaly detection models trained on data points of the healthy sieve are able to detect sieve tears during the respective experiment runs with high performance. The CAE model using scalograms from the CWT achieves the highest overall performance. In the following, the developed framework is applied.

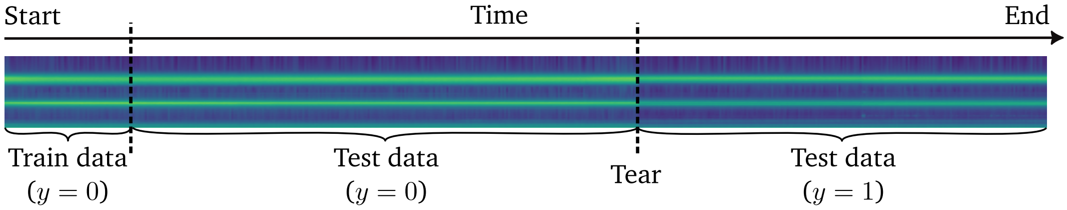


Figure 5.11.: Visualization of the data partitioning for each experiment run with respect to the results in Table 5.15. Own illustration.

Table 5.15.: Results of the model selection phase for the condition monitoring in pigment production case study. The mean and standard deviation are reported over the four experiment runs.

Model	Features	AUROC
IF	Statistical and spectral (Table B.1)	0.881 ± 0.077
LOF		0.899 ± 0.154
OC-SVM		0.847 ± 0.164
CAE	STFT	0.876 ± 0.182
CAE	**CWT**	$\mathbf{0.911} \pm 0.051$

Active vs passive drift detection

The visual evaluation of the differences between the trials has shown that there is significant concept drift in between production runs and it will therefore not be possible to train a single ML model that performs well over multiple production runs. At the same time, this case study does not allow for active concept drift detection, as the sieve tears constitute permanent anomalies and would not be distinguishable from concept drift due to other causes like sensor defects. Thus, a passive strategy is applied as described in Section 4.4. As the production runs of the pigment production line are in the scope of days and the experiments have shown that especially the changeovers between production runs introduce significant concept drift, scenario a.2 in Table 4.1 is suitable, with changeovers between production runs as the external trigger.

This concept has been implemented for production usage at the industry partner. As the ML model is trained unsupervised only on data of the healthy sieve, it can be retrained automatically once the aforementioned concept drift trigger arises. Therefore, this case study is the only case study within this thesis where the maintenance part of the *monitoring & maintenance* phase within CRISP-ML(Q) can be implemented in an automatic way. This is based on the assumption that the sieve is healthy for at least the duration of the time interval used for training the model, which is confirmed by the domain experts at the production site. At the same time, this is a major limitation, as tears that arise during model training are not detectable by this concept. The changeover times of the production runs can be identified via OPC UA signals of the MES at the production site. Once the start of a production run is identified, the implemented software starts recording the sensor data. After a predefined duration ($2\,\mathrm{h}$) is reached, a new CAE model is automatically trained on this data and deployed. The anomaly scores of this model are in turn provided via OPC UA to the MES of the machine operators and consequently visualized. The system has been passively deployed for two sieving machines at the production site of the company in late 2023. Since then, two tears have occurred on the monitored machines which have been retrospectively confirmed to be detected by the ML model.

5.3.4. Interim summary

This case study shows an implementation of the developed framework in a condition monitoring use case. An unsupervised anomaly detection model is used to detect tears in ultrasonic pigment sieves, which are permanent anomalies. Thus, they are not distinguishable from concept drift and a passive concept drift detection approach is implemented. It is shown that there is significant concept drift when the production batch is changed, due to differences in mounting positions and materials. Thus, the changeover is used as an external trigger. Through the assumption that the sieve is intact when installed, the ML model is retrained for each production batch, showing high performance on the four experiment datasets and promising first results in practical deployment, having correctly identified two real-world sieve tears during production. The system is connected to the company's MES system and currently evaluated side-by-side with the preexisting manual method of checking the sieves until the reliability is sufficiently assessed. If permanently used, the system developed within the case study will lower maintenance costs through the automatic and early detection of sieve tears, thereby reducing scrap production and the effort of manually checking the sieve condition.
For passive drift detection as applied in this case study, it is not possible to quantify the performance of the drift detection in terms of precision and recall as in the other case studies, thus RO.a and RO.b cannot be confirmed explicitly. Still, the case study shows the applicability of the framework to a use case that significantly differs from the first case studies and also does not require true labels, thus achieving RO.c and RO.d.

5.4. Summary

In this chapter, the proposed framework for concept drift detection is validated in three ML use cases, showing process monitoring, predictive quality and condition monitoring of different manufacturing processes. In all three case studies, varying levels of concept drift degrade the performance of the ML models over time due to various reasons including aging equipment, parameter variations or differences between production batches. Active and passive strategies for concept drift detection are consequently selected and implemented based on the framework. In the following, the achievements regarding the objectives defined in Section 4.1 are discussed.

In case studies 1 and 2, active concept drift detection is utilized and the performance of the drift detection is quantitatively evaluated. In both case studies, it is shown that high recall as well as a high precision is achieved. Notably, the proposed LRDD method significantly increases the precision in the first case study. Therefore, both RO.a (high recall of the drift detection) and RO.b (high precision of the drift detection) are achieved. In the third case study, passive drift detection is implemented which cannot be quantitively evaluated. However, the experiments have shown that the production batch changeovers are the strongest source of concept drift and, therefore, their usage as external triggers is justified. For a quantitative evaluation, the implemented condition monitoring application needs to be reviewed after longer terms of usage and a statistically significant number of sieve tears. Nevertheless, the case study provides an important contribution, as it shows an actual implementation and productive usage of the application at a commercial company under production conditions. As the case studies show diverse ML applications, covering all introduced target use cases in the monitoring of manufacturing use cases as well as various supervised and unsupervised ML models, RO.c (applicability to monitoring use cases in manufacturing) is achieved. Lastly, all options in the framework and, consequently, all case study implementations detect concept drift without requiring true labels during operation. Therefore, RO.d (concept drift detection without true labels) is achieved, enhancing the practical usage and applicability of the proposed framework.

6. Conclusion and outlook

In this chapter, the efforts for answering the three research questions presented in Chapter 1 are summarized and the results discussed. Based on the findings of the three case studies, recommendations regarding the practical usage of the proposed approaches are formed. Lastly, the limitations of the research in this thesis are presented along with ideas for future research.

Summary and answers to research questions

Supported by the ongoing digitization in the scope of Industry 4.0, ML applications are increasingly used within manufacturing plants. Different to laboratory experiments though, conditions in real plants will change over time, through faults, wear, operator preferences and further cumulative differences within plants, altering the data distribution that the ML model operates on. As ML applications are typically trained using a static dataset, these changes can degrade the application's performance over time, without being evident to the operators – a scenario referred to as concept drift. Motivated by this development, this thesis analyzes methods for the detection of concept drift in the context of ML applications for process and condition monitoring in the manufacturing domain, aiming to improve their reliability and acceptance.

The research in this thesis follows the DSR approach as presented in Section 4.1. In Chapters 2 and 3, the problem is identified and the research is motivated through literature reviews and expert interviews, answering the first research question regarding the impact of concept drift on ML applications in manufacturing. It is derived that the targeted condition monitoring, process monitoring and predictive quality applications within manufacturing primarily rely on a shared set of algorithms belonging to the supervised and unsupervised learning paradigms. The literature review and expert interviews indicate that concept drift is recognized as a serious issue in both practice and academia. Various ideas and methods for concept drift detection exist but, specifically for practitioners working in the domain, there is a gap in standardized frameworks for detection that provide implementation structure and highlight which methods can be applied to the targeted use cases and respective data types. Recent process models for structuring ML projects such as CRISP-ML(Q) acknowledge the need for detecting and handling concept drift but do not provide specific advice on how to do so. As a result, a significant portion of ML applications remain in a prototypic status and practitioners often resort to a *wait and see* mode, waiting for obvious signs of problems. Consequently, the proposed framework serves as a concretization of the monitoring part of the CRISP-ML(Q) process model. The scope of this thesis is limited to the detection of concept drift rather than the adaptation of the ML model, as the adaptation largely depends on the root cause. If, e.g., a sensor malfunctions, the correct adaptation would involve repairing the sensor instead of retraining the ML model.

The objectives of the concept drift detection framework, based on the identified fundamentals and use case properties, are formalized in Chapter 4, specifically, as a high performance in detecting concept drift, quantified as precision and recall (RO.a, RO.b), the applicability to the identified use case architectures and commonly used ML algorithms (RO.c), and lastly, the requirement to detect concept drift without continuous access to true labels/ground truth during operation (RO.d).

The proposed framework identifies active and passive concept drift detection as overarching categories for implementation. Active concept drift detection detects distribution shifts in various properties related to the ML model input data and predictions by comparing the distribution of a reference dataset with that of the most recent data points during operation. It is generally structured in three stages: data collection & windowing, data modeling and hypothesis testing for which implementation guidance is derived based on a systematic literature review, answering the second research question which investigates existing methods for concept drift detection in general ML literature. Distinct alternatives are found in the data modeling stage, with model-dependent and model-independent methods emerging as the major categories. Importantly, it is shown that most two-sample hypothesis tests used in active drift detection rely on the i.i.d. assumption, which is commonly violated in the targeted use cases due to the correlation of data points over time. Thus, LRDD is proposed as a method to perform a preselection of the reference dataset, significantly reducing the number of false positive drift detections.
In contrast to active concept drift detection, passive concept drift detection relies on predefined time intervals or external triggers such as the changeover between production batches as signals for potential concept drift. Active concept drift detection or a combination of both active and passive are generally preferable for most use cases in comparison to only passive concept drift detection, as passive concept drift detection is by definition vulnerable to false alarms and missed detections. In summary, the provided framework answers the third research question by guiding how to implement concept drift detection depending on the use case configuration.

The framework is validated under near-production conditions in three diverse case studies within Chapter 5. The case studies show condition monitoring, process monitoring as well as predictive quality use cases in CNC milling and chemical production processes. It is investigated how concept drift influences the performance of the ML applications and if it can reliably be detected using the proposed methods.
In the first case study concerning accelerometer-based tool condition monitoring in CNC milling, multiple datasets are generated that change specific parameters or aspects of the milling process to introduce concept drift. It is shown that the performance of the tool condition monitoring application significantly degrades for parts of the introduced changes while it stays rather constant for others. Three scenarios for concept drift detection are considered, which rely on different quantities in the data modeling stage of the framework. The experiments show that drift can be reliably detected in all scenarios. Furthermore, the proposed LRDD method is validated and it is shown that the precision of the drift detection is significantly increased.
In the second case study, two datasets for process monitoring and predictive quality in CNC milling based on machine control-internal signals are analyzed which have been recorded during different seasons and 1.5 years apart. It is shown that the performance of the application is significantly degraded in the second dataset. Again, multiple scenarios are investigated, utilizing the proposed framework and showing that the drift can be detected reliably in all of them. Importantly, the second case study additionally shows that anomaly scores from anomaly detection models can be used for concept drift detection in place of prediction confidences of supervised models, which is especially relevant for process monitoring use cases where unsupervised models are often employed.
The last case study is concerned with accelerometer-based condition monitoring of sieves in pigment production and is developed at the site of a chemical company. In this use case, anomalies are permanent and thus passive concept drift detection is used with changeovers between production batches as the trigger. In this case study, automatic retraining is implemented and the system showed promising results after being integrated into the MES system of the company.

Overall, this thesis provides solution approaches and exemplary implementations to deal with concept drift in the manufacturing domain. The experiments show that concept drift is an important issue in all considered use cases that needs to be addressed appropriately to keep ML applications in the manufacturing domain usable and reliable over longer terms of usage. The thesis thereby aims to increase awareness among researchers and practitioners about the issue of evolving input data over time and its effects on deployed ML models. The proposed concept drift detection methods in the framework can be applied to both, existing ML applications, lacking the implementation of the respective CRISP-ML(Q) phase, as well as to new projects in the manufacturing domain as the requirements consider various configurations of the targeted use cases. Ultimately, the author expects that better management of concept drift will not only help maintain the accuracy of deployed ML models over time but also build more trust and acceptance for ML-based applications in manufacturing. As a result, this work can help to increase the overall impact of ML in this domain.

Study limitations and future work

Besides the scientific and practical contributions, this study faces limitations that need to be considered when interpreting the findings. Two major areas of limitations are described within this section.

Drift types In the first and second case study, methods for active drift detection are validated with machine and process configurations that differ from the initial training configuration or have significantly aged, thus introducing abrupt concept drift. While the evaluation methodology allows for the assumption that the drift detection approaches will also work with slow, gradual or incremental drifts in the data, this could not be explicitly tested due to the constraints of the case study design. Further studies, especially involving long-term data recordings spanning multiple months or years of production are needed to validate this scenario conclusively. Furthermore, all presented methods for active drift detection except for explicit monitoring of performance KPIs are unable to detect concept drift that does not influence $P(X)$ *cf.* Section 2.3.1, as this type of drift does not alter the input data distribution. Overall, this type of drift was not practically observed in any of the case studies or the relevant literature and thus seems rather exotic in real-world applications.

Case studies and data types While the case studies within this thesis cover application scenarios that are common in industry and research, other application architectures and data types exist that were not analyzed. Importantly, the case studies in this thesis do not involve image data, which is becoming increasingly relevant in QC applications. While some of the proposed solutions such as model confidences may be transferable without major changes, further studies should be conducted in this direction. Furthermore, only classification and anomaly detection use cases have been analyzed. While this configuration is prevalent within the described use cases, *cf* Section 2.5, regression models can be useful in certain scenarios such as directly predicting equipment RUL or quality-relevant parameters or dimensions and should thus be considered in future work.

In addition to future work that addresses the aforementioned limitations of this study, several other directions are suitable for further scientific exploration based on the results.

Exploitation of correlation properties The proposed LRDD method for enabling concept drift detection in non-i.i.d. scenarios is only a starting point in this direction. It should be investigated whether explicit testing methodologies can be defined that account for the temporal correlation of the data points within the detection windows and thus enhance potential weak points of LRDD such as its insensitivity to drifts in data point density.

Efficient model updates and robustness As this thesis is focussed on the detection of concept drift, less emphasis is put on the adaptation of models to new data distributions once a drift has occurred. As stated in Chapters 3 and 4, automatic adaption of models in case of detected drift is not always desirable as the drift might stem from external changes such as defective sensors that should be repaired instead of blindly adapting the ML model. Concepts such as continual learning as well as domain adaptation and finetuning from the domain of transfer learning can be explored and, in instances where it makes sense, combined with concept drift detection to minimize the computational costs as well as labeling efforts when adapting models to a changed environment. Lastly, with the recent success of foundation models in the fields of computer vision and natural language processing [Bomm21], their application to the manufacturing use cases within this thesis should be evaluated in the future. Models that provide zero-shot or few-shot capabilities would require less effort in updating, e.g., [Tnan22b], or might even generalize robustly across domain shifts.

Bibliography

Own related publications

[Elle23] S. Ellenrieder, N. Jourdan, T. Biegel, B. Bretones Cassoli, J. Metternich, and P. Buxmann. “Towards the Sustainable Development of Machine Learning Applications in Industry 4.0”. In: *Proceedings of the European Conference on Information Systems*. 2023.

[Jour23a] N. Jourdan, T. Bayer, T. Biegel, and J. Metternich. “Handling concept drift in deep learning applications for process monitoring”. In: *Procedia CIRP* 120. 2023.

[Jour21a] N. Jourdan, T. Biegel, V. Knauthe, M. von Buelow, S. Guthe, and J. Metternich. “A computer vision system for saw blade condition monitoring”. In: *Procedia CIRP* 104. 2021.

[Jour21b] N. Jourdan, L. Longard, T. Biegel, and J. Metternich. “Machine learning for intelligent maintenance and quality control: A review of existing datasets and corresponding use cases”. In: *Proceedings of the Conference on Production Systems and Logistics*. 2021.

[Jour23b] N. Jourdan and J. Metternich. “A Nearest Neighbor-Based Concept Drift Detection Strategy for Reliable Tool Condition Monitoring”. In: *Proceedings of the 3rd International Workshop on Software Engineering and AI for Data Quality in Cyber-Physical Systems/Internet of Things*. 2023.

[Jour23c] N. Jourdan and J. Metternich. “A Nearest Neighbor-Based Concept Drift Detection Strategy for Reliable Tool Condition Monitoring”. In: *NeurIPS Workshop on Distribution Shifts*. 2023.

[Jour20] N. Jourdan, E. Rehder, and U. Franke. “Identification of uncertainty in artificial neural networks”. In: *Proceedings of the 13th Uni-DAS eV Workshop Fahrerassistenz und automatisiertes Fahren*. 2020.

[Jour21c] N. Jourdan, S. Sen, E. J. Husom, E. Garcia-Ceja, T. Biegel, and J. Metternich. “On The Reliability Of Machine Learning Applications In Manufacturing Environments”. In: *NeurIPS Workshop on Distribution Shifts*. 2021.

[Mett21] J. Metternich, T. Biegel, B. B. Cassoli, F. Hoffmann, N. Jourdan, J. Rosemeyer, P. Stanula, and A. Ziegenbein. *Künstliche Intelligenz zur Umsetzung von Industrie 4.0 im Mittelstand: Expertise des Forschungsbeirats der Plattform Industrie 4.0*. Deutsche Akademie der Technikwissenschaften (Acatech). 2021.

[Wint23b] A. Winter, N. Jourdan, T. Wirth, V. Knauthe, and A. Kuijper. “An Empirical Study of Uncertainty Estimation Techniques for Detecting Drift in Data Streams”. In: *NeurIPS 2023 Workshop on Distribution Shifts*. 2023.

Articles and books

[Ab G20] N. L. Ab Ghani, I. A. Aziz, and M. Mehat. “Concept drift detection on unlabeled data streams: A systematic literature review”. In: *IEEE conference on big data and analytics (ICBDA)*. IEEE. 2020.

[Abde14] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. “Convolutional neural networks for speech recognition”. In: *IEEE/ACM Transactions on audio, speech, and language processing* 22. 2014.

[Abde18] O. Abdeljaber, S. Sassi, O. Avci, S. Kiranyaz, A. A. Ibrahim, and M. Gabbouj. “Fault detection and severity identification of ball bearings by online condition monitoring”. In: *IEEE Transactions on Industrial Electronics* 66. 2018.

[Acke20a] S. Ackerman, P. Dube, and E. Farchi. “Sequential drift detection in deep learning classifiers”. In: *arXiv preprint arXiv:2007.16109*. 2020.

[Acke21a] S. Ackerman, P. Dube, E. Farchi, O. Raz, and M. Zalmanovici. “Machine learning model drift detection via weak data slices”. In: *IEEE/ACM Third International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)*. IEEE. 2021.

[Acke20b] S. Ackerman, E. Farchi, O. Raz, M. Zalmanovici, and P. Dube. “Detection of data drift and outliers affecting machine learning model performance over time”. In: *arXiv preprint arXiv:2012.09258*. 2020.

[Acke21b] S. Ackerman, O. Raz, M. Zalmanovici, and A. Zlotnick. “Automatically detecting data drift in machine learning classifiers”. In: *arXiv preprint arXiv:2111.05672*. 2021.

[Agga17] C. C. Aggarwal and C. C. Aggarwal. *An introduction to outlier analysis*. Springer, 2017.

[Ahma20] M. I. Ahmad, Y. Yusof, M. E. Daud, K. Latiff, A. Z. A. Kadir, and Y. Saif. “Machine monitoring system: a decade in review”. In: *The International Journal of Advanced Manufacturing Technology*. 2020.

[Ambe23] Amberscript. *Amberscript: Speech Recognition and Audio to Text Transcription Services*. *https://www.amberscript.com*. Accessed: 2024-02-01. 2023.

[Ambh15] N. Ambhore, D. Kamble, S. Chinchanikar, and V. Wayal. “Tool condition monitoring system: A review”. In: *Materials Today: Proceedings* 2. 2015.

[Amin18] M. Amini and S. Chang. “A review of machine learning approaches for high dimensional process monitoring”. In: *Proceedings of the 2018 Industrial and Systems Engineering Research Conference*. 2018.

[Bach21] F. Bachinger, G. Kronberger, and M. Affenzeller. “Continuous improvement and adaptation of predictive models in smart manufacturing and model management”. In: *IET Collaborative Intelligent Manufacturing* 3. 2021.

[Baen06] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno. “Early drift detection method”. In: *Fourth international workshop on knowledge discovery from data streams*. Vol. 6. 2006.

[Baie21] L. Baier. “Concept Drift Handling in Information Systems: Preserving the Validity of Deployed Machine Learning Models”. PhD thesis. 2021.

[Baie23] L. Baier, T. Schlör, J. Schöffer, and N. Kühl. “Detecting Concept Drift With Neural Network Model Uncertainty”. In: *Proceedings of the 56th Hawaii International Conference on System Sciences*. 2023.

[Banf22] M. Banf and G. Steinhagen. “Who supervises the supervisor? Model monitoring in production using deep feature embeddings with applications to workpiece inspection”. In: *arXiv preprint arXiv:2201.06599*. 2022.

[Bena03] P. Benardos and G.-C. Vosniakos. “Predicting surface roughness in machining: a review”. In: *International journal of machine tools and manufacture* 43. 2003.

[Bert21] M. Bertolini, D. Mezzogori, M. Neroni, and F. Zammori. “Machine Learning for industrial applications: A comprehensive literature review”. In: *Expert Systems with Applications* 175. 2021.

[Bieg23] T. Biegel, P. Helm, N. Jourdan, and J. Metternich. “SSMSPC: self-supervised multivariate statistical in-process control in discrete manufacturing processes”. In: *Journal of Intelligent Manufacturing*. 2023.

[Bieg22] T. Biegel, N. Jourdan, C. Hernandez, A. Cviko, and J. Metternich. “Deep learning for multivariate statistical in-process control in discrete manufacturing: A case study in a sheet metal forming process”. In: *Procedia CIRP* 107. 2022.

[Bife07] A. Bifet and R. Gavalda. “Learning from time-changing data with adaptive windowing”. In: *Proceedings of the 2007 SIAM international conference on data mining*. SIAM. 2007.

[Bish23] C. M. Bishop and H. Bishop. “Deep Learning: Foundations and Concepts”. In: *Deep Learning: Foundations and Concepts*. Springer, 2023.

[Bish06] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. Springer, 2006.

[Blan95] J. M. Bland and D. G. Altman. “Multiple significance tests: the Bonferroni method”. In: *British Medical Journal* 310. 1995.

[Bomm21] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. “On the opportunities and risks of foundation models”. In: *arXiv preprint arXiv:2108.07258*. 2021.

[Brec21] C. Brecher and M. Weck. *Werkzeugmaschinen Fertigungssysteme 3: Mechatronische Systeme, Steuerungstechnik und Automatisierung*. Springer, 2021.

[Brem15] P. Bremer and H. Brüggemann. *Grundlagen Qualitätsmanagement: Von den Werkzeugen über Methoden zum TQM*. Vieweg+ Teubner, 2015.

[Bret23] B. Bretones Cassoli and J. Metternich. “Challenges for Predictive Quality in Multi-stage Manufacturing: Insights from Literature Review”. In: *Proceedings of the 3rd International Workshop on Software Engineering and AI for Data Quality in Cyber-Physical Systems/Internet of Things*. 2023.

[Breu00] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. “LOF: identifying density-based local outliers”. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000.

[Cai20] W. Cai, W. Zhang, X. Hu, and Y. Liu. “A hybrid information model based on long short-term memory network for tool condition monitoring”. In: *Journal of Intelligent Manufacturing* 31. 2020.

[Cass23] B. B. Cassoli, N. Jourdan, and J. Metternich. “Multi-source data modelling and graph neural networks for predictive quality”. In: *Procedia CIRP* 120. 2023, pp. 39–44.

[Cass22] B. B. Cassoli, N. Jourdan, P. H. Nguyen, S. Sen, E. Garcia-Ceja, and J. Metternich. “Frameworks for data-driven quality management in cyber-physical systems for manufacturing: A systematic review”. In: *Procedia CIRP* 112. 2022, pp. 567–572.

[Cast21] A. Castellani, S. Schmitt, and B. Hammer. “Task-sensitive concept drift detector with constraint embedding”. In: *IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2021.

[Cava16] R. C. Cavalcante, L. L. Minku, and A. L. Oliveira. “Fedd: Feature extraction for explicit concept drift detection in time series”. In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2016.

[Chan09] V. Chandola, A. Banerjee, and V. Kumar. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41. 2009.

[Chat18] C. Chatfield. *Statistics for technology: a course in applied statistics*. Routledge, 2018.

[Chic20] D. Chicco and G. Jurman. “The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation”. In: *BMC genomics* 21. 2020.

[Chri18] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr. “Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)”. In: *Neurocomputing* 307. 2018.

[Chui21] M. Chui, B. Hall, H. Mayhew, A. Singla, and A. Sukharevsky. *The State of AI in 2021*. McKinsey & Company. 2021.

[Chui22] M. Chui, B. Hall, A. Singla, and A. Sukharevsky. *The State of AI in 2022 – and a half decade in review*. McKinsey & Company. 2022.

[Chui23] M. Chui, M. Issler, R. Roberts, and L. Yee. *McKinsey Technology Trends Outlook 2023*. McKinsey & Company. 2023.

[Cobb22] O. Cobb and A. Van Looveren. “Context-aware drift detection”. In: *International Conference on Machine Learning*. PMLR. 2022.

[Cobb23] O. Cobb and A. Van Looveren. “Towards Practicable Sequential Shift Detectors”. In: *arXiv preprint arXiv:2307.14758*. 2023.

[Corr20] J. C. A. J. Correa and A. A. L. Guzman. *Mechanical vibrations and condition monitoring*. Academic Press, 2020.

[Cost18] A. F. J. Costa, R. A. S. Albuquerque, and E. M. Dos Santos. “A drift detection method based on active learning”. In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018.

[Cost17] F. G. da Costa, F. S. Duarte, R. M. Vallim, and R. F. de Mello. “Multidimensional surrogate stability to detect data stream concept drift”. In: *Expert Systems with Applications* 87. 2017.

[Cost16] F. G. da Costa, R. A. Rios, and R. F. de Mello. “Using dynamical systems tools to detect concept drift in data streams”. In: *Expert Systems with Applications* 60. 2016.

[Dan90] L. Dan and J. Mathew. “Tool wear and failure monitoring techniques for turning—a review”. In: *International Journal of Machine Tools and Manufacture* 30. 1990.

[Delg11] M. Delgado, A. Garcia, and J. Ortega. “Evaluation of feature calculation methods for electromechanical system diagnosis”. In: *8th IEEE Symposium on Diagnostics for Electrical Machines, Power Electronics & Drives*. IEEE. 2011.

[Delo20] Deloitte. *AI Enablement on the Way to Smart Manufacturing*. Deloitte. 2020.

[Diet00] T. G. Dietterich. “Ensemble methods in machine learning”. In: *International workshop on multiple classifier systems*. Springer. 2000.

[Ditz11] G. Ditzler and R. Polikar. “Hellinger distance based drift detection for nonstationary environments”. In: *2011 IEEE symposium on computational intelligence in dynamic and uncertain environments (CIDUE)*. IEEE. 2011.

[Ditz15] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. “Learning in nonstationary environments: A survey”. In: *IEEE Computational Intelligence Magazine* 10. 2015.

[Djur22] A. Djurisic, N. Bozanic, A. Ashok, and R. Liu. “Extremely Simple Activation Shaping for Out-of-Distribution Detection”. In: *arXiv preprint arXiv:2209.09858*. 2022.

[Dos 16] D. M. Dos Reis, P. Flach, S. Matwin, and G. Batista. “Fast unsupervised online drift detection using incremental kolmogorov-smirnov test”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016.

[Dumo16] V. Dumoulin and F. Visin. “A guide to convolution arithmetic for deep learning”. In: *arXiv preprint arXiv:1603.07285*. 2016.

[Eck22] B. Eck, D. Kabakci-Zorlu, Y. Chen, F. Savard, and X. Bao. “A monitoring framework for deployed machine learning models with supply chain examples”. In: *IEEE International Conference on Big Data (Big Data)*. IEEE. 2022.

[Esco18] T. Escovedo, A. Koshiyama, A. A. da Cruz, and M. Vellasco. “DetectA: abrupt concept drift detection in non-stationary environments”. In: *Applied Soft Computing* 62. 2018.

[Esco15] T. Escovedo, A. Koshiyama, M. Vellasco, R. Melo, and A. A. da Cruz. “A2D2: A pre-event abrupt drift detection”. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2015.

[Fari20] A. de Farias, S. L. R. de Almeida, S. Delijaicov, V. Seriacopi, and E. C. Bordinassi. “Simple machine learning allied with data-driven methods for monitoring tool wear in machining processes”. In: *The International Journal of Advanced Manufacturing Technology* 109. 2020.

[Fawc06] T. Fawcett. “An introduction to ROC analysis”. In: *Pattern recognition letters* 27. 2006. An introduction to ROC analysis.

[Fert22a] A. Fertig, M. Weigold, and Y. Chen. “Machine Learning based quality prediction for milling processes using internal machine tool data”. In: *Advances in Industrial and Manufacturing Engineering* 4. 2022.

[Fert23] A. Fertig. “Datenbasierte, prozessparallele Qualitätsprognose für spanend hergestellte Werkstücke mittels maschinellen Lernens”. PhD thesis. 2023.

[Fert22b] A. Fertig, C. Preis, and M. Weigold. “Quality prediction for milling processes: automated parametrization of an end-to-end machine learning pipeline”. In: *Production Engineering*. 2022.

[Fink20] O. Fink, Q. Wang, M. Svensen, P. Dersin, W.-J. Lee, and M. Ducoffe. “Potential, challenges and future directions for deep learning in prognostics and health management applications”. In: *Engineering Applications of Artificial Intelligence* 92. 2020.

[Flic04] U. Flick. “Triangulation in qualitative research”. In: *A companion to qualitative research* 3. 2004.

[Gal16] Y. Gal and Z. Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *International conference on machine learning*. PMLR. 2016.

[Gama04] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. “Learning with drift detection”. In: *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence*. Springer. 2004.

[Gama09] J. Gama, R. Sebastiao, and P. P. Rodrigues. “Issues in evaluation of stream learning algorithms”. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2009.

[Gama14] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. “A survey on concept drift adaptation”. In: *ACM computing surveys (CSUR)* 46. 2014.

[Gama17] S. Gamage and U. Premaratne. “Detecting and adapting to concept drift in continually evolving stochastic processes”. In: *Proceedings of the International Conference on Big Data and Internet of Thing*. 2017.

[Gao11] R. X. Gao, R. Yan, R. X. Gao, and R. Yan. “Continuous wavelet transform”. In: *Wavelets: Theory and Applications for Manufacturing*. 2011.

[Garc22] G. R. Garcia, G. Michau, M. Ducoffe, J. S. Gupta, and O. Fink. “Temporal signals to images: Monitoring the condition of industrial assets with deep learning image processing algorithms”. In: *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* 236. 2022.

[Gonç14] P. M. Gonçalves Jr, S. G. de Carvalho Santos, R. S. Barros, and D. C. Vieira. “A comparative study on concept drift detectors”. In: *Expert Systems with Applications* 41. 2014.

[Good16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[Gori22] V. Gori, G. Veneri, and V. Ballarini. “Continual Learning for anomaly detection on turbomachinery prototypes-A real application”. In: *IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2022.

[Gözü19] Ö. Gözüaçık, A. Büyükçakır, H. Bonab, and F. Can. “Unsupervised concept drift detection with a discriminative classifier”. In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019.

[Grec21] S. Greco and T. Cerquitelli. “Drift lens: Real-time unsupervised concept drift detection by evaluating per-label embedding distributions”. In: *International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2021.

[Greg13] S. Gregor and A. R. Hevner. “Positioning and presenting design science research for maximum impact”. In: *MIS quarterly*. 2013.

[Gret12] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. “A kernel two-sample test”. In: *The Journal of Machine Learning Research* 13. 2012.

[Grim99] H. T. Grimmelius, P. P. Meiler, H. L. Maas, B. Bonnier, J. S. Grevink, and R. F. van Kuilenburg. “Three state-of-the-art methods for condition monitoring”. In: *IEEE Transactions on Industrial Electronics* 46. 1999.

[Han22] M. Han, Z. Chen, M. Li, H. Wu, and X. Zhang. “A survey of active and passive concept drift handling methods”. In: *Computational Intelligence* 38.4. 2022.

[Hast09] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[Hati19] B. Hatiboglu, S. Schuler, A. Bildstein, and M. Hämmerle. *Einsatzfelder von künstlicher Intelligenz im Produktionsumfeld*. Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA, Stuttgart. 2019.

[Hess19] D. F. Hesser and B. Markert. “Tool wear monitoring of a retrofitted CNC milling machine using artificial neural networks”. In: *Manufacturing letters* 19. 2019.

[Heus20] M. Heusinger and F.-M. Schleif. “Reactive concept drift detection using coresets over sliding windows”. In: *IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2020.

[Hevn04] A. R. Hevner, S. T. March, J. Park, and S. Ram. “Design science in information systems research”. In: *MIS quarterly*. 2004.

[Hind21] F. Hinder, J. Brinkrolf, V. Vaquet, and B. Hammer. “A shape-based method for concept drift detection and signal denoising”. In: *IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2021.

[Huss21] S. S. Hussain, M. Hashmani, V. Uddin, T. Ansari, and M. Jameel. “A novel approach to detect concept drift using machine learning”. In: *International Conference on Computer & Information Sciences (ICCOINS)*. IEEE. 2021.

[Huye22] C. Huyen. *Designing machine learning systems*. O’Reilly Media, 2022.

[Ishi90] K. Ishikawa and J. H. Loftus. *Introduction to quality control*. Vol. 98. Springer, 1990.

[Jawo17] M. Jaworski, P. Duda, and L. Rutkowski. “On applying the restricted Boltzmann machine to active concept drift detection”. In: *IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2017.

[Jayn57] E. T. Jaynes. “Information theory and statistical mechanics”. In: *Physical review* 106. 1957. Information theory and statistical mechanics.

[Jiao23] C. Jiao, M. Fengjian, L. Zuohong, and T. Jianhua. “EdgeFD: An Edge-Friendly Drift-Aware Fault Diagnosis System for Industrial IoT”. In: *arXiv preprint arXiv:2310.04704*. 2023.

[John21] M. M. John, H. H. Olsson, and J. Bosch. “Towards mlops: A framework and maturity model”. In: *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2021.

[Jöhn21] J. Jöhnk, M. Weißert, and K. Wyrtki. “Ready or not, AI comes— an interview study of organizational AI readiness factors”. In: *Business & information systems engineering* 63. 2021.

[Joll02] I. T. Jolliffe. *Principal component analysis for special types of data*. Springer, 2002.

[Jord15] M. I. Jordan and T. M. Mitchell. “Machine learning: Trends, perspectives, and prospects”. In: *Science* 349.6245. 2015.

[Kahr22] A. Kahraman, M. Kantardzic, and M. Kotan. “Dynamic Modeling With Integrated Concept Drift Detection for Predicting Real-Time Energy Consumption of Industrial Machines”. In: *IEEE Access* 10. 2022.

[Kami22] D. Kaminskyi, B. Li, and E. Müller. “Reconstruction-based unsupervised drift detection over multivariate streaming data”. In: *IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2022.

[Kank11] P. K. Kankar, S. C. Sharma, and S. P. Harsha. “Fault diagnosis of ball bearings using machine learning methods”. In: *Expert Systems with applications* 38. 2011.

[Kerm23] R. Kermenov, G. Nabissi, S. Longhi, and A. Bonci. “Anomaly Detection and Concept Drift Adaptation for Dynamic Systems: A General Method with Practical Implementation Using an Industrial Collaborative Robot”. In: *Sensors* 23. 2023.

[Kim16] Y. I. Kim and C. H. Park. “Concept drift detection on streaming data under limited labeling”. In: *IEEE International Conference on Computer and Information Technology (CIT)*. IEEE. 2016.

[King21] H. Kingetsu and K. Kobayashi. “Born-again decision boundary: Unsupervised concept drift detection by inspector neural network”. In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021.

[King14] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*. 2014.

[Kitc07] B. Kitchenham, S. Charters, et al. *Guidelines for performing systematic literature reviews in software engineering*. 2007.

[Kory19] L. Korycki and B. Krawczyk. “Unsupervised drift detector ensembles for data stream mining”. In: *IEEE international conference on data science and advanced analytics (DSAA)*. IEEE. 2019.

[Kory21] Ł. Korycki and B. Krawczyk. “Concept drift detection from multi-class imbalanced data streams”. In: *IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE. 2021.

[Krau20] J. Krauß, B. M. Pacheco, H. M. Zang, and R. H. Schmitt. “Automated machine learning for predictive quality in production”. In: *Procedia CIRP* 93. 2020.

[Kraw18] B. Krawczyk, B. Pfahringer, and M. Woźniak. “Combining active learning with concept drift detection for data stream mining”. In: *IEEE international conference on big data (big data)*. IEEE. 2018.

[Kriz12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. Imagenet classification with deep convolutional. 2012.

[Kühl20] N. Kühl, M. Goutier, R. Hirt, and G. Satzger. “Machine learning in artificial intelligence: Towards a common understanding”. In: *arXiv preprint arXiv:2004.04686*. 2020.

[Kühl21] N. Kühl, R. Hirt, L. Baier, B. Schmitz, and G. Satzger. “How to Conduct Rigorous Supervised Machine Learning in Information Systems Research: The Supervised Machine Learning Report Card”. In: *Communications of the Association for Information Systems* 48.1. 2021.

[Kurg06] L. A. Kurgan and P. Musilek. “A survey of knowledge discovery and data mining process models”. In: *The Knowledge Engineering Review* 21. 2006.

[Kusi17] A. Kusiak. “Smart manufacturing must embrace big data”. In: *Nature* 544. 2017.

[Kvak23] D. Kvaktun, D. Liu, and R. Schiffers. “Detection of concept drift for quality prediction and process control in injection molding”. In: *AIP Conference Proceedings*. Vol. 2884. 1. AIP Publishing. 2023.

[Laks17] B. Lakshminarayanan, A. Pritzel, and C. Blundell. “Simple and scalable predictive uncertainty estimation using deep ensembles”. In: *Advances in neural information processing systems* 30. 2017.

[LeCu15] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *Nature* 521. 2015.

[LeCu98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86. 1998.

[LeCu10] Y. LeCun, C. Cortes, and C. Burges. *MNIST handwritten digit database*. ATT Labs. 2010.

[Lee12] J. Lee and F. Magoules. “Detection of concept drift for learning from stream data”. In: *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*. IEEE. 2012.

[Lehm86] E. L. Lehmann, J. P. Romano, and G. Casella. *Testing statistical hypotheses*. Vol. 3. Springer, 1986.

[LeNa19] A. LeNail. “NN-SVG: Publication-Ready Neural Network Architecture Schematics.” In: *J. Open Source Softw.* 4.33. 2019.

[Lewe24] Lewetz. *WinPC-NC Übersicht*. *https://www.lewetz.de/de/sample-sites-2/winpc-nc/uebersicht*. Accessed: 2024-01-25. 2024.

[Lewi22] G. A. Lewis, S. Echeverría, L. Pons, and J. Chrabaszcz. “Augur: A step towards realistic drift detection in production ml systems”. In: *Proceedings of the 1st Workshop on Software Engineering for Responsible AI*. 2022.

[Lin19] C.-C. Lin, D.-J. Deng, C.-H. Kuo, and L. Chen. “Concept drift detection and adaption in big imbalance industrial IoT data using an ensemble learning method of offline classifiers”. In: *IEEE Access* 7. 2019.

[Lind13] P. Lindstrom, B. Mac Namee, and S. J. Delany. “Drift detection using uncertainty distribution divergence”. In: *Evolving Systems* 4. 2013.

[Lipt18] Z. Lipton, Y.-X. Wang, and A. Smola. “Detecting and correcting for label shift with black box predictors”. In: *International conference on machine learning*. PMLR. 2018.

[Liu18] A. Liu, J. Lu, F. Liu, and G. Zhang. “Accumulating regional density dissimilarity for concept drift detection in data streams”. In: *Pattern Recognition* 76. 2018.

[Liu20] A. Liu, J. Lu, and G. Zhang. “Concept drift detection via equal intensity k-means space partitioning”. In: *IEEE transactions on cybernetics* 51. 2020.

[Liu08] F. T. Liu, K. M. Ting, and Z.-H. Zhou. “Isolation forest”. In: *8th IEEE International Conference on Data Mining*. IEEE. 2008.

[Liu17] S. Liu, L. Feng, J. Wu, G. Hou, and G. Han. “Concept drift detection for data stream learning based on angle optimized global embedding and principal component analysis in sensor networks”. In: *Computers & Electrical Engineering* 58. 2017.

[Long15] J. Long, E. Shelhamer, and T. Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

[Lour22] A. Lourenço, M. Fernandes, G. Marreiros, and J. M. Corchado. “Using simulation to evaluate a concept drift detector for condition based maintenance”. In: *IECON 2022–48th Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2022.

[Lu18] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. “Learning under concept drift: A review”. In: *IEEE transactions on knowledge and data engineering* 31. 2018.

[Lugh15] E. Lughofer, E. Weigl, W. Heidl, C. Eitzinger, and T. Radauer. “Drift detection in data stream classification without fully labelled instances”. In: *IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS)*. IEEE. 2015.

[Lyon97] R. G. Lyons. *Understanding digital signal processing*. Pearson Education India, 1997.

[Madd19] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson. “A simple baseline for bayesian uncertainty in deep learning”. In: *Advances in Neural Information Processing Systems* 32. 2019.

[Manu23] Manufacturing Leadership Council. *The Future of AI in Manufacturing*. Manufacturing Leadership Council Report. Accessed: 2024-03-07. 2023. URL: *https://www.manufacturingleadershipcouncil.com*.

[Masc23] B. Maschler. “A Survey on Deep Industrial Transfer Learning in Fault Prognostics”. In: *arXiv preprint arXiv:2301.01705*. 2023.

[Masc20] B. Maschler, H. Vietz, N. Jazdi, and M. Weyrich. “Continual learning of fault prediction for turbofan engines using deep learning with elastic weight consolidation”. In: *25th IEEE international conference on emerging technologies and factory automation (ETFA)*. Vol. 1. IEEE. 2020.

[Md22] A. Q. Md, K. Jha, S. Haneef, A. K. Sivaraman, and K. F. Tee. “A review on data-driven quality prediction in the production process with machine learning for industry 4.0”. In: *Processes* 10. 2022.

[Mell19] R. F. de Mello, Y. Vaz, C. H. Grossi, and A. Bifet. “On learning guarantees to unsupervised concept drift detection on data streams”. In: *Expert Systems with Applications* 117. 2019.

[Mera19] C. Mera, M. Orozco-Alzate, and J. Branch. “Incremental learning of concept drift in Multiple Instance Learning for industrial visual inspection”. In: *Computers in Industry* 109. 2019.

[Ming21] Y. Ming, X. Meng, C. Fan, and H. Yu. “Deep learning for monocular depth estimation: A review”. In: *Neurocomputing* 438. 2021.

[Mitc97] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Moha20] T. Mohanraj, S. Shankar, R. Rajasekar, N. Sakthivel, and A. Pramanik. “Tool condition monitoring techniques in milling process—A review”. In: *Journal of Materials Research and Technology* 9. 2020.

[Mole20] M. Moleda, A. Momot, and D. Mrozek. “Concept Drift and Avoiding its Negative Effects in Predictive Modeling of Failures of Electricity Production Units in Power Plants”. In: *28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2020.

[Murp12] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[Naei15] M. P. Naeini, G. F. Cooper, and M. Hauskrecht. “Obtaining Well Calibrated Probabilities Using Bayesian Binning”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[Naya21] P. A. Nayak, P. Sriganesh, K. Rakshitha, M. M. Kumar, B. Prashanth, and H. Sneha. “Concept drift and model decay detection using machine learning algorithm”. In: *12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE. 2021.

[Neto13] F. Neto, T. M. Gerônimo, C. E. D. Cruz, P. R. d. Aguiar, and E. Bianchi. “Neural models for predicting hole diameters in drilling processes”. In: *Procedia CIRP* 12. 2013.

[Okaw21] Y. Okawa and K. Kobayashi. “Concept drift detection via boundary shrinking”. In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021.

[Ovad19] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek. “Can you trust your model’s uncertainty? evaluating predictive uncertainty under dataset shift”. In: *Advances in neural information processing systems* 32. 2019.

[Page54] E. S. Page. “Continuous inspection schemes”. In: *Biometrika* 41. 1954.

[Pasq23] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba. “DARWIN: An online deep learning approach to handle concept drifts in predictive process monitoring”. In: *Engineering Applications of Artificial Intelligence* 123. 2023.

[Pasz19] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. 2019.

[Pati22] R. Patil, P. Patil, A. Ghongade, A. Dsa, P. Lokhande, and H. Munot. “Online System for Identifying Need of Machine Maintenance by Mining Data Streams and Handling Concept Drifts”. In: *Sentimental Analysis and Deep Learning: Proceedings of ICSADL 2021*. Springer. 2022.

[Pear01] K. Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2. 1901.

[Pedr11] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12. 2011.

[Peff07] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. “A design science research methodology for information systems research”. In: *Journal of management information systems* 24. 2007.

[PHM 10] PHM Society. *PHM Society 2010 Data Challenge*. Accessed: 2024-01-07. 2010. URL: *https://phmsociety.org/phm_competition/2010-phm-society-conference-data-challenge/*.

[Pian22] L. Piano, F. Garcea, V. Gatteschi, F. Lamberti, and L. Morra. “Detecting drift in deep learning: A methodology primer”. In: *IT Professional* 24. 2022.

[Piat14] G. Piatetsky. *What main methodology are you using for your analytics, data mining, or data science projects? Poll*. *https://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html*. Accessed: 2024-01-07. 2014.

[Pina15] F. A. Pinage and E. M. dos Santos. “A dissimilarity-based drift detection method”. In: *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2015.

[Plat99] J. Platt et al. “Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods”. In: *Advances in large margin classifiers* 10. 1999.

[Porw22] P. Porwik and B. M. Dadzie. “Detection of data drift in a two-dimensional stream using the Kolmogorov-Smirnov test”. In: *Procedia Computer Science* 207. 2022.

[Preu18] C. Preusche. “Clusterbasierte Zustandsbewertung von technischen Systemen zur Unterstützung der prädiktiven Instandhaltung”. PhD thesis. 2018.

[Pric17] PriceWaterhouseCoopers (PWC). *Sizing the Prize: What's the Real Value of AI for Your Business and How Can You Capitalise?* PriceWaterhouseCoopers (PWC). 2017.

[Quin14] J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[Quin86] J. R. Quinlan. "Induction of decision trees". In: *Machine learning* 1. 1986.

[Raab20] C. Raab, M. Heusinger, and F.-M. Schleif. "Reactive soft prototype computing for concept drift streams". In: *Neurocomputing* 416. 2020.

[Raba19] S. Rabanser, S. Günnemann, and Z. Lipton. "Failing loudly: An empirical study of methods for detecting dataset shift". In: *Advances in Neural Information Processing Systems* 32. 2019.

[Redm16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[Reho05] A. G. Rehorn, J. Jiang, and P. E. Orban. "State-of-the-art methods and results in tool condition monitoring: a review". In: *The International Journal of Advanced Manufacturing Technology* 26. 2005.

[Rose58] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65. 1958.

[Sagl03] H. Saglam and A. Unuvar. "Tool condition monitoring in milling based on cutting forces by a neural network". In: *International Journal of Production Research* 41. 2003.

[Samu59] A. L. Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of research and development* 3. 1959.

[Sari23] E. Sarikaya, M. von Elling, X. Lu, and M. Weigold. "Continual Learning Based Machining Simulation for the Prediction of NC Signals". In: *Procedia CIRP* 120. 2023.

[Sark13] S. Sarker, X. Xiao, and T. Beaulieu. "Guest editorial: Qualitative studies in information systems: A critical review and some guiding principles". In: *MIS quarterly* 37. 2013.

[Sche15] S. Schelter, F. Biessmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas. *On challenges in machine learning model management*. Amazon. 2015.

[Seif21] C. Seiffer, H. Ziekow, U. Schreier, and A. Gerling. "Detection of Concept Drift in Manufacturing Data with SHAP Values to Improve Error Prediction". In: *Data Analytics*. 2021.

[Sen23] S. Sen, S. M. Nielsen, E. J. Husom, A. Goknil, S. Tverdal, and L. S. Pinilla. "Replay-Driven Continual Learning for the Industrial Internet of Things". In: *IEEE/ACM 2nd International Conference on AI Engineering–Software Engineering for AI (CAIN)*. IEEE. 2023.

[Seri20] G. Serin, B. Sener, A. Ozbayoglu, and H. Unver. "Review of tool condition monitoring in machining and opportunities for deep learning". In: *The International Journal of Advanced Manufacturing Technology*. 2020.

[Seth15] T. S. Sethi and M. Kantardzic. "Don't pay for validation: Detecting drifts from unlabeled data using margin density". In: *Procedia Computer Science* 53. 2015.

[Seth17] T. S. Sethi and M. Kantardzic. "On the reliable detection of concept drift from streaming unlabeled data". In: *Expert Systems with Applications* 82. 2017.

[Shan17] D. Shang, G. Zhang, and J. Lu. "Fast concept drift detection using singular vector decomposition". In: *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. IEEE. 2017.

[Soll20] S. Soller, G. Hölzl, and M. Kranz. "Predicting machine errors based on adaptive sensor data drifts in a real world industrial setup". In: *IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE. 2020.

[Soua14] A. Soualhi, K. Medjaher, and N. Zerhouni. "Bearing health monitoring based on Hilbert–Huang transform, support vector machine, and regression". In: *IEEE Transactions on instrumentation and measurement* 64. 2014.

[Souz20a] V. M. Souza, F. A. Chowdhury, and A. Mueen. "Unsupervised drift detection on high-speed data streams". In: *IEEE International Conference on Big Data (Big Data)*. IEEE. 2020.

[Souz20b] V. M. Souza, D. M. dos Reis, A. G. Maletzke, and G. E. Batista. "Challenges in benchmarking stream learning algorithms with real-world data". In: *Data Mining and Knowledge Discovery* 34. 2020.

[Stud21] S. Studer, T. B. Bui, C. Drescher, A. Hanuschkin, L. Winkler, S. Peters, and K.-R. Müller. "Towards CRISP-ML (Q): a machine learning process model with quality assurance methodology". In: *Machine learning and knowledge extraction* 3. 2021.

[Sun19] R. Sun and C. H. Lampert. "KS (conf): A light-weight test if a ConvNet operates outside of its specifications". In: *40th German Conference on Pattern Recognition (GCPR)*. Springer. 2019.

[Syme22] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas. "Mlops-definitions, tools and challenges". In: *IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2022.

[Tan21] C. H. Tan, V. C. Lee, M. Salehi, S. Marusic, S. Jayawardena, and D. Lucke. "A fully unsupervised and efficient anomaly detection approach with drift detection capability". In: *International conference on data mining workshops (ICDMW)*. IEEE. 2021.

[Terc22a] H. Tercan, P. Deibert, and T. Meisen. "Continual learning of neural networks for quality prediction in production using memory aware synapses and weight transfer". In: *Journal of Intelligent Manufacturing* 33. 2022.

[Terc19] H. Tercan, A. Guajardo, and T. Meisen. "Industrial transfer learning: Boosting machine learning in production". In: *IEEE 17th international conference on industrial informatics (INDIN)*. Vol. 1. IEEE. 2019.

[Terc22b] H. Tercan and T. Meisen. "Machine learning and deep learning based predictive quality in manufacturing: a systematic review". In: *Journal of Intelligent Manufacturing* 33. 2022.

[Test22] M. Testi, M. Ballabio, E. Frontoni, G. Iannello, S. Moccia, P. Soda, and G. Vessio. "MLOps: A taxonomy and a methodology". In: *IEEE Access* 10. 2022.

[Tian19] H. Tian, N. L. D. Khoa, A. Anaissi, Y. Wang, and F. Chen. "Concept drift adaption for online anomaly detection in structural health monitoring". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019.

[Tian21] H. Tian, D. Ren, K. Li, and Z. Zhao. “An adaptive update model based on improved long short term memory for online prediction of vibration signal”. In: *Journal of Intelligent Manufacturing* 32. 2021.

[Ting19] T. Tinga and R. Loendersloot. “Physical model-based prognostics and health monitoring to enable predictive maintenance”. In: *Predictive Maintenance in Dynamic Systems: Advanced Methods, Decision Support Tools and Real-World Applications*. 2019.

[Tnan22a] M.-A. Tnani, M. Feil, and K. Diepold. “Smart data collection system for brownfield CNC milling machines: A new benchmark dataset for data-driven machine monitoring”. In: *Procedia CIRP* 107. 2022.

[Tnan22b] M.-A. Tnani, P. Subarnaduti, and K. Diepold. “Efficient feature learning approach for raw industrial vibration data using two-stage learning framework”. In: *Sensors* 22. 2022.

[Tora15] A. J. Torabi, M. J. Er, X. Li, B. S. Lim, and G. O. Peen. “Application of clustering methods for online tool condition monitoring and fault diagnosis in high-speed milling processes”. In: *IEEE Systems Journal* 10. 2015.

[Tsym04] A. Tsymbal. “The problem of concept drift: definitions and related work”. In: *Computer Science Department, Trinity College Dublin* 106. 2004.

[Ultr24] T. Ultrasonics. *Sieving Resonators*. *https://www.telsonic.com/en/products/sieving-resonators/*. Accessed: 2024-01-28. 2024.

[Vais07] V. K. Vaishnavi. *Design science research methods and patterns: innovating information and communication technology*. Auerbach Publications, 2007.

[Van 08] L. Van der Maaten and G. Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11. 2008.

[Van 19] A. Van Looveren, J. Klaise, G. Vacanti, O. Cobb, A. Scillitoe, R. Samoilescu, and A. Athorne. *Alibi detect: Algorithms for outlier, adversarial and drift detection*. 2019.

[Vela22] D. Vela, A. Sharp, R. Zhang, T. Nguyen, A. Hoang, and O. S. Pianykh. “Temporal quality degradation in AI models”. In: *Scientific reports* 12. 2022.

[Vers17] D. Verstraete, A. Ferrada, E. L. Droguett, V. Meruane, M. Modarres, et al. “Deep learning enabled fault diagnosis using time-frequency image analysis of rolling element bearings”. In: *Shock and Vibration* 2017. 2017.

[Wan22] Y.-N. Wan, B. P. Jaysawal, and J.-W. Huang. “Unsupervised Concept Drift Detection Using Dynamic Crucial Feature Distribution Test in Data Streams”. In: *International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. IEEE. 2022.

[Wang17] L.-H. Wang, X.-P. Zhao, J.-X. Wu, Y.-Y. Xie, and Y.-H. Zhang. “Motor fault diagnosis based on short-time Fourier transform and convolutional neural network”. In: *Chinese Journal of Mechanical Engineering* 30. 2017.

[Wang20] P. Wang, N. Jin, and G. Fehringer. “Concept drift detection with false positive rate for multi-label classification in iot data stream”. In: *International Conference on UK-China Emerging Technologies (UCET)*. IEEE. 2020.

[Webb16] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean. “Characterizing concept drift”. In: *Data mining and knowledge discovery* 30. 2016.

[Widm96] G. Widmer and M. Kubat. “Learning in the Presence of Concept Drift and Hidden Contexts”. In: *Machine learning* 23.1. 1996.

[Wils20] G. Wilson and D. J. Cook. “A Survey of Unsupervised Deep Domain Adaptation”. In: *ACM transactions on intelligent systems and technology* 11.5. 2020.

[Wirt00] R. Wirth and J. Hipp. “CRISP-DM: Towards a standard process model for data mining”. In: *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*. Vol. 1. Springer-Verlag London, UK. 2000.

[Wu21] X. Wu, M. El-Shamouty, and P. Wagner. *Whitepaper: Dependable AI.Using AI in Safety-Critical Industrial Applications*. 2021.

[Wues16] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben. “Machine learning in manufacturing: advantages, challenges, and applications”. In: *Prod. Manuf. Res.* 4. 2016.

[Xuan20] J. Xuan, J. Lu, and G. Zhang. “Bayesian nonparametric unsupervised concept drift detection for data stream mining”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 12. 2020.

[Yang20] J. Yang, J. Zhang, and S. Qin. “A Concept Drift Detection Algorithm based on Fuzzy Marginal Density”. In: *Proceedings of the 2020 5th International Conference on Mathematics and Artificial Intelligence*. 2020.

[Yang21] L. Yang and A. Shami. “A lightweight concept drift detection and adaptation framework for IoT data streams”. In: *IEEE Internet of Things Magazine* 4. 2021.

[Yong20] B. X. Yong, Y. Fathy, and A. Brintrup. “Bayesian autoencoders for drift detection in industrial environments”. In: *IEEE International Workshop on Metrology for Industry 4.0 & IoT*. IEEE. 2020.

[Yuan21] Y. Yuan, Z. Wang, and W. Wang. “Unsupervised concept drift detection based on multi-scale slide windows”. In: *Ad Hoc Networks* 111. 2021.

[Zeni19] J. Zenisek, F. Holzinger, and M. Affenzeller. “Machine learning based concept drift detection for predictive maintenance”. In: *Computers & Industrial Engineering* 137. 2019.

[Zhan23] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into deep learning*. Cambridge University Press, 2023.

[Zhan16] C. Zhang, X. Yao, J. Zhang, and H. Jin. “Tool condition monitoring and remaining useful life prognostic based on a wireless sensor in dry milling operations”. In: *Sensors* 16. 2016.

[Zhou18] Y. Zhou and W. Xue. “Review of tool condition monitoring methods in milling processes”. In: *The International Journal of Advanced Manufacturing Technology* 96. 2018.

[Zieg20] A. Ziegenbein, A. Fertig, J. Metternich, and M. Weigold. “Data-based process analysis in machining production: Case study for quality determination in a drilling process”. In: *Procedia CIRP* 93. 2020.

[Žlio10a] I. Žliobaite. “Change with delayed labeling: When is it detectable?” In: *IEEE international conference on data mining workshops*. IEEE. 2010.

[Žlio10b] I. Žliobaitė. “Adaptive training set formation”. PhD thesis. 2010.

[Žlio16] I. Žliobaitė, M. Pechenizkiy, and J. Gama. “An overview of concept drift applications”. In: *Big data analysis: new algorithms for a new society*. 2016.

Norms

[ISO/IEC 24029-2] ISO/IEC 24029-2:2023. *Artificial intelligence (AI) - Assessment of the robustness of neural networks - Part 2: Methodology for the use of formal methods*.

[DIN EN 13306] DIN EN 13306:2018. *Instandhaltung - Begriffe der Instandhaltung*.

[DIN 6583] DIN 6583:1981. *Begriffe der Zerspantechnik; Standbegriffe*.

[DIN EN ISO 9001] DIN EN ISO 9001:2015. *Qualitätsmanagementsysteme - Anforderungen*.

[DIN EN ISO 9000] DIN EN ISO 9000:2015. *Qualitätsmanagementsysteme - Grundlagen und Begriffe*.

[ISO 13374-2] ISO 13374-2:2007. *Condition monitoring and diagnostics of machines – Data processing, communication and presentation – Part 2: Data processing*.

[NISTIR 8012] NISTIR 8012:2014. *Standards related to prognostics and health management (PHM) for manufacturing*.

Student theses

[Wint23a] A. Winter. “Analysis of Uncertainty Estimation Methods to Detect Drifts in Data Streams”. Bachelors’s thesis. Technical University of Darmstadt, 2023.

[Zhu22] F. Zhu. “Präventive Erkennung von Siebfehlern bei der Pigmentsiebung durch maschinelles Lernen”. Master’s thesis. Technical University of Darmstadt, 2022.

A. Supplementary material for Chapter 4

A.1. Literature review studies

Table A.1.: Studies analyzed within the systematic literature review in Section 4.3.1 ordered by year of publication. Studies marked with (*) appear in more than one category.

Data modeling category	Title	Year	Reference
Input data distribution	Change with delayed labeling: When is it detectable?*	2010	[Žlio10a]
	Hellinger distance based drift detection for nonstationary environments	2011	[Ditz11]
	Detection of concept drift for learning from stream data	2012	[Lee12]
	A dissimilarity-based drift detection method	2015	[Pina15]
	A2D2: A pre-event abrupt drift detection	2015	[Esco15]
	Fast unsupervised online drift detection using incremental kolmogorov-smirnov test	2016	[Dos 16]
	Using dynamical systems tools to detect concept drift in data streams	2016	[Cost16]
	Fedd: Feature extraction for explicit concept drift detection in time series	2016	[Cava16]
	Detecting and adapting to concept drift in continually evolving stochastic processes	2017	[Gama17]
	Concept drift detection for data stream learning based on angle optimized global embedding and principal component analysis in sensor networks	2017	[Liu17]
	On applying the restricted Boltzmann machine to active concept drift detection	2017	[Jawo17]
	Multidimensional surrogate stability to detect data stream concept drift	2017	[Cost17]
	Fast concept drift detection using singular vector decomposition	2017	[Shan17]
	Accumulating regional density dissimilarity for concept drift detection in data streams	2018	[Liu18]

Continued on next page

Table A.1 – continued from previous page

Data modeling category	Title	Year	Reference
Input data distribution	DetectA: abrupt concept drift detection in non-stationary environments	2018	[Esco18]
	Unsupervised drift detector ensembles for data stream mining	2019	[Kory19]
	Failing loudly: An empirical study of methods for detecting dataset shift*	2019	[Raba19]
	Unsupervised concept drift detection with a discriminative classifier	2019	[Gözü19]
	On learning guarantees to unsupervised concept drift detection on data streams	2019	[Mell19]
	Bayesian nonparametric unsupervised concept drift detection for data stream mining	2020	[Xuan20]
	Reactive soft prototype computing for concept drift streams	2020	[Raab20]
	Reactive concept drift detection using coresets over sliding windows	2020	[Heus20]
	Unsupervised drift detection on high-speed data streams	2020	[Souz20a]
	Concept drift detection via equal intensity k-means space partitioning	2020	[Liu20]
	Unsupervised concept drift detection based on multi-scale slide windows	2021	[Yuan21]
	A novel approach to detect concept drift using machine learning	2021	[Huss21]
	Concept drift and model decay detection using machine learning algorithm	2021	[Naya21]
	A fully unsupervised and efficient anomaly detection approach with drift detection capability	2021	[Tan21]
	Concept drift detection from multi-class imbalanced data streams	2021	[Kory21]
	A shape-based method for concept drift detection and signal denoising	2021	[Hind21]
	Detection of data drift in a two-dimensional stream using the Kolmogorov-Smirnov test	2022	[Porw22]
	Reconstruction-based unsupervised drift detection over multivariate streaming data	2022	[Kami22]
	Unsupervised Concept Drift Detection Using Dynamic Crucial Feature Distribution Test in Data Streams	2022	[Wan22]
	Context-aware drift detection	2022	[Cobb22]
Prediction confidence	Change with delayed labeling: When is it detectable?*	2010	[Žlio10a]
	Drift detection using uncertainty distribution divergence	2013	[Lind13]
	Don't pay for validation: Detecting drifts from unlabeled data using margin density	2015	[Seth15]

Continued on next page

Table A.1 – continued from previous page

Data modeling category	Title	Year	Reference
Prediction confidence	Drift detection in data stream classification without fully labelled instances	2015	[Lugh15]
	Concept drift detection on streaming data under limited labeling	2016	[Kim16]
	On the reliable detection of concept drift from streaming unlabeled data	2017	[Seth17]
	Detecting and correcting for label shift with black box predictors	2018	[Lipt18]
	Combining active learning with concept drift detection for data stream mining	2018	[Kraw18]
	A drift detection method based on active learning	2018	[Cost18]
	Failing loudly: An empirical study of methods for detecting dataset shift*	2019	[Raba19]
	KS (conf): A light-weight test if a ConvNet operates outside of its specifications	2019	[Sun19]
	A concept drift detection algorithm based on fuzzy marginal density	2020	[Yang20]
	Detection of data drift and outliers affecting machine learning model performance over time	2020	[Acke20b]
	Born-again decision boundary: Unsupervised concept drift detection by inspector neural network	2021	[King21]
	Detecting concept drift with neural network model uncertainty	2021	[Baie23]
	Automatically detecting data drift in machine learning classifiers	2021	[Acke21b]
	Concept drift detection via boundary shrinking	2021	[Okaw21]
	Augur: A step towards realistic drift detection in production ml systems	2022	[Lewi22]
Learned embeddings	Sequential drift detection in deep learning classifiers	2020	[Acke20a]
	Drift lens: Real-time unsupervised concept drift detection by evaluating per-label embedding distributions	2021	[Grec21]
	Task-sensitive concept drift detector with constraint embedding	2021	[Cast21]
	Machine learning model drift detection via weak data slices	2021	[Acke21a]
	Who supervises the supervisor? Model monitoring in production using deep feature embeddings with applications to workpiece inspection	2022	[Banf22]
	Detecting drift in deep learning: A methodology primer	2022	[Pian22]

B. Supplementary material for Chapter 5

B.1. Scalar features

Table B.1.: Statistical and spectral features that are extracted from univariate time series $\{x_i\}_{i=1}^n$ and consequently used for the training of the shallow models within the case study experiments. In the case of multivariate time series, the features are computed for each signal separately and consequently concatenated, *cf.* Figure 4.7. Own illustration.

Category	Name and computation	
Time domain	RMS	$\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$
	Variance	$\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n}$
	Mean	$\frac{1}{n}\sum_{i=1}^{n} x_i$
	Median	$\operatorname{median}(\{x_i\})$
	Maximum	$\max(\{x_i\})$
	Minimum	$\min(\{x_i\})$
	Skewness	$E\left(\left(\frac{x_i-\mu}{\sigma}\right)^3\right)$
	Kurtosis	$E\left(\left(\frac{x_i-\mu}{\sigma}\right)^4\right)$
	Peak-to-Peak	$\max(\{x_i\}) - \min(\{x_i\})$
	Zero-Crossing Rate	$\frac{1}{2}\sum_{i=1}^{n-1} \lvert \text{sgn}(x_{i+1}) - \text{sgn}(x_i) \rvert$
Frequency domain	Spectral Skewness	$\sum_{i=1}^{k}\left(\frac{f_i-\bar{f}}{\sigma_f}\right)^3 S\left(f_i\right)$
	Spectral Kurtosis	$\sum_{i=1}^{k}\left(\frac{f_i-\bar{f}}{\sigma_f}\right)^4 S\left(f_i\right)$
	Spectral Centroid	$\frac{\sum_{i=1}^{k} f_i S(f_i)}{\sum_{i=1}^{k} S(f_i)}$
	Spectral Spread	$\sqrt{\frac{\sum_{i=1}^{k}(f_i-\text{Spectral Centroid})^2 S(f_i)}{\sum_{i=1}^{k} S(f_i)}}$
	Spectral Entropy	$-\sum_{i=1}^{k} \frac{S(f_i)}{\sum_{j=1}^{k} S(f_j)} \log_2\left(\frac{S(f_i)}{\sum_{j=1}^{k} S(f_j)}\right)$
Time-frequency domain	Wavelet energy	$\sum_{i=1}^{a}\sum_{j=1}^{b} \lvert\tilde{x}_{i,j}\rvert^2$

B.2. Case study 1: Tool condition monitoring in a CNC milling process

Table B.2.: Final hyperparameter settings in the tool condition monitoring case study.

Method	Hyperparameter	Final fitted value
RF	bootstrap	false
	max_depth	10
	max_features	sqrt
	min_samples_leaf	1
	min_samples_split	5
	n_estimators	100
SVC	C	1
	degree	2
	gamma	scale
	kernel	rbf
KNN	metric	euclidean
	n_neighbors	1
	weights	uniform
DT	criterion	entropy
	max_depth	30
	min_samples_leaf	1
	min_samples_split	5
MLP	activation	relu
	alpha	0.0001
	hidden_layer_sizes	50
	learning_rate_init	0.001
	learning_rate	invscaling
	max_iter	200
	optimizer	adam
CNN	learning_rate_init	0.001
	optimizer	adam
	max_iter	20

Table B.3.: CNN architecture used for the tool condition monitoring case study.

Architecture part	Layer type	Output shape (NCHW)	Parameters
Feature extractor	Input	$(b, 3, 128, 256)$	0
	Conv2d	$(b, 32, 128, 256)$	2432
	Conv2d	$(b, 64, 10, 58)$	51264
	Linear	$(b, 32)$	296992
Classifier	Linear	$(b, 1)$	33
	Sigmoid	$(b, 1)$	0

Table B.4.: Drift detection performance of LRDD for different choices of the number of neighbors k and two-sample tests.

		Precision			Recall		
Test	k	$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.05$	$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.05$
MMD	1	0.803 ± 0.003	0.793 ± 0.002	0.769 ± 0.000	0.980 ± 0.016	0.993 ± 0.009	1.000 ± 0.000
MMD	3	0.723 ± 0.002	0.704 ± 0.000	0.685 ± 0.000	0.993 ± 0.009	1.000 ± 0.000	1.000 ± 0.000
MMD	5	0.714 ± 0.000	0.694 ± 0.000	0.602 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
MMD	8	0.658 ± 0.000	0.588 ± 0.000	0.550 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
KS	1	0.607 ± 0.005	0.602 ± 0.000	0.562 ± 0.000	0.987 ± 0.020	1.000 ± 0.000	1.000 ± 0.000
KS	3	0.544 ± 0.000	0.544 ± 0.000	0.510 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
KS	5	0.521 ± 0.000	0.521 ± 0.000	0.510 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
KS	8	0.510 ± 0.000	0.510 ± 0.000	0.510 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000

		F_1-Scores		
Test	k	$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.05$
MMD	1	0.883 ± 0.007	0.882 ± 0.004	0.869 ± 0.000
MMD	3	0.837 ± 0.003	0.826 ± 0.000	0.813 ± 0.000
MMD	5	0.833 ± 0.000	0.819 ± 0.000	0.752 ± 0.000
MMD	8	0.794 ± 0.000	0.741 ± 0.000	0.710 ± 0.000
KS	1	0.752 ± 0.007	0.752 ± 0.000	0.720 ± 0.000
KS	3	0.705 ± 0.000	0.705 ± 0.000	0.675 ± 0.000
KS	5	0.685 ± 0.000	0.685 ± 0.000	0.675 ± 0.000
KS	8	0.675 ± 0.000	0.675 ± 0.000	0.675 ± 0.000

Table B.5.: Drift detection precision in scenario b.2 of the tool condition monitoring case study. Mean precision and the corresponding standard deviation are reported over 10 experiment runs.

Configuration	Data modeling	Detector	Precision $\alpha = 0.005$	 $\alpha = 0.01$	 $\alpha = 0.05$
Spindle change	Features	KS	0.500 ± 0.000	0.500 ± 0.000	0.500 ± 0.000
		MMD	0.502 ± 0.003	0.501 ± 0.001	0.500 ± 0.001
		ContextMMD	0.798 ± 0.013	0.767 ± 0.006	0.642 ± 0.026
		LRDD	0.858 ± 0.024	0.822 ± 0.020	0.786 ± 0.021
Crooked mounting	Features	KS	0.500 ± 0.000	0.500 ± 0.000	0.500 ± 0.000
		MMD	0.502 ± 0.003	0.501 ± 0.001	0.500 ± 0.001
		ContextMMD	0.428 ± 0.061	0.447 ± 0.038	0.463 ± 0.031
		LRDD	0.153 ± 0.069	0.223 ± 0.072	0.313 ± 0.074
Three-flute cutter	Features	KS	0.500 ± 0.000	0.500 ± 0.000	0.500 ± 0.000
		MMD	0.502 ± 0.003	0.501 ± 0.001	0.500 ± 0.001
		ContextMMD	0.798 ± 0.013	0.767 ± 0.006	0.642 ± 0.026
		LRDD	0.858 ± 0.024	0.822 ± 0.020	0.786 ± 0.021
Parameter drift	Features	KS	0.500 ± 0.000	0.500 ± 0.000	0.500 ± 0.000
		MMD	0.502 ± 0.003	0.501 ± 0.001	0.500 ± 0.001
		ContextMMD	0.796 ± 0.012	0.766 ± 0.005	0.642 ± 0.026
		LRDD	0.857 ± 0.025	0.821 ± 0.020	0.786 ± 0.021

Table B.6.: Drift detection detection in scenario b.2 of the tool condition monitoring case study. Mean recall and the corresponding standard deviation are reported over 10 experiment runs.

Configuration	Data modeling	Detector	Recall $\alpha = 0.005$	 $\alpha = 0.01$	 $\alpha = 0.05$
Spindle change	Features	KS	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		MMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		ContextMMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		LRDD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
Crooked mounting	Features	KS	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		MMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		ContextMMD	0.193 ± 0.047	0.247 ± 0.031	0.480 ± 0.014
		LRDD	0.030 ± 0.013	0.063 ± 0.023	0.126 ± 0.039
Three-flute cutter	Features	KS	0.667 ± 0.000	0.667 ± 0.000	0.667 ± 0.000
		MMD	0.668 ± 0.002	0.667 ± 0.001	0.667 ± 0.001
		ContextMMD	0.888 ± 0.008	0.868 ± 0.004	0.782 ± 0.019
		LRDD	0.923 ± 0.014	0.902 ± 0.012	0.880 ± 0.013
Parameter drift	Features	KS	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		MMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		ContextMMD	0.987 ± 0.009	0.993 ± 0.005	1.000 ± 0.000
		LRDD	0.994 ± 0.010	0.998 ± 0.006	1.000 ± 0.000

Table B.7.: Drift detection precision in scenario c.1 of the tool condition monitoring case study. Mean precision and the corresponding standard deviation are reported over 10 experiment runs.

Configuration	Data modeling	Detector	Precision $\alpha = 0.005$	 $\alpha = 0.01$	 $\alpha = 0.05$
Spindle change	Entropy	KS (RF)	0.875 ± 0.027	0.871 ± 0.026	0.798 ± 0.033
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.495 ± 0.005	0.498 ± 0.004	0.502 ± 0.003
		KS (kNN)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		KS (MLP)	0.590 ± 0.035	0.573 ± 0.034	0.565 ± 0.019
		KS (CNN)	0.486 ± 0.003	0.489 ± 0.003	0.497 ± 0.004
Crooked mounting	Entropy	KS (RF)	0.508 ± 0.091	0.569 ± 0.069	0.563 ± 0.047
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.501 ± 0.007	0.502 ± 0.005	0.502 ± 0.004
		KS (kNN)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (MLP)	0.526 ± 0.026	0.533 ± 0.027	0.542 ± 0.015
		KS (CNN)	0.505 ± 0.000	0.505 ± 0.000	0.503 ± 0.003
Three-flute cutter	Entropy	KS (RF)	0.878 ± 0.026	0.873 ± 0.026	0.798 ± 0.032
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.473 ± 0.008	0.486 ± 0.008	0.500 ± 0.004
		KS (kNN)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		KS (MLP)	0.652 ± 0.029	0.635 ± 0.029	0.617 ± 0.018
		KS (CNN)	0.497 ± 0.002	0.502 ± 0.003	0.499 ± 0.004
Parameter drift	Entropy	KS (RF)	0.877 ± 0.027	0.872 ± 0.026	0.798 ± 0.032
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.504 ± 0.005	0.504 ± 0.004	0.503 ± 0.003
		KS (kNN)	0.800 ± 0.400	0.900 ± 0.300	1.000 ± 0.000
		KS (MLP)	0.617 ± 0.037	0.605 ± 0.029	0.590 ± 0.020
		KS (CNN)	0.487 ± 0.002	0.501 ± 0.004	0.503 ± 0.003

Table B.8.: Drift detection recall in scenario c.1 of the tool condition monitoring case study. Mean recall and the corresponding standard deviation are reported over 10 experiment runs.

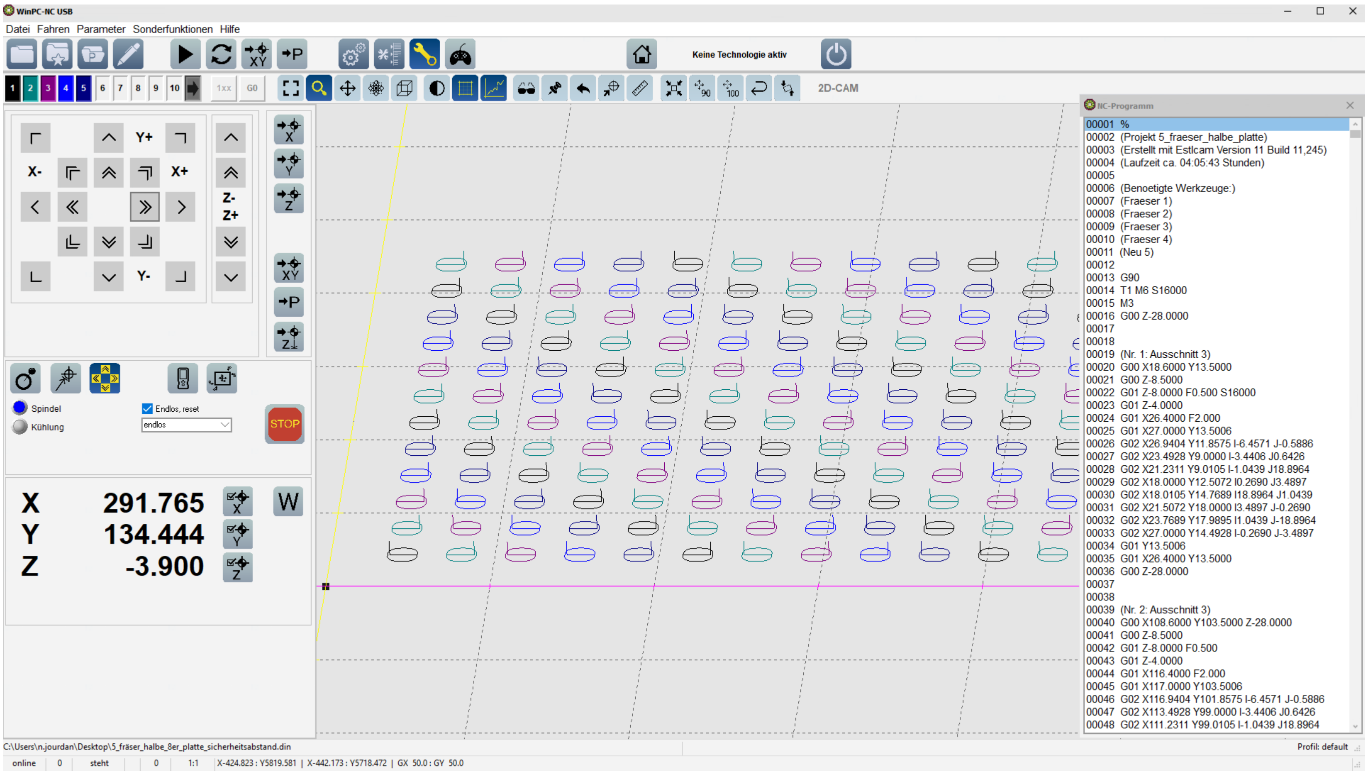
Configuration	Data modeling	Detector	Recall $\alpha = 0.005$	Recall $\alpha = 0.01$	Recall $\alpha = 0.05$
Spindle change	Entropy	KS (RF)	0.970 ± 0.015	0.980 ± 0.010	0.998 ± 0.004
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.965 ± 0.011	0.979 ± 0.012	0.996 ± 0.007
		KS (kNN)	0.165 ± 0.042	0.232 ± 0.055	0.300 ± 0.046
		KS (MLP)	0.646 ± 0.050	0.673 ± 0.052	0.768 ± 0.028
		KS (CNN)	0.923 ± 0.012	0.933 ± 0.012	0.973 ± 0.005
Crooked mounting	Entropy	KS (RF)	0.144 ± 0.035	0.191 ± 0.028	0.324 ± 0.029
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.987 ± 0.013	0.992 ± 0.010	0.995 ± 0.007
		KS (kNN)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (MLP)	0.497 ± 0.040	0.573 ± 0.034	0.699 ± 0.021
		KS (CNN)	0.997 ± 0.005	0.997 ± 0.005	1.000 ± 0.000
Three-flute cutter	Entropy	KS (RF)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	0.881 ± 0.030	0.933 ± 0.023	0.988 ± 0.010
		KS (kNN)	0.257 ± 0.031	0.302 ± 0.027	0.371 ± 0.024
		KS (MLP)	0.840 ± 0.033	0.874 ± 0.029	0.954 ± 0.018
		KS (CNN)	0.967 ± 0.005	0.983 ± 0.009	0.983 ± 0.009
Parameter drift	Entropy	KS (RF)	0.990 ± 0.004	0.991 ± 0.003	1.000 ± 0.000
		KS (DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS (SVC)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		KS (kNN)	0.016 ± 0.011	0.025 ± 0.014	0.037 ± 0.014
		KS (MLP)	0.723 ± 0.048	0.767 ± 0.034	0.850 ± 0.026
		KS (CNN)	0.923 ± 0.012	0.933 ± 0.012	0.973 ± 0.005

Table B.9.: Drift detection precision in scenario c.2 of the tool condition monitoring case study. Mean precision and the respective standard deviation are reported over 10 experiment runs.

			Precision		
Configuration	Data modeling	Detector	$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.05$
Spindle change	Embeddings (CNN)	KS	0.504 ± 0.001	0.503 ± 0.001	0.503 ± 0.001
		MMD	0.503 ± 0.001	0.503 ± 0.001	0.501 ± 0.001
		ContextMMD	0.815 ± 0.006	0.802 ± 0.008	0.748 ± 0.007
		LRDD	1.000 ± 0.000	1.000 ± 0.000	0.996 ± 0.006
Crooked mounting	Embeddings (CNN)	KS	0.504 ± 0.001	0.503 ± 0.001	0.503 ± 0.001
		MMD	0.503 ± 0.001	0.503 ± 0.001	0.501 ± 0.001
		ContextMMD	0.560 ± 0.052	0.585 ± 0.062	0.581 ± 0.047
		LRDD	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Three-flute cutter	Embeddings (CNN)	KS	0.504 ± 0.001	0.503 ± 0.001	0.503 ± 0.001
		MMD	0.503 ± 0.001	0.503 ± 0.001	0.501 ± 0.001
		ContextMMD	0.815 ± 0.006	0.802 ± 0.008	0.748 ± 0.007
		LRDD	0.815 ± 0.006	0.822 ± 0.006	0.996 ± 0.005
Parameter drift	Embeddings (CNN)	KS	0.504 ± 0.001	0.503 ± 0.001	0.503 ± 0.001
		MMD	0.503 ± 0.001	0.503 ± 0.001	0.501 ± 0.001
		ContextMMD	0.815 ± 0.006	0.802 ± 0.008	0.748 ± 0.007
		LRDD	1.000 ± 0.000	1.000 ± 0.000	0.997 ± 0.005

Table B.10.: Drift detection recall in scenario c.2 of the tool condition monitoring case study. Mean recall and the respective standard deviation are reported over 10 experiment runs.

			Recall		
Configuration	Data modeling	Detector	$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.05$
Spindle change	Embeddings (CNN)	KS	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		MMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		ContextMMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		LRDD	0.727 ± 0.034	0.770 ± 0.014	0.823 ± 0.009
Crooked mounting	Embeddings (CNN)	KS	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		MMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		ContextMMD	0.293 ± 0.049	0.357 ± 0.074	0.480 ± 0.100
		LRDD	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Three-flute cutter	Embeddings (CNN)	KS	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		MMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		ContextMMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		LRDD	1.000 ± 0.000	1.000 ± 0.056	0.860 ± 0.008
Parameter drift	Embeddings (CNN)	KS	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		MMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		ContextMMD	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		LRDD	0.903 ± 0.041	0.950 ± 0.029	0.990 ± 0.000



☑ Endlos, reset
endlos

X 291.765
Y 134.444
Z -3.900

```
%
(Projekt 5_fraeser_halbe_platte)
(Erstellt mit Estlcam Version 11 Build 11,245)
(Laufzeit ca. 04:05:43 Stunden)

(Benoetigte Werkzeuge:)
(Fraeser 1)
(Fraeser 2)
(Fraeser 3)
(Fraeser 4)
(Neu 5)

G90
T1 M6 S16000
M3
G00 Z-28.0000


(Nr. 1: Ausschnitt 3)
G00 X18.6000 Y13.5000
G00 Z-8.5000
G01 Z-8.0000 F0.500 S16000
G01 Z-4.0000
G01 X26.4000 F2.000
G01 X27.0000 Y13.5006
G02 X26.9404 Y11.8575 I-6.4571 J-0.5886
G02 X23.4928 Y9.0000 I-3.4406 J0.6426
G02 X21.2311 Y9.0105 I-1.0439 J18.8964
G02 X18.0000 Y12.5072 I0.2690 J3.4897
G02 X18.0105 Y14.7689 I18.8964 J1.0439
G02 X21.5072 Y18.0000 I3.4897 J-0.2690
G02 X23.7689 Y17.9895 I1.0439 J-18.8964
G02 X27.0000 Y14.4928 I-0.2690 J-3.4897
G01 Y13.5006
G01 X26.4000 Y13.5000
G00 Z-28.0000


(Nr. 2: Ausschnitt 3)
G00 X108.6000 Y103.5000 Z-28.0000
G00 Z-8.5000
G01 Z-8.0000 F0.500
G01 Z-4.0000
G01 X116.4000 F2.000
G01 X117.0000 Y103.5006
G02 X116.9404 Y101.8575 I-6.4571 J-0.5886
G02 X113.4928 Y99.0000 I-3.4406 J0.6426
G02 X111.2311 Y99.0105 I-1.0439 J18.8964
```

C:\Users\n.jourdan\Desktop\5_fräser_halbe_8er_platte_sicherheitsabstand.din

online | 0 | steht | | 0 | 1:1 | X-424.823 : Y5819.581 | X-442.173 : Y5718.472 | GX 50.0 : GY 50.0

Figure B.1.: Example of the tool assignment on the aluminum plates in the tool condition monitoring case study. Own illustration.

B.3. Case study 2: Predictive quality and process monitoring in a CNC milling process

Table B.11.: Selected features in the predictive quality and process monitoring case study.

Signal name	Axes	Features
CURRENT	Y Spindle	Kurtosis, Variance, Skewness Wavelet Energy, Maximum
TORQUE	Y Spindle	Kurtosis, Variance Wavelet energy,
POWER	Spindle	Minimum, Wavelet energy

Table B.12.: Final hyperparameter settings in the predictive quality and process monitoring case study.

Method	Hyperparameter	Final fitted value
RF	bootstrap	false
	max_depth	10
	max_features	sqrt
	min_samples_leaf	1
	min_samples_split	5
	n_estimators	100
SVC	C	1
	degree	2
	gamma	scale
	kernel	rbf
KNN	metric	euclidean
	n_neighbors	1
	weights	uniform
DT	criterion	entropy
	max_depth	30
	min_samples_leaf	1
	min_samples_split	5
MLP	activation	relu
	alpha	0.0001
	hidden_layer_sizes	50
	learning_rate_init	0.001
	learning_rate	invscaling
	max_iter	200
	optimizer	adam
IF	n_estimators	100
	max_samples	auto
	max_features	1.0
	bootstrap	false
LOF	n_neighbors	20
	algorithm	auto
	leaf_size	30
OC-SVM	kernel	rbf
	gamma	scale
	nu	0.05
	tol	1e-3

Table B.13.: Drift detection precision in the predictive quality and process monitoring case study. Mean precision and the respective standard deviations are reported over 10 experiment runs.

Dataset	Scenario	Detector	Precision		
			$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.05$
Early 2022	Scenario b.2	KS	0.906 ± 0.109	0.807 ± 0.088	0.868 ± 0.028
		MMD	0.996 ± 0.005	0.983 ± 0.006	0.957 ± 0.015
	Scenario c.1.1	KS(RF)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		KS(DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS(SVC)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		KS(kNN)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS(MLP)	1.000 ± 0.000	1.000 ± 0.000	0.923 ± 0.021
	Scenario c.1.2	KS(IF)	1.000 ± 0.000	0.973 ± 0.012	0.908 ± 0.024
		KS(OC-SVM)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		KS(LOF)	1.000 ± 0.000	1.000 ± 0.000	0.917 ± 0.018

Table B.14.: Drift detection recall in the predictive quality and process monitoring case study. Mean recall and the respective standard deviations are reported over 10 experiment runs.

Dataset	Scenario	Detector	Recall		
			$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.05$
Early 2022	Scenario b.2	KS	0.103 ± 0.037	0.137 ± 0.043	0.594 ± 0.042
		MMD	0.947 ± 0.024	0.999 ± 0.003	1.000 ± 0.000
	Scenario c.1.1	KS(RF)	0.787 ± 0.050	0.859 ± 0.054	0.967 ± 0.020
		KS(DT)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS(SVC)	0.101 ± 0.034	0.131 ± 0.049	0.371 ± 0.050
		KS(kNN)	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		KS(MLP)	0.710 ± 0.031	0.756 ± 0.033	0.940 ± 0.030
	Scenario c.1.2	KS(IF)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		KS(OC-SVM)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
		KS(LOF)	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000

B.4. Case study 3: Condition monitoring in a pigment sieving process

Table B.15.: Final hyperparameter settings in pigment sieving case study.

Method	Hyperparameter	Final fitted value
IF	`n_estimators`	100
	`max_samples`	auto
	`max_features`	1.0
	`bootstrap`	true
LOF	`n_neighbors`	20
	`algorithm`	auto
	`leaf_size`	30
OC-SVM	`kernel`	rbf
	`gamma`	scale
	`nu`	0.01
	`tol`	1e-4
CAE (STFT)	`learning_rate`	1e-4
	`epochs`	10
	`final_activation`	false
	`batch_size`	32
CAE (CWT)	`learning_rate`	1e-4
	`epochs`	5
	`final_activation`	false
	`batch_size`	64

Table B.16.: CAE architecture used for the condition monitoring in pigment production case study.

Architecture part	Layer type	Output shape (NCHW)	Parameters
Encoder	Input	$(b, 2, 128, 256)$	0
	Conv2d	$(b, 32, 128, 256)$	608
	ReLU	$(b, 32, 128, 256)$	0
	MaxPool2d	$(b, 32, 64, 128)$	0
	Conv2d	$(b, 16, 64, 128)$	4624
	ReLU	$(b, 16, 64, 128)$	0
	MaxPool2d	$(b, 16, 32, 64)$	0
	Conv2d	$(b, 8, 32, 64)$	1160
	ReLU	$(b, 8, 32, 64)$	0
	MaxPool2d	$(b, 8, 16, 32)$	0
	Conv2d	$(b, 4, 16, 32)$	292
	ReLU	$(b, 4, 16, 32)$	0
	MaxPool2d	$(b, 4, 8, 16)$	0
	Conv2d	$(b, 4, 8, 16)$	148
	ReLU	$(b, 4, 8, 16)$	0
Decoder	Upsample	$(b, 4, 16, 32)$	0
	Conv2d	$(b, 8, 16, 32)$	296
	ReLU	$(b, 8, 16, 32)$	0
	Upsample	$(b, 8, 32, 64)$	0
	Conv2d	$(b, 16, 32, 64)$	1168
	ReLU	$(b, 16, 32, 64)$	0
	Upsample	$(b, 16, 64, 128)$	0
	Conv2d	$(b, 32, 64, 128)$	4640
	ReLU	$(b, 32, 64, 128)$	0
	Upsample	$(b, 32, 128, 256)$	0
	Conv2d	$(b, 2, 128, 256)$	578

C. Comparison of neural network uncertainty estimation methods for drift detection

Parts of this chapter have been previously presented as a workshop contribution based on a bachelor thesis supervised by the thesis author in the scope of this dissertation titled "An Empirical Study of Uncertainty Estimation Techniques for Detecting Drift in Data Streams" at the Neural Information Processing Systems (NeurIPS) 2023 Workshop on Distribution Shifts. [Wint23a; Wint23b]

C.1. Methodology and experiments

An alternative to performance-based drift detectors is given by a class of drift detectors that work in an unsupervised way, utilizing a model's prediction confidence/uncertainty as a proxy for the error rate. Proposed approaches include Confidence Distribution Batch Detection (CDBD) [Lind13] and Margin Density Drift Detection (MD3) [Seth15]. More recently, Uncertainty Drift Detection (UDD) was proposed by Baier et al. [Baie23], which utilizes neural network uncertainty estimates from Monte Carlo Dropout (MCD) sampling [Gal16] as input for the Adaptive Windowing (ADWIN) detection algorithm [Bife07]. As MCD is only one possibility of extracting uncertainty estimates from NNs, an analysis on the influence of the choice of uncertainty estimation method on the performance of the overall drift detection capability is performed in this chapter. In prior work, Ovadia et al. [Ovad19] analyzed uncertainty estimation methods under dataset shift but only for synthetic drifts. While Baier et al. [Baie23] consider real-world datasets, they limit their experiments to a single uncertainty estimation method. Thus, the core contribution of this chapter is an empirical comparison of four state-of-the-art NN uncertainty estimation methods, as well as less a baseline method, for classification tasks in combination with the ADWIN detector to identify drifts in real-world data streams. These uncertainty estimators are evaluated using seven commonly used real-world datasets.

To compare the uncertainty estimation methods introduced in the following, two experiments are conducted for each method and dataset. Both start by training the method with the initial five percent of the whole data stream. The first experiment serves as a baseline and thus, the remaining data is tested without analyzing uncertainty estimates or triggering retrainings. In the main experiment, however, batches of the stream are evaluated and uncertainty estimates are used as a proxy for the error rate of the ADWIN detector. Once a drift is detected, a retraining is triggered with the initial five percent plus the most recent samples equivalent to one percent of the stream size. Thereby, models may adapt to new concepts while retaining sufficient generalization. Every experiment is repeated five times with different random seeds and results are averaged to allow for a fair comparison. Figure C.1 illustrates the process of the main experiment.

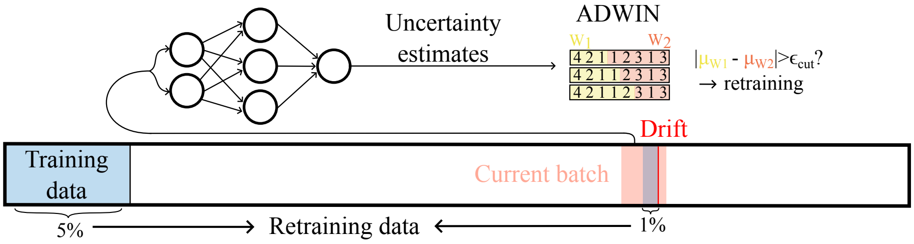


Figure C.1.: Approach to uncertainty drift detection. Own illustration.

C.2. Uncertainty estimation methods

To quantify model uncertainty, Bayesian NNs, which learn a posterior distribution over model parameters, can be employed [Ovad19]. This distribution enables the application of Bayesian Model Averaging (BMA) during inference. Therefore, P sets of weights $\boldsymbol{w}_i$ are drawn to gather a distribution of predictions $p_i(y|\boldsymbol{w}_i, \boldsymbol{x})$, given input features $\boldsymbol{x}$ and target labels y. The final prediction $p_{\text{BMA}}(y|\boldsymbol{x})$ is then given as the average

$$p_{\text{BMA}}(y|\boldsymbol{x}) = \frac{1}{P}\sum_{i=1}^{P} p_i(y|\boldsymbol{w}_i, \boldsymbol{x}). \tag{C.1}$$

For regression tasks, the uncertainty is the standard deviation of said distribution. While there are several uncertainty-related metrics for classification tasks, only Shannon's entropy H, *cf.* Equation (2.17), does not require ground-truth labels. Although Bayesian methods were previously considered state-of-the-art, they are computationally intractable for modern NNs with millions of parameters [Madd19]. Therefore, alternatives have been developed, of which the following are analyzed in the experiments. To get an uncertainty estimate, Shannon's entropy H is applied to the final prediction of each method.

Basic Neural Network. Given the focus on classification tasks, a distribution of predictions is not necessarily required. Hence, the simplest method is to use a single prediction from an unmodified NN. The motivation for this is to have a baseline for the more sophisticated methods.

MCD. Rather than drawing multiple weights from a posterior distribution as in BMA, a random dropout filter is applied to the neurons for several forward passes. These estimates are then averaged to get a final prediction. This allows for estimating the uncertainty in the model parameters based on the variability of the predictions across different dropout masks [Gal16; Baie23].

Ensemble. A distribution of predictions can also be won by training multiple neural networks. Different seeds of members introduce randomness due to their influence on the initial weights as well as the shuffling of data during training. As Lakshminarayanan et al. [Laks17] have shown, few members, i.e. 5, can be sufficient for good uncertainty estimates.

Stochastic Weight Averaging Gaussian (SWAG). Based on Stochastic Weight Averaging (SWA), a generalization technique in deep networks, Maddox et al. [Madd19] propose a method to approximate a posterior distribution over NN weights. Therefore, a Gaussian is fit utilizing the SWA solution as the first moment and a low rank plus diagonal covariance also inferred from stochastic gradient descent iterates. Given this posterior distribution, BMA is applied to get a final prediction.

Activation Shaping (ASH). The ASH method can be considered a more advanced version of the basic neural network, as it also works on single predictions. Djurisic et al. [Djur22] introduced it as an Out-of-Distribution (OOD) detection method that reaches state-of-the-art performance. Assuming over-parameterized feature representations in modern NNs, the hypothesis is that pruning a larger percentage of activations in a late layer helps with tasks such as OOD detection.

The hyperparameters of the introduced methods as well as the model architectures can be found in Appendix C.7.

C.3. Drift detector

Concept drift detectors, such as Drift Detection Method [Gama04], Page Hinkley Test [Page54], and ADWIN [Bife07], are typically error rate-based, necessitating access to costly true labels [Gonç14]. In contrast, data distribution-based detectors exclusively analyze input features, often using distance metrics like the Kolmogorov-Smirnov test [Raab20] to identify changes in feature distribution. Regardless of the detection method employed, distinguishing between noise and genuine concept drift poses a significant challenge [Tsym04], requiring a balance between swift adaptation to changes and resilience to noise. ADWIN offers performance guarantees for false positives and false negatives, making it an attractive choice. Furthermore, it is able to work with any real-valued input instead of being limited to an error rate between 0-1. As introduced by Bifet et al. [Bife07], ADWIN utilizes sliding windows of variable size. While no drift is present, new samples are added to a window W. After each sample, the algorithm attempts to find two sub-windows W_0 and W_1 that contain distinct averages. Once this happens a drift is assumed and the older sub-window is discarded. The variability of heterogeneous real-world data streams can be addressed by the sensitivity parameter $\delta \in (0, 1)$. The configuration for the experiments can be found in Appendix C.7.

C.4. Datasets

Seven real-world classification datasets from the USP Data Stream Repository [Souz20b] are used for the analysis. They encompass abrupt, incremental and reoccurring drifts, along with combinations thereof. In the **Gas** sensor dataset chemical sensor data is analyzed to identify one of six gases. The **Electricity** dataset focuses on predicting market price changes driven by supply and demand. For the **Rialto** dataset, segments of images from a timelapse with changing weather conditions shall be classified. Lastly, optical sensors are used to analyze moving patterns of flying insect species while drift is artificially introduced to generate the **InsAbr**, **InsInc**, **InsIncAbr** and **InsIncReo** datasets.

C.5. Metrics and results

For evaluation, the following two metrics are used to capture the quality of the uncertainty estimates as well as the drift detection performance:

Expected Calibration Error (ECE) ↓ [Naei15] measures the average deviation between prediction confidence and accuracy. As the name suggests, it quantifies how well a model is calibrated. It is hypothesized that calibration correlates positively with drift detection capability.

Matthew's Correlation Coefficient (MCC) ↑ is able to handle class imbalances which generally makes it a good metric for classification tasks [Chic20]. The MCC is employed to measure the overall prediction performance of the models, averaged over the complete experiment runs. It is hypothesized that poor drift detection performance will lead to unsuitable retraining points, in turn producing low MCC scores and vice versa.

Results

The results of the experiments can be found in Table C.1. To give an impression of the computational cost, the last row contains the total execution times of the main experiment. Analyzing the MCC values of the main experiment shows that the SWAG method offers the most balanced performance across all datasets. However, the gap in performance to the other methods is minimal. In fact, all methods perform fairly similarly. Surprisingly, even the basic method without any modifications keeps up with the others. Meanwhile, the execution time rises for the sampling-based approaches. Greater differences can be identified when analyzing the ECE as depicted in Figure C.2. Here, the SWAG method offers significantly better-calibrated predictions in nearly all datasets. The only exception is the InsIncAbr dataset, where all methods achieve a proficient calibration. All other methods appear to be equally worse calibrated compared to SWAG for the remaining datasets. Despite that, this does not directly translate to a better drift detection performance, as shown by the MCC values.

Table C.1.: Performance comparison of uncertainty estimation methods for drift detection. Table cells contain the average MCC (↑) values and (number of retrainings) for the naive baselines without retrainings (upper) and retrainings triggered by ADWIN when using the respective uncertainty estimation method (lower), respectively. Bold numbers indicate the best performance.

	Basic	MCD	Ensemble	SWAG	ASH
Gas	0.273 (0) 0.455 (36)	0.256 (0) 0.46 (55)	0.245 (0) **0.492** (50)	**0.299** (0) 0.46 (52)	0.275 (0) 0.459 (35)
Electricity	0.178 (0) 0.424 (11)	**0.198** (0) 0.421 (10)	0.183 (0) 0.405 (10)	0.191 (0) 0.419 (7)	0.175 (0) **0.438** (10)
Rialto	0.532 (0) 0.537 (43)	**0.534** (0) **0.553** (48)	0.505 (0) 0.527 (45)	0.52 (0) 0.54 (52)	0.525 (0) 0.539 (43)
InsAbr	0.471 (0) **0.519** (9)	0.472 (0) 0.509 (8)	0.461 (0) 0.503 (8)	**0.48** (0) 0.514 (6)	0.474 (0) 0.508 (7)
InsInc	0.087 (0) 0.241 (3)	**0.1** (0) 0.238 (3)	0.081 (0) 0.241 (3)	**0.1** (0) **0.301** (4)	0.085 (0) 0.231 (3)
InsIncAbr	0.304 (0) 0.53 (24)	0.307 (0) 0.525 (26)	0.308 (0) 0.518 (23)	0.299 (0) 0.445 (25)	**0.316** (0) **0.531** (25)
InsIncReo	0.141 (0) 0.253 (18)	0.133 (0) 0.247 (20)	**0.172** (0) 0.236 (18)	0.16 (0) **0.302** (21)	0.133 (0) 0.243 (20)
Total exec. time	6821s	7339s	15653s	9036s	6890s

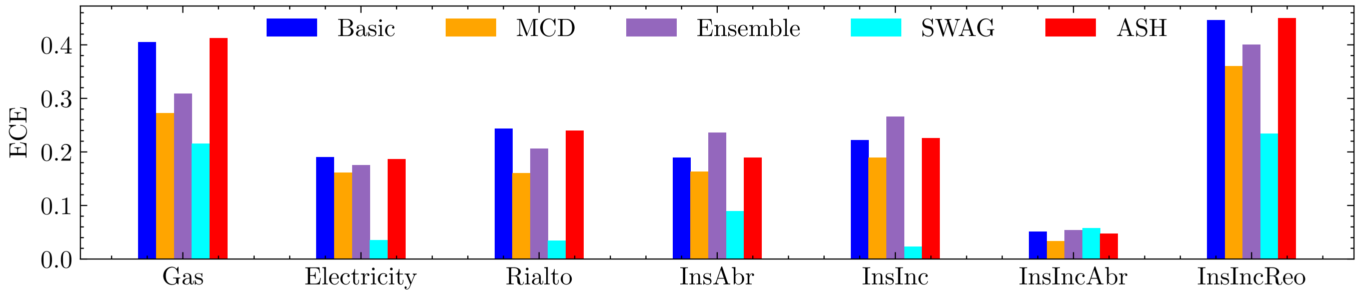


Figure C.2.: Calibration of the employed uncertainty estimation methods measured by ECE (↓) across the seven datasets. Own illustration.

C.6. Conclusion

In this chapter, five uncertainty estimation methods for classification tasks were implemented and evaluated in experiments including seven real-world datasets. The goal was to compare the utility of their uncertainty estimates for unsupervised concept drift detection by using them as a proxy for the error rate in combination with the ADWIN detector. Thereby, drift points in data streams shall be identified to trigger retrainings at the appropriate time and ultimately prevent model decay. Interestingly, even the baseline method, relying solely on the entropy calculated from the softmax scores, performed competitively with more sophisticated state-of-the-art methods. Moreover, all methods performed fairly similar in terms of overall classification performance as measured by the MCC metric. While the SWAG method achieved the most balanced MCC values and differences were only marginal. However, this was not the case when analyzing the ECE. Here the SWAG method offers significantly better calibrated predictions than all other methods. Regardless, these did not translate to better results for the drift detection. Thus, the assumption can be made, that the choice of method does not have a noteworthy influence on the performance of uncertainty-based concept drift detection for real-world applications.

To confirm the previous assumption, future work may include testing further real-world datasets, including regression problems. For those, the basic NN and the ASH method are no longer applicable. Instead, the effect of the ASH method in combination with the remaining approaches could be studied.

C.7. Hyperparameters and architectures

To make the experiments reproducible, Table C.2 gives an overview of the neural network architecture used for each dataset. Hidden layers use ReLU activations, while the softmax is applied in the final layer. The ADAM optimizer is used with binary or categorical cross-entropy loss, depending on the number of classes. For **MCD**, 100 forward passes are carried out. The **Ensemble** consists of three members. BMA is conducted with 100 samples from the posterior approximation of the **SWAG** method. Furthermore, the estimated covariance matrix utilized in the approach has a rank of 25 and is updated each epoch, starting at the first iteration. For the **ASH** method, the version termed ASH-p was chosen, where unpruned activations are not modified at all. Pruning is applied in the penultimate hidden layer (i.e. third last overall layer) with a pruning percentage of 60%. Lastly, Table C.3 indicates the sensitivity values δ for the ADWIN detector.

Table C.2.: Overview of model architectures.

Name	No. Layers	Neurons per layer	Dropout rate	Epochs
Gas	5	128, 64, 32, 16, 8	0.2	100
Electricity	3	32, 16, 8	0.1	400
Rialto	4	512, 512, 256, 32	0.2	200
InsAbr	5	128, 64, 32, 16, 8	0.1	200
InsInc	5	128, 64, 32, 16, 8	0.1	100
InsIncAbr	3	32, 16, 8	0.1	50
InsIncReo	3	128, 64, 32	0.1	400

Table C.3.: Sensitivity values for ADWIN detector.

Gas	Electricity	Rialto	InsAbr	InsInc	InsIncAbr	InsIncReo
0.1	1e-15	1e-20	0.002	0.002	0.1	0.1