Marwin Otto Züfle

Proactive Critical Event Prediction based on Monitoring Data with Focus on Technical Systems

Dissertation, Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik, 2021
Gutachter: Prof. Dr. Samuel Kounev, Julius-Maximilians-Universität Würzburg
Prof. Dr. Bernhard Sick, Universität Kassel

Datum der mündlichen Prüfung: 7. Februar 2022

Abstract

The importance of proactive and timely prediction of critical events is steadily increasing, whether in the manufacturing industry or in private life. In the past, machines in the manufacturing industry were often maintained based on a regular schedule or threshold violations, which is no longer competitive as it causes unnecessary costs and downtime. In contrast, the predictions of critical events in everyday life are often much more concealed and hardly noticeable to the private individual, unless the critical event occurs. For instance, our electricity provider has to ensure that we, as end users, are always supplied with sufficient electricity, or our favorite streaming service has to guarantee that we can watch our favorite series without interruptions. For this purpose, they have to constantly analyze what the current situation is, how it will develop in the near future, and how they have to react in order to cope with future conditions without causing power outages or video stallings.

In order to analyze the performance of a system, monitoring mechanisms are often integrated to observe characteristics that describe the workload and the state of the system and its environment. Reactive systems typically employ thresholds, utility functions, or models to determine the current state of the system. However, such reactive systems cannot proactively estimate future events, but only as they occur. In the case of critical events, reactive determination of the current system state is futile, whereas a proactive system could have predicted this event in advance and enabled timely countermeasures. To achieve proactivity, the system requires estimates of future system states. Given the gap between design time and runtime, it is typically not possible to use expert knowledge to *a priori* model all situations a system might encounter at runtime. Therefore, prediction methods must be integrated into the system. Depending on the available monitoring data and the complexity of the prediction task, either time series forecasting in combination with thresholding or more sophisticated machine and deep learning models have to be trained.

Although numerous forecasting methods have been proposed in the literature, these methods have their advantages and disadvantages depending on the characteristics of the time series under consideration. Therefore, expert knowledge is required to decide which forecasting method to choose. However, since the time series observed at runtime cannot be known at design time,

such expert knowledge cannot be implemented in the system. In addition to selecting an appropriate forecasting method, several time series preprocessing steps are required to achieve satisfactory forecasting accuracy. In the literature, this preprocessing is often done manually, which is not practical for autonomous computing systems, such as Self-Aware Computing Systems. Several approaches have also been presented in the literature for predicting critical events based on multivariate monitoring data using machine and deep learning. However, these approaches are typically highly domain-specific, such as financial failures, bearing failures, or product failures. Therefore, they require in-depth expert knowledge. For this reason, these approaches cannot be fully automated and are not transferable to other use cases. Thus, the literature lacks generalizable end-to-end workflows for modeling, detecting, and predicting failures that require only little expert knowledge.

To overcome these shortcomings, this thesis presents a system model for meta-self-aware prediction of critical events based on the LRA-M loop of Self-Aware Computing Systems. Building upon this system model, this thesis provides six further contributions to critical event prediction. While the first two contributions address critical event prediction based on univariate data via time series forecasting, the three subsequent contributions address critical event prediction for multivariate monitoring data using machine and deep learning algorithms. Finally, the last contribution addresses the update procedure of the system model. Specifically, the seven main contributions of this thesis can be summarized as follows:

- First, we present a system model for meta self-aware prediction of critical events. To handle both univariate and multivariate monitoring data, it offers univariate time series forecasting for use cases where a single observed variable is representative of the state of the system, and machine learning algorithms combined with various preprocessing techniques for use cases where a large number of variables are observed to characterize the system's state. However, the two different modeling alternatives are not disjoint, as univariate time series forecasts can also be included to estimate future monitoring data as additional input to the machine learning models. Finally, a feedback loop is incorporated to monitor the achieved prediction quality and trigger model updates.

- We propose a novel hybrid time series forecasting method for univariate, seasonal time series, called Telescope. To this end, Telescope automatically preprocesses the time series, performs a kind of divide-and-conquer technique to split the time series into multiple components, and derives

additional categorical information. It then forecasts the components and categorical information separately using a specific state-of-the-art method for each component. Finally, Telescope recombines the individual predictions. As Telescope performs both preprocessing and forecasting automatically, it represents a complete end-to-end approach to univariate seasonal time series forecasting.

Experimental results show that Telescope achieves enhanced forecast accuracy, more reliable forecasts, and a substantial speedup. Furthermore, we apply Telescope to the scenario of predicting critical events for virtual machine auto-scaling. Here, results show that Telescope considerably reduces the average response time and significantly reduces the number of service level objective violations.

- For the automatic selection of a suitable forecasting method, we introduce two frameworks for recommending forecasting methods. The first framework extracts various time series characteristics to learn the relationship between them and forecast accuracy. In contrast, the other framework divides the historical observations into internal training and validation parts to estimate the most appropriate forecasting method. Moreover, this framework also includes time series preprocessing steps.

  Comparisons between the proposed forecasting method recommendation frameworks and the individual state-of-the-art forecasting methods and the state-of-the-art forecasting method recommendation approach show that the proposed frameworks considerably improve the forecast accuracy.

- With regard to multivariate monitoring data, we first present an end-to-end workflow to detect critical events in technical systems in the form of anomalous machine states. The end-to-end design includes raw data processing, phase segmentation, data resampling, feature extraction, and machine tool anomaly detection. In addition, the workflow does not rely on profound domain knowledge or specific monitoring variables, but merely assumes standard machine monitoring data.

  We evaluate the end-to-end workflow using data from a real CNC machine. The results indicate that conventional frequency analysis does not detect the critical machine conditions well, while our workflow detects the critical events very well with an F1-score of almost 91%.

- To predict critical events rather than merely detecting them, we compare different modeling alternatives for critical event prediction in the use case of time-to-failure prediction of hard disk drives. Given that failure

records are typically significantly less frequent than instances representing the normal state, we employ different oversampling strategies. Next, we compare the prediction quality of binary class modeling with downscaled multi-class modeling. Furthermore, we integrate univariate time series forecasting into the feature generation process to estimate future monitoring data. Finally, we model the time-to-failure using not only classification models but also regression models.

The results suggest that multi-class modeling provides the overall best prediction quality with respect to practical requirements. In addition, we prove that forecasting the features of the prediction model significantly improves the critical event prediction quality.

- We propose an end-to-end workflow for predicting critical events of industrial machines. Again, this approach does not rely on expert knowledge except for the definition of monitoring data, and therefore represents a generalizable workflow for predicting critical events of industrial machines. The workflow includes feature extraction, feature handling, target class mapping, and model learning with integrated hyperparameter tuning via a grid-search technique. Drawing on the result of the previous contribution, the workflow models the time-to-failure prediction in terms of multiple classes, where we compare different labeling strategies for multi-class classification.

  The evaluation using real-world production data of an industrial press demonstrates that the workflow is capable of predicting six different time-to-failure windows with a macro F1-score of 90%. When scaling the time-to-failure classes down to a binary prediction of critical events, the F1-score increases to above 98%.

- Finally, we present four update triggers to assess when critical event prediction models should be re-trained during on-line application. Such re-training is required, for instance, due to concept drift. The update triggers introduced in this thesis take into account the elapsed time since the last update, the prediction quality achieved on the current test data, and the prediction quality achieved on the preceding test data.

  We compare the different update strategies with each other and with the static baseline model. The results demonstrate the necessity of model updates during on-line application and suggest that the update triggers that consider both the prediction quality of the current and preceding test data achieve the best trade-off between prediction quality and number of updates required.

We are convinced that the contributions of this thesis constitute significant impulses for the academic research community as well as for practitioners. First of all, to the best of our knowledge, we are the first to propose a fully automated, end-to-end, hybrid, component-based forecasting method for seasonal time series that also includes time series preprocessing. Due to the combination of reliably high forecast accuracy and reliably low time-to-result, it offers many new opportunities in applications requiring accurate forecasts within a fixed time period in order to take timely countermeasures. In addition, the promising results of the forecasting method recommendation systems provide new opportunities to enhance forecasting performance for all types of time series, not just seasonal ones. Furthermore, we are the first to expose the deficiencies of the prior state-of-the-art forecasting method recommendation system.

Concerning the contributions to critical event prediction based on multivariate monitoring data, we have already collaborated closely with industrial partners, which supports the practical relevance of the contributions of this thesis. The automated end-to-end design of the proposed workflows that do not demand profound domain or expert knowledge represents a milestone in bridging the gap between academic theory and industrial application. Finally, the workflow for predicting critical events in industrial machines is currently being operationalized in a real production system, underscoring the practical impact of this thesis.

Zusammenfassung

Die Bedeutung einer proaktiven und rechtzeitigen Vorhersage von kritischen Ereignissen nimmt immer weiter zu, sei es in der Fertigungsindustrie oder im Privatleben. In der Vergangenheit wurden Maschinen in der Fertigungsindustrie oft auf der Grundlage eines regelmäßigen Zeitplans oder aufgrund von Grenzwertverletzungen gewartet, was heutzutage nicht mehr wettbewerbsfähig ist, da es unnötige Kosten und Ausfallzeiten verursacht. Im Gegensatz dazu sind die Vorhersagen von kritischen Ereignissen im Alltag oft wesentlich versteckter und für die Privatperson kaum spürbar, es sei denn das kritische Ereignis tritt ein. So muss zum Beispiel unser Stromanbieter dafür sorgen, dass wir als Endverbraucher immer ausreichend mit Strom versorgt werden, oder unser Streaming-Dienst muss garantieren, dass wir unsere Lieblingsserie jederzeit ohne Unterbrechungen anschauen können. Hierzu müssen diese ständig analysieren wie der aktuelle Zustand ist, wie er sich in naher Zukunft entwickeln wird und wie sie reagieren müssen, um die zukünftigen Bedingungen zu bewältigen, ohne dass es zu Stromausfällen oder Videoabbrüchen kommt.

Zur Analyse der Leistung eines Systems werden häufig Überwachungsmechanismen integriert, um Merkmale zu beobachten, die die Arbeitslast und den Zustand des Systems und seiner Umgebung abbilden. Reaktive Systeme verwenden typischerweise Schwellenwerte, Nutzenfunktionen oder Modelle, um den aktuellen Zustand des Systems zu bestimmen. Allerdings können solche reaktiven Systeme zukünftige Ereignisse nicht proaktiv abschätzen, sondern lediglich sobald diese eintreten. Bei kritischen Ereignissen ist die reaktive Bestimmung des aktuellen Systemzustands jedoch zwecklos, während ein proaktives System dieses Ereignis im Voraus hätte vorhersagen und rechtzeitig Gegenmaßnahmen einleiten können. Um Proaktivität zu erreichen, benötigt das System Abschätzungen über zukünftige Systemzustände. Angesichts der Kluft zwischen Entwurfszeit und Laufzeit ist es typischerweise nicht möglich Expertenwissen zu verwenden, um alle Situationen zu modellieren, auf die ein System zur Laufzeit stoßen könnte. Daher müssen Vorhersagemethoden in das System integriert werden. Abhängig von den verfügbaren Überwachungsdaten und der Komplexität der Vorhersageaufgabe müssen entweder Zeitreihenprognosen in Kombination mit Schwellenwerten oder ausgefeiltere Modelle des „Machine Learning“ und „Deep Learning“ trainiert werden.

Obwohl in der Literatur schon zahlreiche Zeitreihenprognosemethoden vorgeschlagen wurden, haben alle diese Methoden in Abhängigkeit der Eigenschaften der betrachteten Zeitreihen ihre Vor- und Nachteile. Daher ist Expertenwissen erforderlich, um zu entscheiden, welche Zeitreihenprognosemethode gewählt werden sollte. Da jedoch die zur Laufzeit beobachteten Zeitreihen zur Entwurfszeit nicht bekannt sein können, lässt sich ein solches Expertenwissen nicht im System integrieren. Zusätzlich zur Auswahl einer geeigneten Zeitreihenprognosemethode sind mehrere Zeitreihenvorverarbeitungsschritte erforderlich, um eine zufriedenstellende Prognosegenauigkeit zu erreichen. In der Literatur wird diese Vorverarbeitung oft manuell durchgeführt, was für autonome Computersysteme, wie z. B. „Self-Aware Computing Systems", nicht praktikabel ist. Hinsichtlich der Vorhersage kritischer Ereignisse auf der Grundlage multivariater Überwachungsdaten wurden in der Literatur auch bereits mehrere Ansätze unter Verwendung von „Machine Learning" und „Deep Learning" vorgestellt. Diese Ansätze sind jedoch typischerweise sehr domänenspezifisch, wie z. B. für finanzielle Zusammenbrüche, Lagerschäden oder Produktfehler. Aus diesem Grund erfordern sie umfassendes Expertenwissen. Durch den spezifischen Zuschnitt auf die jeweilige Domäne können diese Ansätze nicht vollständig automatisiert werden und sind nicht auf andere Anwendungsfälle übertragbar. Somit fehlt es in der Literatur an verallgemeinerbaren Ende-zu-Ende Prozessen zur Modellierung, Erkennung und Vorhersage von Ausfällen, die lediglich wenig Expertenwissen erfordern.

Um diese Unzulänglichkeiten zu überwinden, wird in dieser Arbeit ein Systemmodell zur meta-selbstbewussten Vorhersage kritischer Ereignisse vorgestellt, das auf der LRA-M-Schleife von „Self-Aware Computing Systems" basiert. Aufbauend auf diesem Systemmodell liefert diese Arbeit sechs weitere Beiträge zur Vorhersage kritischer Ereignisse. Während sich die ersten beiden Beiträge mit der Vorhersage kritischer Ereignisse auf der Basis univariater Daten mittels Zeitreihenprognose befassen, adressieren die drei folgenden Beiträge die Vorhersage kritischer Ereignisse für multivariate Überwachungsdaten unter Verwendung von „Machine Learning" und „Deep Learning" Algorithmen. Der letzte Beitrag schließlich behandelt das Aktualisierungsverfahren des Systemmodells. Im Einzelnen lassen sich die sieben Hauptbeiträge dieser Arbeit wie folgt zusammenfassen:

- Zunächst stellen wir ein Systemmodell für die meta-selbstbewusste Vorhersage von kritischen Ereignissen vor. Um sowohl univariate als auch multivariate Überwachungsdaten verarbeiten zu können, bietet es univariate Zeitreihenprognosen für Anwendungsfälle, in denen eine einzelne Beobachtungsgröße repräsentativ für den Zustand des Systems ist, sowie

„Machine Learning“ und „Deep Learning“ Algorithmen in Kombination mit verschiedenen Vorverarbeitungstechniken für Anwendungsfälle, in denen eine große Anzahl von Variablen beobachtet wird, um den Zustand des Systems zu charakterisieren. Die beiden unterschiedlichen Modellierungsalternativen sind jedoch nicht disjunkt, da auch univariate Zeitreihenprognosen einbezogen werden können, um zukünftige Überwachungsdaten als zusätzliche Eingabe für die „Machine Learning“ und „Deep Learning“ Modelle zu schätzen. Schließlich ist eine Rückkopplungsschleife eingebaut, die die erreichte Vorhersagequalität überwacht und gegebenenfalls Modellaktualisierungen auslöst.

- Wir präsentieren eine neuartige, hybride Zeitreihenvorhersagemethode für univariate, saisonale Zeitreihen, die wir Telescope nennen. Telescope verarbeitet die Zeitreihe automatisch vor, führt eine Art „Divide-and-Conquer“ Technik durch, welche die Zeitreihe in mehrere Komponenten unterteilt, und leitet zusätzliche kategoriale Informationen ab. Anschließend prognostiziert es die Komponenten und kategorialen Informationen getrennt voneinander mit einer spezifischen Methode für jede Komponente. Abschließend setzt Telescope die einzelnen Vorhersagen wieder zusammen. Da Telescope alle Vorverarbeitungsschritte und Vorhersagen automatisch durchführt, stellt es einen vollständigen Ende-zu-Ende Ansatz für univariate, saisonale Zeitreihenvorhersagen dar.

  Experimentelle Ergebnisse zeigen, dass Telescope eine verbesserte Vorhersagegenauigkeit, zuverlässigere Vorhersagen und eine erhebliche Beschleunigung erreicht. Darüber hinaus wenden wir Telescope für die Vorhersage kritischer Ereignisse bei der automatischen Skalierung von virtuellen Maschinen an. Die Ergebnisse belegen, dass Telescope die durchschnittliche Antwortzeit erheblich reduziert und die Anzahl der Verletzungen der Service Level Zielvorgaben signifikant verringert.

- Für die automatische Auswahl einer geeigneten Zeitreihenprognosemethode führen wir zwei Empfehlungssysteme ein. Das erste System extrahiert verschiedene Zeitreihencharakteristika, um die Beziehung zwischen ihnen und der Prognosegenauigkeit zu erlernen. Im Gegensatz dazu unterteilt das zweite System die historischen Beobachtungen in interne Trainings- und Validierungsteile, um die am besten geeignete Zeitreihenprognosemethode zu schätzen. Außerdem beinhaltet letzteres System auch Zeitreihenvorverarbeitungsschritte.

  Vergleiche zwischen den vorgeschlagenen Empfehlungssystemen für Zeitreihenprognosemethoden und den einzelnen Prognosemethoden

sowie dem Ansatz zur Empfehlung von Zeitreihenprognosemethoden nach dem Stand der Technik ergeben, dass die vorgeschlagenen Systeme die Prognosegenauigkeit erheblich verbessern.

- Im Hinblick auf multivariate Überwachungsdaten stellen wir zunächst einen Ende-zu-Ende Prozess vor, mit dem kritische Ereignisse in technischen Systemen in Form von anomalen Maschinenzuständen erkannt werden können. Der Ende-zu-Ende Entwurf umfasst die Rohdatenverarbeitung, die Phasensegmentierung, das Datenresampling, die Merkmalsextraktion und die Maschinenanomalieerkennung. Darüber hinaus stützt sich der Prozess explizit nicht auf tiefgreifendes Domänenwissen oder spezifische Überwachungsgrößen, sondern setzt lediglich gängige Maschinenüberwachungsdaten voraus.

  Wir evaluieren den Ende-zu-Ende Prozess anhand von Daten einer realen CNC-Maschine. Die Ergebnisse zeigen, dass die konventionelle Frequenzanalyse die kritischen Maschinenzustände nicht gut erkennt, während unser Prozess die kritischen Ereignisse mit einem F1-Wert von fast 91% sehr gut identifiziert.

- Um kritische Ereignisse vorherzusagen, anstatt sie nur reaktiv zu erkennen, vergleichen wir verschiedene Modellierungsalternativen für die Vorhersage kritischer Ereignisse im Anwendungsfall der Vorhersage der Zeit bis zum nächsten Fehler von Festplattenlaufwerken. Da Fehlerdatensätze typischerweise wesentlich seltener sind als Instanzen, die den Normalzustand repräsentieren, setzen wir verschiedene Strategien zum Erzeugen künstlicher Fehlerinstanzen ein. Im nächsten Schritt vergleichen wir die Vorhersagequalität der binären Klassenmodellierung mit der herunterskalierten Mehrklassenmodellierung. Des Weiteren integrieren wir die univariate Zeitreihenprognose in den Merkmalsgenerierungsprozess, um so die zukünftigen Überwachungsdaten zu schätzen. Schließlich modellieren wir die Zeit bis zum nächsten Fehler nicht nur mithilfe von Klassifikationsmodellen, sondern auch mit Regressionsmodellen.

  Die Ergebnisse legen nahe, dass die Mehrklassenmodellierung die insgesamt beste Vorhersagequalität hinsichtlich praktischer Anforderungen liefert. Außerdem belegen wir, dass die Prognose der Merkmale des Vorhersagemodells mittels univariater Zeitreihenprognose die Qualität der Vorhersage kritischer Ereignisse signifikant verbessert.

- Wir stellen einen Ende-zu-Ende Prozess für die Vorhersage kritischer Ereignisse von Industriemaschinen vor. Auch dieser Ansatz verlässt sich

nicht auf Expertenwissen, mit Ausnahme der Definition von Überwachungsdaten, und stellt daher einen verallgemeinerbaren Prozess für die Vorhersage kritischer Ereignisse von Industriemaschinen dar. Der Prozess umfasst Merkmalsextraktion, Merkmalsverarbeitung, Zielklassenzuordnung und Modelllernen mit integrierter Hyperparameter-Abstimmung mittels einer Gittersuchtechnik. Ausgehend von den Ergebnissen des vorherigen Beitrags modelliert der Prozess die Vorhersage der Zeit bis zum nächsten Fehler in Form mehrerer Klassen, wobei wir verschiedene Beschriftungsstrategien für die Mehrklassenklassifizierung vergleichen.

Die Evaluierung anhand realer Produktionsdaten einer großen Industriepresse demonstriert, dass der Prozess in der Lage ist, sechs verschiedene Zeitfenster für bevorstehende Fehler mit einem Makro F1-Wert von 90% vorherzusagen. Wenn man die Klassen der Zeit bis zum nächsten Fehler auf eine binäre Vorhersage von kritischen Ereignissen herunterskaliert, steigt der F1-Wert sogar auf über 98%.

- Schließlich stellen wir vier Aktualisierungsauslöser vor, um zu bestimmen, wann Modelle zur Vorhersage kritischer Ereignisse während der Online-Anwendung neu trainiert werden sollten. Ein solches Neutraining ist bspw. aufgrund von Konzeptdrift erforderlich. Die in dieser Arbeit vorgestellten Aktualisierungsauslöser berücksichtigen die Zeit, die seit der letzten Aktualisierung verstrichen ist, die auf den aktuellen Testdaten erreichte Vorhersagequalität und die auf den vorangegangenen Testdaten erreichte Vorhersagequalität.

  Wir vergleichen die verschiedenen Aktualisierungsstrategien miteinander und mit dem statischen Ausgangsmodell. Die Ergebnisse veranschaulichen die Notwendigkeit von Modellaktualisierungen während der Online-Anwendung und legen nahe, dass die Aktualisierungsauslöser, die sowohl die Vorhersagequalität der aktuellen als auch der vorangegangenen Testdaten berücksichtigen, den besten Kompromiss zwischen Vorhersagequalität und Anzahl der erforderlichen Aktualisierungen erzielen.

Wir sind der festen Überzeugung, dass die Beiträge dieser Arbeit sowohl für die akademische Forschungsgemeinschaft als auch für die praktische Anwendung wichtige Impulse darstellen. Zuallererst sind wir unseres Wissens nach die ersten, die eine vollautomatische, hybride, komponentenbasierte, Ende-zu-Ende Prognosemethode für saisonale Zeitreihen vorschlagen, die auch die Zeitreihenvorverarbeitung beinhaltet. Durch die Verbindung einer zuverlässig hohen Vorhersagegenauigkeit mit einer zuverlässig niedrigen Zeit bis zum

Ergebnis eröffnet diese viele neue Möglichkeiten für Anwendungen, die genaue Vorhersagen innerhalb eines festen Zeitraums erfordern, um rechtzeitig Gegenmaßnahmen ergreifen zu können. Darüber hinaus bieten die vielversprechenden Ergebnisse der Empfehlungssysteme für Zeitreihenprognosemethoden neue Ansätze zur Verbesserung der Vorhersageleistung für alle Arten von Zeitreihen, nicht nur für saisonale Zeitreihen. Ferner sind wir die ersten, die die Schwachstellen des bisherigen Stands der Technik bei der Empfehlung von Zeitreihenprognosemethoden aufgedeckt haben.

Hinsichtlich der Beiträge zur Vorhersage kritischer Ereignisse mittels multivariater Überwachungsdaten haben wir bereits eng mit Industriepartnern zusammengearbeitet, wodurch die hohe praktische Relevanz der Beiträge dieser Arbeit verdeutlicht wird. Der automatisierte Ende-zu-Ende Entwurf der vorgeschlagenen Prozesse, die kein tiefes Domänen- oder Expertenwissen erfordern, stellt einen Meilenstein in der Überbrückung der Kluft zwischen akademischer Theorie und industrieller Anwendung dar. Diese Tatsache wird insbesondere dadurch untermauert, dass der Prozess zur Vorhersage kritischer Ereignisse in Industriemaschinen derzeit bereits in einem realen Produktionssystem operationalisiert wird.

Acknowledgments

This thesis would have been impossible without the help and support of many people. Foremost, I would like to thank my advisor Prof. Dr. Samuel Kounev. I first met him during my studies at the University of Würzburg, where I attended all his courses. Especially since my master's thesis, he has always supported me with advice and encouragement on my journey in the academic world.

From the the Chair of Software Engineering at the University of Würzburg, I would like to thank all my current and former colleagues, especially Dr. André Bauer, Lukas Beierlieb, Vanessa Borst, Simon Eismann, Johannes Grohmann, Stefan Herrnleben, Dr. Lukas Iffländer, Dennis Kaiser, Dr. Jóakim von Kistowski, Jun.-Prof. Dr. Christian Krupitzer, Robert Leppich, Veronika Lesch, Thomas Prantl, Norbert Schmitt, and Martin Sträßer, for the collaboration and support during the last couple of years. In particular, I want to thank Jun.-Prof. Dr. Christian Krupitzer for his supervision over several years. His expertise in scientific writing helped me a lot to improve my work. Furthermore, I would like to express special thanks to my colleagues from the "Kaffeekränzchen" group for their moral support, especially during the rough times. I also want to thank Fritz Kleemann, Susanne Stenglin, and Erika Littmann, who always helped me out with technical and administrative tasks.

In addition, I would like to thank Florian Erhard and Joachim Agne, who supported parts of my research as research assistant, supervised student, and co-authors. In general, I want to thank all my co-authors for many fertile conversations that led to new ideas, and for their help in getting the hard work published in scientific conferences and journals. Special thanks go to Valentin Curtef from the COSMO CONSULT Data Science GmbH for many interesting discussions and ideas around the topic of time series analysis and forecasting.

I would also like to thank the IHK Würzburg-Schweinfurt for the financial support and the stage for presenting our ideas, which enabled us to get in touch with industrial partners. To these industrial partners, especially ZF Friedrichshafen AG and Bosch Rexroth AG, I would also like to express my gratitude for their collaboration, without which the evaluation of the proposed approaches would not have been possible.

Finally, I would like to thank the people in my personal life who have supported me throughout my studies and research. The utmost gratitude goes to

my wife Dorothée, who has always supported me, believed in me, and had my back in my personal life so that I could focus on this thesis. Her continuous support, encouragement, and trust in me have made this thesis possible. Furthermore, I would like to thank my parents-in-law, Dr. Beate Kawaler-Hermann and Dr. Wilhelm Hermann, and my sister-in-law, Jessica Hermann, with her boyfriend, Markus Wirth, for their unconditional support during all phases.

Contents

Publication List

Peer Reviewed Journal Articles

[ZML+21] Marwin Züfle, Felix Moog, Veronika Lesch, Christian Krupitzer, Samuel Kounev. A Machine Learning-based Workflow for Automatic Detection of Anomalies in Machine Tools. In: *ISA Transactions*. Elsevier, 2021. (in press)

[BZH+20b] André Bauer, Marwin Züfle, Nikolas Herbst, Albin Zehe, Andreas Hotho, and Samuel Kounev. Time Series Forecasting for Self-Aware Systems. In: *Proceedings of the IEEE*, 108(7):1068 – 1093, 2020.

Peer Reviewed International Full Conference Papers

[ZEK21] Marwin Züfle, Florian Erhard, and Samuel Kounev. Machine Learning Model Update Strategies for Hard Disk Drive Failure Prediction. In: *Proceedings of the 20th IEEE International Conference on Machine Learning and Applications* (*ICMLA*), pages 1379–1386. IEEE, December 2021.

[ZAG+21] Marwin Züfle, Joachim Agne, Johannes Grohmann, Ibrahim Dörtoluk, and Samuel Kounev. A Predictive Maintenance Methodology: Predicting the Time-to-Failure of Machines in Industry 4.0. In: *Proceedings of the 219th IEEE IES International Conference on Industrial Informatics* (*INDIN*), pages 1–8. IEEE, July 2021.

[FML+21] Nils Finke, Marisa Mohr, Alexander Lontke, Marwin Züfle, Samuel Kounev, and Ralf Möller. Recommendations for Data-driven Degradation Estimation with Case Studies from Manufacturing and Dry-bulk Shipping. In: *Proceedings of the 15th International Conference on Research Challenges in Information Science* (*RCIS*), pages 189—204. Springer, May 2021.

[BZE+21] André Bauer, Marwin Züfle, Simon Eismann, Johannes Grohmann, Nikolas Herbst, and Samuel Kounev. Libra: A Benchmark for Time Series Forecasting Methods. In: *Proceedings of the 12th ACM/SPEC International Conference*

on Performance Engineering (*ICPE*), pages 189–200. ACM, April 2021.

[ZKE+20] Marwin Züfle, Christian Krupitzer, Florian Erhard, Johannes Grohmann, and Samuel Kounev. To Fail or Not to Fail: Predicting Hard Disk Drive Failure Time Windows. In: *Proceedings of the International Conference on Measurement, Modelling and Evaluation of Computing Systems* (*MMB*), pages 19–36. Springer, March 2020.

[ZBL+19] Marwin Züfle, André Bauer, Veronika Lesch, Christian Krupitzer, Nikolas Herbst, Samuel Kounev, and Valentin Curtef. Autonomic Forecasting Method Selection: Examination and Ways Ahead. In: *Proceedings of the 16th IEEE International Conference on Autonomic Computing* (*ICAC*), pages 167–176. IEEE, June 2019.

Peer Reviewed International Short Conference Papers

[ZK20] Marwin Züfle and Samuel Kounev. A Framework for Time Series Preprocessing and History-based Forecasting Method Recommendation. In: *Proceedings of the 2020 Federated Conference on Computer Science and Information Systems* (*FedCSIS*), pages 141–144. IEEE, September 2020.

[BZH+20a] André Bauer, Marwin Züfle, Nikolas Herbst, Samuel Kounev, and Valentin Curtef. Telescope: An Automatic Feature Extraction and Transformation Approach for Time Series Forecasting on a Level-Playing Field. In: *Proceedings of the 36th International Conference on Data Engineering* (*ICDE*), pages 1902–1905. IEEE, April 2020.

[BZG+20] André Bauer, Marwin Züfle, Johannes Grohmann, Norbert Schmitt, Nikolas Herbst, and Samuel Kounev. An Automated Forecasting Framework based on Method Recommendation for Seasonal Time Series. In: *Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering* (*ICPE*), pages 48–55. ACM, April 2020.

[ZBH+17] Marwin Züfle, André Bauer, Nikolas Herbst, Valentin Curtef, and Samuel Kounev. Telescope: A Hybrid Forecast Method for Univariate Time Series. In: *Proceedings of the International Work-Conference on Time Series Analysis* (*ITISE*). September 2017.

Peer Reviewed Workshop and Tutorial Papers

[HZZ+20] Stefan Herrnleben, Bernd Zeidler, Marwin Züfle, Christian Krupitzer, and Samuel Kounev. A Concept for Crowd-sensed Prediction of Mobile Network Connectivity. In: *GI/ITG Workshop on Machine Learning in the Context of Communication Networks 2020*. February 2020.

[GEB+19] Johannes Grohmann, Simon Eismann, André Bauer, Marwin Züfle, Nikolas Herbst, and Samuel Kounev. Utilizing Clustering to Optimize Resource Demand Estimation Approaches. In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 134-–139. IEEE, June 2019.

[BZHK19] André Bauer, Marwin Züfle, Nikolas Herbst, and Samuel Kounev. Best Practices for Time Series Forecasting. In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 255–256. IEEE, June 2019.

Book Chapters

[Züf20] Marwin Züfle. Towards a Self-Aware Prediction of Critical States. In: *Organic Computing: Doctoral Dissertation Colloquium 2020*. Edited by Sven Tomforde and Christian Krupitzer. July 2020.

Technical Reports

[KWZ+20] Christian Krupitzer, Tim Wagenhals, Marwin Züfle, Veronika Lesch, Dominik Schäfer, Amin Mozaffarin, Janick Edinger, Christian Becker, and Samuel Kounev. *A Survey on Predictive Maintenance for Industry 4.0*. Technical report, University of Würzburg and University of Mannheim and Syntax Systems GmbH and MOZYS Engineering GmbH. arXiv:2002.08224, February 2020.

[KML+20] Christian Krupitzer, Sebastian Müller, Veronika Lesch, Marwin Züfle, Janick Edinger, Alexander Lemken, Dominik Schäfer, Samuel Kounev, and Christian Becker. *A Survey on Human Machine Interaction in Industry 4.0*. Technical report, University of Würzburg and University of Mannheim and ioxp GmbH and Syntax Systems GmbH. arXiv:2002.01025, February 2020.

Chapter 1

Introduction

The introduction of this thesis aims at motivating the addressed research field, elaborating the shortcomings of existing approaches, and presenting the contributions of this thesis. Therefore, we first describe the particular research field and highlight its importance in everyday life. Then, we briefly present the state-of-the-art and identify drawbacks and shortcomings of existing approaches. Subsequently, we introduce the research questions that form the foundations of this thesis. Based on these, we also derive our main contributions. Before presenting the outline of the remaining thesis, we additionally describe several potential use cases for the introduced system model.

1.1 Motivation

Although hardly noticed, *critical event prediction* plays an immense role in the daily lives of most people. When hearing the term critical event prediction, most people may think of severe natural disasters or the outbreak of a new virus mutation. However, critical events are all kinds of situations that may cause *malfunctions* of the observed *system* or its *environment*. This, in turn, leads directly into the area of *failure prediction* of *technical systems*, such as predicting the time-to-failure of a hard disk drive in a large cloud system [IBM16, Eme16] or *predictive maintenance* applications in the field of *Industry 4.0* [Sta18, MH18]. Such technical systems are often complex and, therefore, monitored with numerous sensors. These multivariate monitoring data describe the current state of the technical system, so that the entire amount of data must be analyzed in order to predict impending critical events.

Apart from these technical systems, critical events also occur in our everyday lives. Here, however, the critical events are often characterized by a single variable. For instance, our electricity provider must analyze the electricity demand to adjust to fluctuations in the electricity demand in order to provide us with enough electricity at all times, for instance to watch television or to use our computer [Tay03, KYO]. This leads us directly to other potentially critical

situations, since the web pages we browse must also process our requests. To this end, the arrival rate of requests must be monitored to decide when to scale up and add more instances to handle the increased traffic, or when to scale down as the workload decreases to save money [JYJ13, LJ+18]. Another critical event can be seen in the domain of supermarkets, namely the amount of products, such as fresh vegetables and fruits, that must be ordered to meet the customer demand as closely as possible [Små21, REL20]. Ordering too much leads to the critical event of food going bad while ordering too little leads to the critical event of poor customer satisfaction. A parallel can be drawn to the planning of flights and tourist visits, where airlines and other tourist agencies must adjust their capacity in time to handle the demand [WM18, Cen20]. Furthermore, there are also critical events in the stock market. Here, the customer may wonder how the price of a stock will develop and whether it is better to sell now, because the price will crash critically afterwards, or whether it would be better to hold the stock? A recent example on this topic is the meteoric rise of GameStop shares [PL21, Med21]. Here, it was obvious that this high could not last long, but the critical event of the stock crash was still difficult to predict. Another highly topical subject for critical event prediction can be seen in the development of the SARS-CoV-2 incidence value [Fuc20, Rei21]. In this context, critical events would be represented by the thresholds set by the government, which if exceeded or undercut would lead to harsher or more lenient countermeasures, respectively.

In order to not only reactively resolve such critical events, but also to be able to initiate countermeasures at an early stage that may even prevent the critical event, it is essential to predict these critical events at an early point in time.

1.2 Problem Statement and Shortcomings of Existing Approaches

Predicting critical events is one of the most important tasks in any system to handle changes and plan countermeasures. Therefore, systems often implement monitoring mechanisms to observe characteristics that describe the workload and the state of the system and its environment. For this purpose, thresholds, utility functions, or models are typically employed to infer the current state of the system [KRV+15]. However, due to digitalization and increased computing power, these systems can nowadays collect much more information and store it for future analysis. Thus, more advanced data-driven models can be deployed to enable the integration of proactive adaptations [KRV+15, Wey17]. The advantage of such proactive systems over typical systems based on reactive

adaptation is that delays in the adaptation process can be eliminated. Moreover, in the case of critical events, reactive identification of the current system state is futile, whereas a proactive system could have predicted this event in advance and allowed timely countermeasures.

To make a reactive system proactive, the system must have estimates of future states. In most cases, however, *a priori* modeling of all the situations a system might encounter at runtime is not possible. Therefore, merely integrating expert knowledge at design time is not sufficient. That is why prediction methods have to be integrated into the system. Based on the available data and the complexity of the prediction task, either time series forecasting combined with thresholding can be used to achieve proactivity, or more sophisticated machine and deep learning models need to be trained to provide early predictions while keeping the required domain knowledge as small as possible.

In the case of representative univariate monitoring data, such as the utilization of a server, the electricity demand of a particular area, or the number of airline passengers, time series forecasting methods can be applied to obtain estimates of future observations of this univariate data generation process. However, for critical event prediction, domain knowledge is required to define thresholds for the univariate data, such as the maximum utilization a server should not exceed or the maximum number of airline passengers that can be served per month. If the forecast indicates an exceedance or undercut of the defined threshold, the critical event is predicted [HZS99, LWP12, ZG15].

In contrast, larger and more complex systems cannot be characterized by univariate data. Instead, numerous properties are monitored for such systems, resulting in multivariate data. Here, thresholds are no longer applicable, requiring more sophisticated models to be learned. Yet, simply training machine and deep learning predictors is not always directly possible. First, a meaningful target must be defined that represents the critical event with respect to the system under consideration. As monitoring applications are often error-prone, missing data or anomalous values in the records often occur. However, most prediction methods cannot handle missing values, so these gaps must be reconstructed. Furthermore, anomalous values can distort the learning process, which is the reason these must also be removed. Next, the multivariate monitoring data may not contain useful information in its raw format, requiring the system to preprocess the raw data to create meaningful features. Moreover, more features do not necessarily improve the model, but may actually worsen the model performance by overfitting to irrelevant features. Therefore, only the most relevant features should be selected to reduce the time required to learn the model and preprocess the raw data and to avoid distortion of the

model. In addition, a crucial task is the selection of an appropriate prediction method and the optimization of its hyperparameters. Finally, the system must update the initial, off-line learned model during runtime to adapt to changes in the system and the environment. For this purpose, different strategies can be applied to achieve a trade-off between computation time and model accuracy.

In this thesis, we provide contributions to both approaches to enable proactivity for adaptive systems. First, we present novel approaches on time series forecasting and evaluate them on a broad set of time series. Although numerous forecasting methods have been proposed in the literature, these methods have their advantages and drawbacks depending on the properties of the time series under consideration [WM97]. We tackle this problem by combining several forecasting methods, leveraging their strengths and shortcomings to obtain more robust results for a wide range of time series. Second, we introduce end-to-end workflows for modeling, detecting, and predicting failures in the technical domain. Given that this second category addresses much more complex systems, we have focused only on technical systems, such as computer systems and industrial machines. Although there are already several approaches to predict failures for specific applications, such as financial failures [TK92, XW09], bearing failures [LWZ+14, HRBA+18], or product failures [EW74, Ku17], these methods are tailored to the specific application and require significant expert knowledge. Therefore, these approaches cannot be automated and are not applicable to other use cases. In contrast, our approaches represent end-to-end workflows that require no or almost negligible expert knowledge. Although it is indisputable that one explicit approach cannot cover multiple use cases simultaneously, the literature lacks a generalizable system model that is integrated into an autonomous computing system architecture and provides several functionalities to cover this multitude of use cases. Therefore, we introduce such a generalizable system model for meta self-aware critical event prediction that offers a broad range of modeling options to handle multiple use cases.

1.3 Research Questions

Drawing from the problem statement and shortcomings presented in Section 1.2, we derived two major goals for this thesis. The first goal addresses the design of novel, hybrid time series forecasting methods to improve forecast accuracy compared with state-of-the-art forecasting methods. With increased forecast accuracy, the prediction of critical events in terms of threshold violations is inevitably enhanced. The second goal, in contrast, focuses on multivariate monitoring data and the development of more complex, end-to-end critical

event prediction workflows. The contributions related to **Goal A** are presented in Part II, whereas Part III provides the contributions regarding **Goal B**. In addition, we break down each goal into several research questions, which are addressed in the individual chapters of the two parts.

Goal A: *Improving the accuracy of automated time series forecasting over state-of-the-art methods via novel, hybrid forecasting methods.*

This goal is formed by the following research questions. The first and third research question address the design of novel hybrid forecasting methods, while the second and fourth research questions tackle their evaluation by comparing the proposed hybrid methods with state-of-the-art forecasting methods.

RQ A.1: How can we use time series decomposition to design a hybrid time series forecasting method?

RQ A.2: To what extent does the decomposition-based hybrid time series forecasting method outperform the state-of-the-art in individual time series forecasting methods?

RQ A.3: How can we improve the forecast accuracy by employing time series forecasting method recommendation?

RQ A.4: Compared with the state-of-the-art recommendation approach for forecasting methods and individual forecasting methods, how do our recommendation approaches perform?

Goal B: *Development of generalizable end-to-end workflows to model, detect, and predict machine failures, with minimal expert knowledge required.*

Similar to **Goal A**, we also split **Goal B** into four individual research questions. However, the research questions might also consist of several short questions.

RQ B.1: How can we infer current degradation states of industrial machines based only on standard monitoring data without additional domain knowledge?

RQ B.2: In which way can we realize a proactive component for the prediction of impending failure events of technical systems?

- **RQ B.2.1:** How can we balance the number of instances for different failure classes?

RQ B.2.2: How can we model the time-to-failure?

RQ B.2.3: To what extent does time series forecasting improve the time-to-failure prediction?

RQ B.3: How can we design an end-to-end workflow for predicting critical events of industrial machines without requiring expert knowledge?

RQ B.4: On the basis of which triggers should machine learning-based critical event prediction models be updated at runtime as new data arrives?

1.4 Contributions of this Thesis

To present the main contributions of this thesis, we first introduce our meta self-aware system model for the prediction of critical events and, subsequently, derive the key contributions with respect to the system model.

1.4.1 System Model for Meta Self-Aware Prediction of Critical Events

In order to integrate proactive analysis for predicting critical events into *Self-Aware Computing Systems* [KLB+17] (cf. Section 2.1), primarily the learning component must be adapted. Figure 1.1 illustrates our proposed system model for a meta self-aware analysis for critical event prediction using the typical *LRA-M loop* [KLB+17] (cf. Section 2.1.2). Depending on the data complexity, the system model decides whether to apply only time series forecasting and meta-learning for time series forecasting or to perform additional preprocessing steps and a more complex model training.

In the case of univariate data that have a specified threshold indicating critical events, the monitored data are passed to the time series forecasting component to model the time series and produce forecasts. In addition, a meta self-aware time series forecasting method recommendation can be trained to select an appropriate forecasting method with respect to the time series under consideration. Finally, the resulting time series forecasting model is returned as a critical event prediction model.

In contrast, for multivariate monitoring data, these time series forecasts can also be applied to add estimates of future monitoring data to the existing ones. However, other preprocessing steps, such as transformation techniques and feature extraction, are also performed on the raw monitoring data. The composition of forecasts, derived features, and raw monitoring data are then passed to the model learning component. On the basis of these inputs, a feature engineering step is performed that can combine existing features, normalize

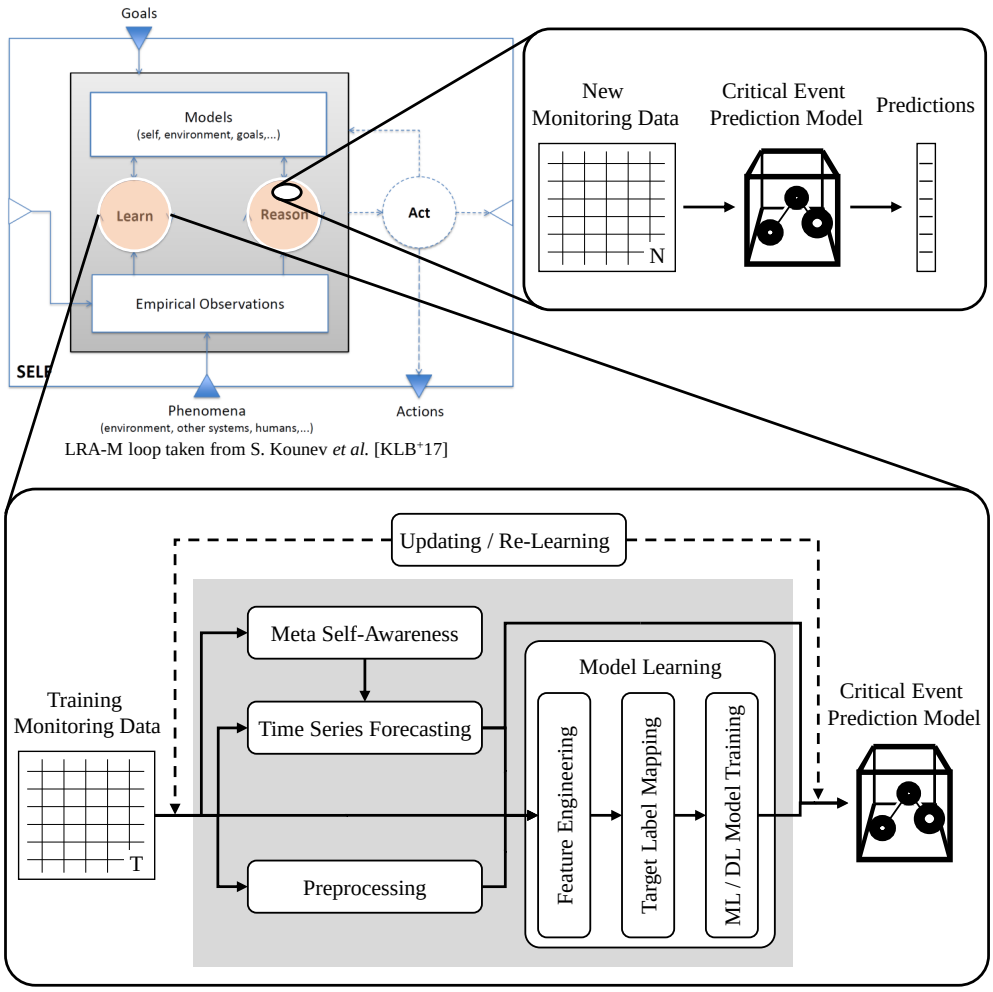


Figure 1.1: Design of the meta self-aware system model for critical event prediction.

their ranges, or select only the features most relevant to the goal with respect to a given relevance measure. Regarding the target of the model, a meaningful choice has to be done, which heavily depends on the context in which the system model is used. For the prediction of critical events, a reasonable target would be the time-to-failure. This can be modeled as either a continuous regression task or a discretized classification task. The targets must then be mapped to the respective training instances. Finally, the relationship between the features and the target is learned using machine learning (ML) or deep learning (DL) methods and the derived critical event prediction model is returned.

In order to improve the prediction of critical events, the system model includes a feedback loop that monitors the actual result and compares it with the predicted one. Thus, this component is also responsible for ongoing on-line learning. Based on a pre-defined trigger, this component initiates re-learning of the model using the previous data as well as the new data received since the last model update. This trigger could be, for instance, simple time spans (e.g., daily or weekly), deviations between expected behavior and observations (e.g., accuracy below 90%, runtime accuracy below 95% of training accuracy), or a measure to determine significant concept drift (i.e., Hoeffding bound [DH00]).

Once the critical event prediction model is derived in the learning component of the LRA-M loop, its application is performed in the reasoning component. Here, the new monitoring data arrives, which is then forwarded to the critical event prediction model, including possible preprocessing steps. The critical event prediction model subsequently provides the previously defined type of prediction. These results are then passed to a planning module to determine whether an adaptation is necessary. However, this planning module is beyond the scope of this thesis. Possible planning approaches can rely on rules (e.g., [KDM+18]), models (e.g., [PKWB17]), goals (e.g., [KM07]), or utility functions (e.g., [VSSB13]).

Although the results obtained by using this system model are very promising, the approaches included in the system model are nevertheless subject to certain assumptions. First, the proposed time series forecasting method, Telescope, assumes seasonal time series with a length of more than two seasonal patterns. However, if the time series does not meet this assumption, the meta-learning approaches to recommend the most suitable forecasting method can still be applied. Second, the main part of modeling expects a large training data set to learn the relationship between features and critical events. However, not only must the data set be large, especially the set of critical events must be sufficient. Third, if there is a concept drift in the data, the model will be updated using the update strategies in the feedback loop, but the performance of the model still decreases until the new pattern is significant enough in the new data.

1.4.2 Individual Contributions

On the basis of the proposed system model, we have derived seven main contributions of this thesis. While the first contribution focuses directly on the system model, the following six contributions can be assigned to the two goals introduced in Section 1.3, namely time series forecasting as well as machine learning and deep learning for critical event prediction.

Contribution 1: **System Model for Critical Event Prediction**

Given that most critical event prediction approaches in the literature are highly tailored to specific problems (e.g., estimating the health of lithium-ion batteries [MXC+13, ZXHP18] or predicting the remaining useful life of bearings [LWZ+14, HRBA+18]) and lack a proper generalizable model, we introduce such a system model for critical event prediction based on the LRA-M loop of Self-Aware Computing Systems. This system model can handle representative univariate data by applying time series forecasting and meta-learning for time series forecasting as well as multivariate data. For multivariate data, a more complex model learning process is followed. However, time series forecasting can also be used in this case to estimate future developments of sensor readings, allowing anomalous behaviors to be detected earlier and, thus, improving prediction quality.

Contribution 2: **Component-based Forecasting Method**

This contribution focuses on the research questions **RQ A.1** and **RQ A.2** of **Goal A**. Here, we propose a novel, hybrid time series forecasting method for seasonal time series based on a kind of divide-and-conquer technique to enhance the forecast accuracy. That is, the forecasting method uses time series decomposition, multiple state-of-the-art individual forecasting methods, and remainder learning via XGBoost. The proposed time series forecasting method represents a complete end-to-end approach involving frequency estimation, anomaly removal, feature extraction, composition type detection, and, eventually, forecasting.

To investigate the performance of the proposed component-based forecasting method, we assess the forecast accuracy and the time required to build the model and perform the forecast of the proposed method as well as numerous state-of-the-art forecasting methods. To this end, we use a data set consisting of 53 time series drawn from different areas of interest for threshold-based critical event prediction. The results show that the proposed component-based forecasting method achieves a higher average forecast accuracy and has a lower variation in forecast quality across the different time series than the other forecasting methods. Compared with the best competing state-of-the-art methods, our proposed method also yields a substantial speedup. Finally, we apply the proposed forecasting method to the scenario of predicting critical events for automatic scaling of virtual machines. Here, the results show that our proposed forecasting method considerably decreases the average response time and substantially reduces the number of service level objective violations.

Contribution 3: **Forecasting Method Recommendation Frameworks**

The remaining research questions of **Goal A**, namely research questions **RQ A.3** and **RQ A.4**, are addressed by this contribution. Specifically, we introduce two time series forecasting method recommendation approaches as well as an approach that combines a weighted forecast ensemble with method recommendation. While one of the recommendation approaches simply relies on the historical observations of the time series to be forecast, the other approach utilizes a large data set of time series and their forecasts to learn a model via machine learning techniques that estimates the most suitable forecasting method for a new time series only by computing its time series characteristics (for instance, kurtosis, skewness, and serial correlation). The third approach combining weighted forecast ensemble and method recommendation assigns weights according to these time series characteristics, but includes a dynamic threshold based on the derived weights to combine the most appropriate forecasting methods using a weighted ensemble.

We evaluate the first recommendation approach in an online competition to show its applicability and superiority over individual forecasting methods. Furthermore, we evaluate the other two approaches by comparing their forecast accuracy and the achieved ranks of the recommendation with state-of-the-art individual forecasting methods as well as a well-known recommendation approach for forecasting methods. Thereby, we identify the drawbacks of the existing approach and demonstrate that our approaches significantly enhance the forecast accuracy.

Contribution 4: **Critical Event Detection for Machine Tools**

This contribution addresses **Goal B** by approaching research question **RQ B.1**. We apply an instance of the proposed system model to critical event detection in a technical system, namely the detection of anomalous machine states. That is, we introduce a highly generalizable end-to-end workflow consisting of five steps, namely raw data processing, phase segmentation, data resampling, feature extraction, and machine tool anomaly detection. In this approach, only standard machine monitoring data are assumed and the only domain knowledge required is the number of work steps the machine must execute.

To evaluate the model, we acquire data from a CNC machine in normal condition and with an unbalance attached to the spindle to emulate a critical machine state. The results show that conventional frequency

analysis cannot detect the critical machine conditions well, whereas our model detects the critical events very well with an F1-score of almost 91%.

Contribution 5: **Critical Event Prediction Modeling Alternatives**

In order to predict critical events rather than merely detecting them, we address research question **RQ B.2** of **Goal B** by this contribution. Here, we compare different modeling alternatives for critical event prediction in the use case of hard disk drive time-to-failure prediction. First, we apply different oversampling strategies and evaluate their performance with respect to the final time-to-failure prediction quality. Furthermore, we assess the impact of binary class modeling compared with downscaled multi-class modeling. Finally, we model the time-to-failure not only by means of classification models, but also with regression models.

Based on the experimental results, we derive the finding that multi-class modeling is more capable of finding patterns in the data, even after downscaling the multi-class results to the same two classes as for the binary modeling. In addition, the regression model also predicts the critical events very well, however, such real-valued prediction is often not required and more error-prone than discretized time windows that indicate the time remaining until the critical event.

Contribution 6: **Critical Event Prediction for Industrial Machines**

This contribution addresses research question **RQ B.3** of **Goal B** by introducing an end-to-end workflow for predicting critical events of industrial machines. Here, critical events are represented by machine failures that result in downtime. Similar to the fourth contribution, this end-to-end workflow does not require any expert knowledge other than the definition of monitoring data and, therefore, represents a generalizable workflow for critical event prediction of industrial machines. The end-to-end workflow consists of four main steps, namely extraction of generally applicable features from sensor monitoring data, processing of the extracted features, mapping of target class labels to training instances, and model learning with integrated hyperparameter tuning via a grid-search technique. Based on the results of the fifth contribution, we model the time-to-failure in terms of multiple classes. Thereby, we compare different machine learning methods and strategies for multi-class labeling.

We evaluate the model using real-world production data of an industrial press. The results demonstrate that the model is capable of predicting the different time-to-failure windows very well, achieving a macro F1-score

of 90% for six different time-to-failure classes and an even higher F1-score of above 98% for the binary prediction of critical events.

Contribution 7: **Update Strategies for Critical Event Prediction Models**
The final research question **RQ B.4** of **Goal B** is approached by this contribution. Due to concept drift, critical event prediction models trained on a static training set need to be updated from time to time to properly cover the current patterns in the data. To this end, we present four update triggers to determine when critical event prediction models should be re-trained during on-line application. While one update trigger merely takes the elapsed time since the last update into account, the other three update triggers consider the achieved prediction quality.
We compare the prediction qualities achieved by the update triggers with each other as well as with the static baseline on a real-world data set of hard disk drive failures. To this end, we introduce a novel measure of hard disk drive prediction quality that combines the commonly used measures of failure detection rate and false alarm rate into a single measure. The results demonstrate the necessity of model updates during on-line application and suggest that the update triggers that take into account the prediction qualities of the current and previous test batches achieve the best overall trade-off between prediction quality and update cost in terms of required model re-trainings.

1.5 Use Cases for the Proposed System Model

To demonstrate the importance of the proposed system model for critical event prediction, we describe numerous use cases in which the meta self-aware system model has already proven useful or could be beneficial in the future.

1.5.1 Univariate Time Series Forecasting with Thresholds

A natural use case for critical event prediction based on forecasting of univariate time series in combination with thresholds is the scaling of virtual machines, also known as auto-scaling in cloud computing. Here, the workload, i.e., the number of incoming requests, can be monitored over time and used as input to forecasting methods to estimate the upcoming load to be handled by the virtual machines. By combining the estimation of the future arrival rate with queueing theory or simple thresholds, the critical events of overload, resulting in higher response times, and overprovisioning, resulting in wasted resources,

can be predicted and prevented by starting additional or stopping currently running virtual machines. Another example is the provision of electricity by energy suppliers. Here, the electricity demand requested by customers must be forecast to plan electricity delivery. The same applies to planning other services, such as sizing the number of flights to match the demand as closely as possible, or ordering fresh fruits and vegetables as a supermarket manager. Buying or selling stocks on the stock market can be considered a further use case for critical event prediction based on univariate time series forecasting. By means of such forecasting methods, the future development of the stock value can be estimated to determine the critical point for buying or selling the particular stock. Critical event prediction based on time series forecasting can also be useful for planning countermeasures to the SARS-CoV-2 situation. The last one and a half years have shown that reactive countermeasures have been inadequate due to the exponential spread of the virus and, therefore, the exponential growth of the incidence value. However, integrating forecasting into the critical event determination will allow for early estimation of the exceedance of the thresholds set by the government in order to take timely countermeasures.

1.5.2 Critical Event Prediction using Multivariate Monitoring Data

A typical use case for our system model is the technology sector, especially Industry 4.0. The concept of Industry 4.0 already includes continuous online monitoring, which provides the necessary database for prediction methods. Furthermore, another goal of Industry 4.0 is to reduce human intervention. This fits very well with the idea of Self-Aware Computing Systems. However, for a production plant to run (semi-)autonomously, impending machine failures must be known in advance so that countermeasures can be initiated. This is where the critical event prediction component comes into play. We have already deployed an instance of our system model for automatic detection of tool deterioration for CNC machines, which showed promising results. Furthermore, we used a version of our system model to predict the time-to-failure of a large-scale press. Here, we achieved a highly accurate prediction of upcoming machine downtimes using multiple prediction windows. However, Industry 4.0 is not the only application scenario in the technology domain. Another relevant area is cloud computing. According to Backblaze, hard disk drives (the driver of cloud computing) have a comparatively low annual failure rate of only about 2% [Kle20]. However, for large cloud providers running several thousands of hard disk drives in parallel, this translates into daily hard disk drive failures. Nevertheless, these cloud providers need to provide fast and reliable services to end users. Therefore, cloud providers need to identify failing hard

disk drives in advance based on monitoring data. To this end, we have developed a time-to-failure prediction approach based on components of our system model that compares different preprocessing steps and time-to-failure modeling alternatives.

Apart from the field of technology, critical events are also occurring in the area of biology. Due to the age of digitalization, more and more data are monitored and stored online in this field as well. One potential use case of the proposed system model in biology could be the prediction of severe heart infarctions. In clinics, it is a regular occurrence for patients to suffer a severe heart attack. If this happens outside the intensive care unit, these patients often die or suffer permanent damage. Here, even a 30-second to 2-minute lead time and an alarm that reaches doctors and nurses would help to save many lives. Another biological application could be prediction of insect mortality, especially predicting the development of bee colonies in terms of their population size. As most agricultural and wild plants are pollinated by bees, they are of great relevance to our crops and biodiversity. Studies have already shown that the biomass of flying insects in protected areas has decreased by an average of 2.8% per year over the last 27 years [LHS+17]. We are therefore working with two bee institutes that have already deployed more than 300 bee colony scales throughout Germany to monitor weight trends over the years. In the future, we want to use the proposed system model to predict critical developments at an early stage on the basis of these data, so that beekeepers can initiate timely countermeasures to keep their hives healthy.

1.6 Thesis Outline

The remainder of this thesis is organized into four parts. Part I provides relevant background information for a better understanding of the contributions presented in this thesis. In addition, this part summarizes the state-of-the-art in time series forecasting as well as critical event prediction using time series forecasting and machine learning methods. Subsequently, Part II describes our contributions on automated hybrid time series forecasting methods and presents our evaluations on them. Part III focuses on our second research goal, namely modeling, detecting, and predicting critical events with respect to failures in the technical domain. Finally, Part IV briefly summarizes the thesis and outlines potential future work.

Part I

Fundamentals

Chapter 2

Foundations

This chapters aims at providing the most relevant background information on methods applied in this thesis. However, we only present a brief introduction into the topics. For more detailed explanations, we refer to reference books on the respective topics. First, we present the concept of *Self-Aware Computing Systems*. For more information on Self-Aware Computing Systems, we refer to the book by S. Kounev *et al.* [KLB+17]. Next, we introduce fundamentals on time series analysis. For a deeper insight into time series analysis, we refer to J. Cryer and K.-S. Chan [CC08]. Following time series analysis, we present state-of-the-art statistical forecasting methods. A detailed book on time series forecasting in general can be found in the work of R. Hyndman and G. Athanasopoulos [HA18]. Hereafter, we briefly present a number of machine learning methods used in this thesis. For more details on clustering methods, we refer the reader to the book by L. Rokach and O. Maimon [RM05], while we refer to the book by T. Hastie *et al.* [HTF09] for classification and regression models. Finally, we succinctly present fundamentals on the deep learning methods employed in this thesis. An extensive reference on deep learning methods can be found in the book by I. Goodfellow *et al.* [GBCB16].

2.1 Self-Aware Computing Systems

Several concepts have been developed in the field of autonomous computing systems. For instance, well-known concepts include *Autonomic Computing Systems* [KC03], *Self-Aware Computing Systems* [KLB+17], and *Organic Computing Systems* [MSvdMW04]. As described in Section 1.4, the contributions of this thesis are situated in the concept of Self-Aware Computing Systems. Therefore, we first briefly introduce the vision of Self-Aware Computing Systems, followed by their key component, namely the so-called *LRA-M loop*. The content of this section is based on the book by S. Kounev *et al.* [KLB+17].

2.1.1 Vision of Self-Aware Computing Systems

In their book [KLB+17, p. 5], S. Kounev *et al.* defined Self-Aware Computing Systems as follows:

> "**Self-Aware Computing Systems** are computing systems that:
>
> 1. *learn models* capturing *knowledge* about themselves and their environment (such as their structure, design, state, possible actions, and runtime behavior) on an ongoing basis and
> 2. *reason* using the models (e.g., predict, analyze, consider, and plan) enabling them to *act* based on their knowledge and reasoning (e.g., explore, explain, report, suggest, self-adapt, or impact their environment)
>
> in accordance with *higher-level goals*, which may also be subject of change."

Hence, the concept of Self-Aware Computing Systems focuses on computer systems that use monitoring data to learn models about themselves and the environment. In this way, these systems gather knowledge and utilize it to reason and act on the basis of that knowledge, while striving to fulfill higher-level goals throughout their entire operation.

2.1.2 LRA-M Loop of Self-Aware Computing Systems

The key concept of Self-Aware Computing Systems that performs all elements presented in Section 1.4.1 is the LRA-M loop with its main components Learning, Reasoning, Acting, and Monitoring. Figure 2.1 shows the system, referred to as *self*, with its LRA-M loop and interfaces. The interfaces of the self serve to monitor both itself as well as the environment and obtain higher-level goals as input. On the basis of the acquired monitoring data (i.e., empirical observations), the self continuously learns a model (*learning* component). This learning is accomplished in two ways. First, an initial model is learned in an off-line step. Second, the model is continuously re-learned during runtime using the newly acquired empirical observations. The learned models, the self, and the goals form the knowledge base. Drawing on this knowledge base and newly incoming monitoring data, the *reasoning* component derives a finding that may trigger an action. The self monitors the achieved results, which in turn has an impact on the learning and reasoning of the self.

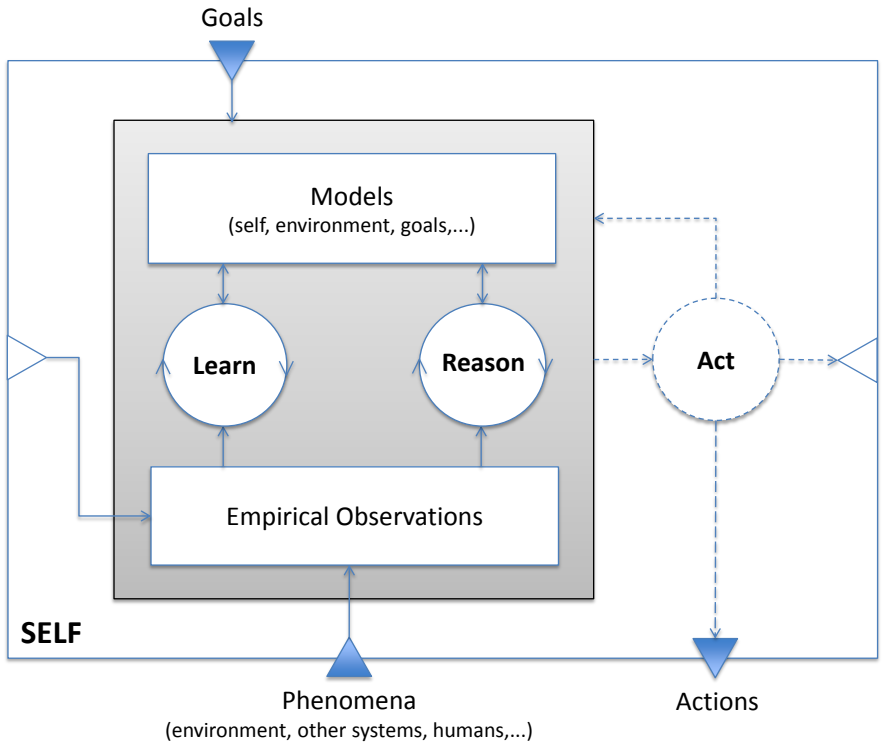


Figure 2.1: Learning and reasoning loop of Self-Aware Computing Systems taken from S. Kounev *et al.* [KLB+17].

In order to integrate automated proactive analysis into Self-Aware Computing Systems, primarily the learning component needs to be adapted. Figure 1.1 in Section 1.4 shows our vision of a meta self-aware analysis in the typical LRA-M loop with the goal of predicting critical events. To this end, the learning component requires a raw data preprocessing component to impute missing or anomalous values, transform the data into meaningful components and scales, and derive expressive features from the raw observations. Furthermore, time series forecasting is an important pillar to estimate future values of the observations and, thus, to provide additional information to the main model learning component. The derived features and forecasts are then used to train a machine or deep learning model for critical event prediction. To this end, the features and forecasts must be mapped to a suitable target, i.e., either to binary classes for detecting or predicting failures in general, to multiple classes for predicting the impending failure within time windows, or to a regression for obtaining an arbitrary real-valued estimate of the time remaining until the

critical event occurs. On top of these steps, a meta-self-aware component is integrated to target method recommendation. In addition, a feedback loop is implemented that exploits knowledge about past forecasts and predictions to improve the model via on-line re-training. With respect to the reasoning component, the derived critical event prediction model can be employed to obtain an estimate on the status of the system, which may trigger an action. For instance, a short time-to-failure of an industrial machine could trigger a maintenance action. However, planning and scheduling countermeasures is beyond the scope of this thesis. Possible planning approaches can rely on rules (e.g., [KDM+18]), models (e.g., [PKWB17]), goals (e.g., [KM07]), or utility functions (e.g., [VSSB13]).

2.2 Time Series Analysis

The research field of forecasting mainly focuses on *univariate, equidistant time series*. Thus, we first define the term univariate, equidistant time series. Such a univariate, equidistant time series X of length n is an ordered set of n observations x_t, where each observation x_t originates from the same time-dependent data generation process and is mapped to a unique time t, with the temporal difference between successive observations Δt being a constant value. Formally, a univariate, equidistant time series X with length n is defined as

$$X := \{x_t : t \in T\}, \tag{2.1}$$

where T is a discrete set of equidistant points in time with $|X| = n$. However, the temporal difference between two successive observation time points of a time series is not necessarily of equal length. In such a case, the time series is still univariate, but the equidistant property is lost. Such *univariate, non-equidistant time series* are out of the scope of this thesis.

Moreover, a time series can also be of *multivariate* nature. In a multivariate time series, unlike univariate time series, several time-dependent variables are observed. However, all variables of a multivariate time series are sampled at the same points in time. Furthermore, the observed variables in a multivariate time series are not only time-dependent, but also exhibit interdependencies. A typical example of such a multivariate time series are weather measurements when considering temperature, humidity, precipitation, cloud cover, and probability of rainfall. In the following, however, we use the term time series to refer to univariate, equidistant time series unless otherwise specified.

A well-known example of a univariate, equidistant time series is the number of monthly international airline passengers from 1949 to 1960. This time series

is displayed in Figure 2.2, where the horizontal axis represents the time domain and the vertical axis the number of monthly international airline passengers in thousands. As can be seen, this time series exhibits numerous typical time series properties, such as trend, seasonality, and multiplicative composition. The following sections provide relevant background information on such time series characteristics and how these can be extracted from a raw time series.

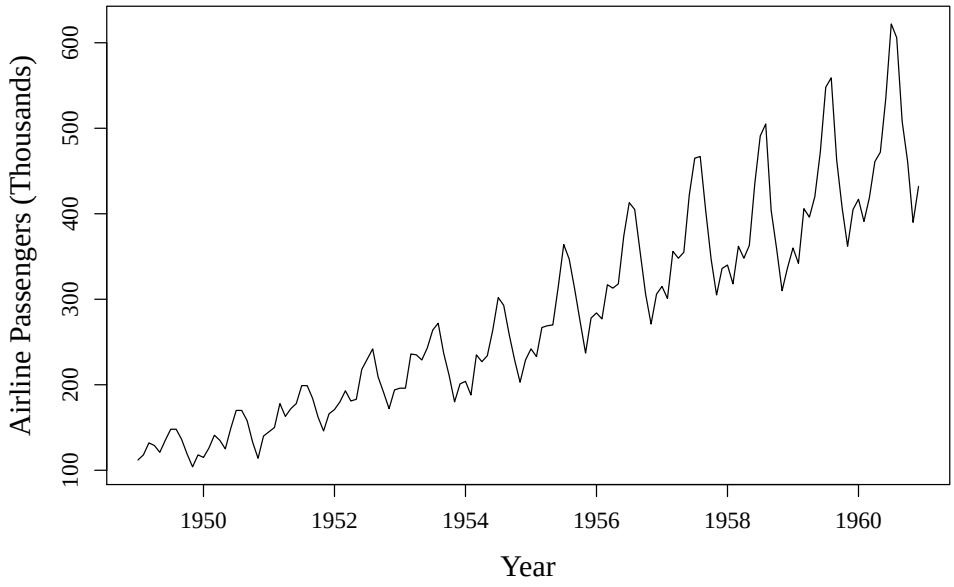


Figure 2.2: The number of monthly international airline passengers in thousands from 1949 to 1960.

2.2.1 Time Series Characteristics

For describing time series in terms of properties, there exist many different characteristics. Apart from general characteristics, such as sample mean and sample standard deviation, several more characteristics have been proposed to be used for time series characterization [WSMH09, LG10]. However, to focus on the essential features for this thesis, we present only those employed in the contributions. Note that the different characteristics have different value ranges. Yet, in order to derive rules in an automatic manner, the raw values of the

features are typically less important than its degree of dominance. Therefore, time series characteristics are typically normalized into the range $[0, 1]$, with a larger value indicating a stronger presence of the particular property.

- *Trend strength*: With respect to time series analysis, trend is the long-term movement of observations. Unless an external trigger is present, the trend is usually a monotonic function, i.e., stagnant, increasing, or decreasing. The strength of a time series is thereby calculated as

$$1 - \frac{\text{Var}\left(X_{-S,-T}\right)}{\text{Var}\left(X_{-S}\right)}, \tag{2.2}$$

  where $\text{Var}\left(X_{-S,-T}\right)$ denotes the variance of the de-seasonalized and de-trended time series, while $\text{Var}\left(X_{-S}\right)$ represents the variance of the de-seasonalized time series.

- *Seasonal strength*: In contrast to trend, seasonality of time series refers to regularly recurring oscillations. Typical drivers for such seasonal fluctuations are yearly climate patterns and common human behavior (for instance daily routines or traditional habits like Christmas). The seasonal strength is calculated similarly to the trend strength, but the denominator is changed to the variance of the de-trended time series X_{-T}:

$$1 - \frac{\text{Var}\left(X_{-S,-T}\right)}{\text{Var}\left(X_{-T}\right)}. \tag{2.3}$$

- *Frequency*: The frequency, often also called periodicity, is closely related to seasonality, as it reflects the length of the seasonal pattern. Determining the frequency of a time series is a complex task. An approach to calculating the frequency is presented in Section 2.2.3.

- *Skewness*: Whether the distribution of a time series is symmetric or not is described by the skewness, which exhibits a value range of $[-1, 1]$, where a value of 0 indicates a strongly symmetric distribution of the time series and a non-zero value indicates that the distribution of the time series lacks symmetry. While a negative skewness indicates that the distribution of the time series is skewed left, a positive skewness indicates that the distribution of the time series is skewed right. Finally, the skewness S is the third standardized moment and is therefore calculated as

$$S = \frac{1}{n\sigma^3} \sum_{i=1}^{n} \left(X_i - \overline{X}\right)^3, \tag{2.4}$$

where $\overline{X}$ denotes the mean of all observations in the time series, X_i is the i-th observation in the time series, σ represents the standard deviation of the time series, and n is the length of the time series.

- *Kurtosis*: The kurtosis describes whether a distribution is peaked or flat with respect to the normal distribution, with a high kurtosis representing a sharp peak near the mean that declines rapidly into heavy tails. In contrast, a low kurtosis indicates that the time series tends to have a flat peak near the mean. A shifted version that maps the normal distribution to a value of zero is called excess kurtosis. In this case, a high kurtosis is implied by a positive value, while a negative value represents a low kurtosis. In the following, we use only the excess kurtosis K, defined by

$$K = \frac{1}{n\sigma^4} \sum_{i=1}^{n} \left(X_i - \overline{X}\right)^4 - 3. \tag{2.5}$$

- *Serial correlation*: The serial correlation, often also called autocorrelation, refers to the correlation of a time series with itself at an earlier point in time at lag k. Thus, a high serial correlation indicates repeating, i.e., seasonal, patterns in a time series. To calculate the serial correlation Q_h, the Box-Pierce statistics [BP70] are used:

$$Q_h = n \cdot \sum_{k=1}^{h} r_k^2, \tag{2.6}$$

  where h is the maximum lag considered (usually $h \approx 20$) and r_k represents the correlation coefficient between the original time series and the time series with lag k.

- *Non-linearity*: A system is considered non-linear if the changes of the input are not proportional to the changes of the output. Hence, non-linearity describes the degree to which the time series can be poorly written as a linear combination of unknown variables or functions. In order to test for non-linearity, non-parametric kernel tests or neural networks can be utilized. In this thesis, Teräsvirta's neural network approach [TLG93] is implemented for this purpose.

- *Self-similarity*: The similarity of an object to a part of itself is expressed by the self-similarity, which is given by the Hurst exponent H [WPT98, Ros96]. To this end, Autoregressive Fractionally Integrated Moving Average (ARFIMA) processes can be employed as estimation method for

computing H. In ARFIMA models, the parameter of the integrated part of the Autoregressive Integrated Moving Average (ARIMA) model d is replaced by a non-integer value. To estimate d, an ARFIMA$(0, d, 0)$ is fitted with maximum likelihood. Finally, the Hurst exponent H is calculated as

$$H = d + 0.5. \tag{2.7}$$

- *Chaos*: In order to understand the random behavior of a time series, identifying and investigating chaos is of great value. The characteristic chaos is given by the Lyapunov exponent, which is determined by the method of R. C. Hilborn [Hil00].

2.2.2 Seasonality and Cyclicality of Time Series

According to R. Hyndman and G. Athanasopoulos, the terms *seasonality* and *cyclicality* are often confused with each other or used synonymously in the field of time series analysis [HA18]. Yet, the distinction between seasonal and cyclic time series is crucial for time series modeling. Therefore, we briefly introduce both types of time series and illustrate the differences using examples based on R. Hyndman [Hyn11].

The main property of seasonality in time series are fluctuations with a fixed frequency around a certain baseline. For this reason, such time series are often referred to as *periodic*. The oscillations are typically caused by seasonal influence factors or the day-and-night behavior of humans. In fact, seasonal time series may also exhibit multiple seasonal patterns with different frequencies. An example of such overlapping frequencies can be caused by a weekday and weekend pattern in addition to the day-and-night pattern. Figure 2.3 displays such a seasonal time series with two overlapping frequencies. This time series depicts the electricity demand of Victoria, Australia, in Gigawatts from the beginning of week 25 to the end of week 29 of 2014. Each value in the curve specifies the electricity demand for the last half an hour. Victoria's electricity demand is affected by two seasonal factors. First, the time series encompasses a diurnal (day-and-night) period. The electricity demand increases until midday, after which the electricity demand decreases and bottoms out in the middle of the night. This behavior repeats with a fixed regularity. In addition, a weekly period (weekdays vs. weekends) is evident. Apart from weekends, the pattern of electricity demand is similar for each day. On weekends, in contrast, the peaks are substantially lower than on weekdays. Since this pattern occurs weekly, the frequency is also constant.

Similar to seasonal time series, cyclic time series also exhibit recurring fluctuations with peaks and troughs, but unlike seasonal time series, these fluctuations

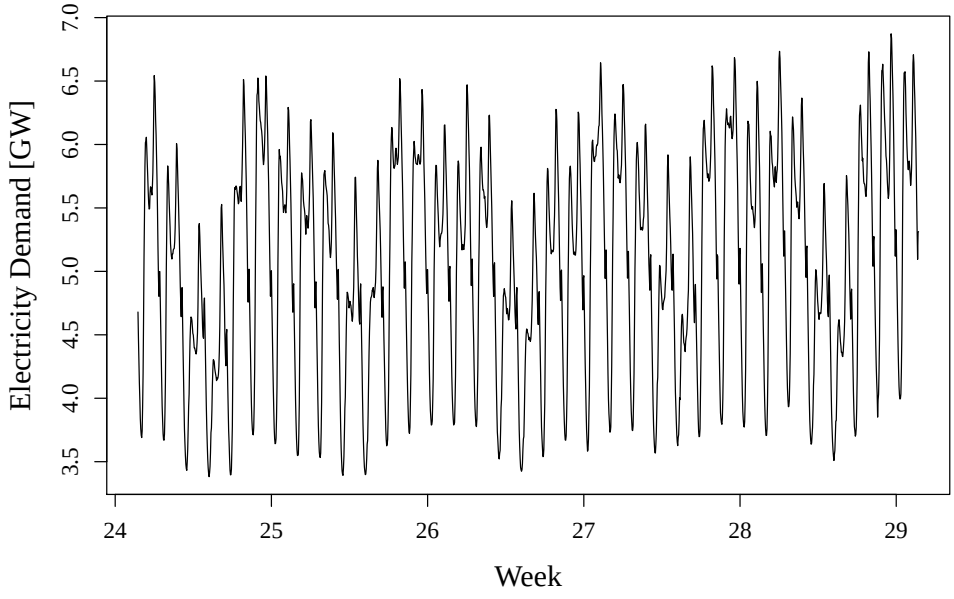


Figure 2.3: The half-hourly electricity demand of Victoria, Australia, in GW in 2014.

actually do not emerge with a fixed frequency. Therefore, time series exhibiting such non-periodic behavior are referred to as cyclic instead of seasonal. Apart from the different distances between the fluctuations, cyclic time series often also show highly varying amplitudes of the peaks. In contrast, seasonal time series typically have roughly the same amplitude of their periods, unless the seasonal pattern is overlapped by another seasonal pattern or the time series exhibits a multiplicative composition type (cf. Section 2.2.4). An example cyclic time series representing the annual number of lynx trappings in Canada from 1821 to 1934 is shown in Figure 2.4. At first glance, the time series appears to be seasonal due to the repeating peaks, but a closer look at the time series reveals that some of the cycles last 8 years, 9 years, and sometimes even 10 years or more. Thus, the frequency of the cycles is not fixed. Furthermore, the amplitudes of the peaks differ highly, although there is no trend in the time series, nor is there any overlapping seasonality evident.

As already pointed out, the distinction between seasonal and cyclic time series is a critical aspect, because some forecasting methods cannot handle seasonal time series, while others perform much better for seasonal time series.

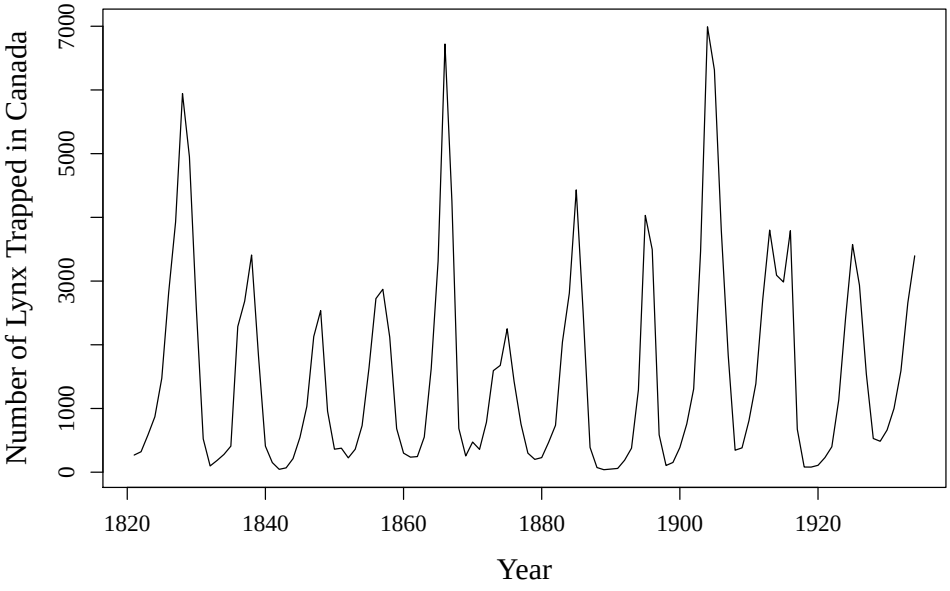


Figure 2.4: The annual number of lynx trappings in Canada from 1821 to 1934.

Therefore, distinguishing whether a time series is seasonal or cyclic is also of great importance for the automatic selection of time series forecasting methods. Finally, with respect to cyclic time series, typical frequency estimation approaches also find a dominant frequency, although applying the derived frequency to cyclic time series leads to poor forecasting results, since the time series do not have a steady periodic pattern and, therefore, the forecast drifts compared to the actual values.

2.2.3 Periodograms for Frequency Estimation

In order to determine the frequency of a seasonal time series, spectral analysis is typically applied. In the field of spectral analysis, a time series is considered as a sum of cosines and sines with different frequencies and amplitudes. Based on this assumption, the periodogram is a mathematical tool for estimating spectral density [Sch98]. To identify the most dominant frequency, the periodogram computes the spectrum for numerous frequencies by applying a brute force algorithm. Thus, for each potential frequency, the periodogram determines the

spectrum using Fourier transform. More specifically, the periodogram involves splitting a time series X with observations $x_t, t = 1, \ldots, n$, into multiple cosines and sines under the constraint that the sum of the cosines and sines found along with the mean value of the time series must yield the original time series observations. Thus, the *discrete Fourier transform* is applied for the frequencies $v_k = \frac{k}{n}$ with $k = 1, \ldots, \frac{n}{2}$:

$$\begin{aligned} X(v_k) &= \frac{1}{\sqrt{n}} \sum_{t=1}^{n} e^{-2\pi i t v_k} x_t \\ &= \frac{1}{\sqrt{n}} \sum_{t=1}^{n} \cos(2\pi t v_k) x_t - i \frac{1}{\sqrt{n}} \sum_{t=1}^{n} \sin(2\pi t v_k) x_t \end{aligned} \tag{2.8}$$

Subsequently, the periodogram computes the squared modulus of the discrete Fourier transform as $I(v_k)$:

$$\begin{aligned} I(v_k) &= |X(v_k)|^2 \\ &= \frac{1}{n} \left| \sum_{t=1}^{n} e^{-2\pi i t v_k} x_t \right|^2 \\ &= \left(\frac{1}{\sqrt{n}} \sum_{t=1}^{n} \cos(2\pi t v_k) x_t \right)^2 + \left(i \frac{1}{\sqrt{n}} \sum_{t=1}^{n} \sin(2\pi t v_k) x_t \right)^2 \\ &= \frac{1}{n} \left(\left(\sum_{t=1}^{n} \cos(2\pi t v_k) x_t \right)^2 + \left(\sum_{t=1}^{n} \sin(2\pi t v_k) x_t \right)^2 \right) \end{aligned} \tag{2.9}$$

The result of applying the periodogram to the airline passengers time series (cf. Figure 2.2) is given in Figure 2.5. Note that for illustrative purposes, we have replaced the frequencies v_k with the respective length of the seasonal pattern under consideration. As this time series exhibits annual seasonality, the length of a seasonal pattern is twelve. This is also apparent in Figure 2.5, as the computed spectrum shows a prominent peak for this frequency. Furthermore, decreasing peaks can also be seen for multiples of this frequency.

In general, a high spectral value $I(v_k)$ indicates a dominant frequency, while small spectral values can be caused by noise. For noisy, random time series, however, the spectrum should have a similar value for all possible frequencies, since such time series do not contain a distinct seasonal pattern. Although the periodogram finds a dominant frequency for seasonal time series, it may

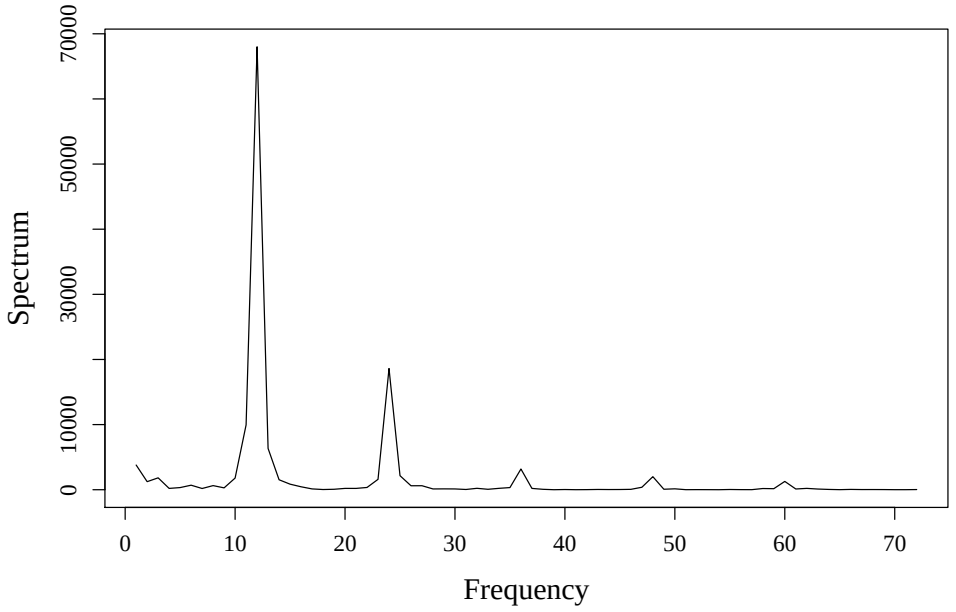


Figure 2.5: The resulting periodogram for the monthly international airline passengers time series (cf. Figure 2.2).

also find a dominant frequency for cyclic time series. As demonstrated in Figure 2.4, cyclic time series can also have sinusoidal patterns, making the automatic distinction between seasonal and cyclic time series a particularly challenging and error-prone task. Thus, the distinction between seasonal and cyclic time series must be carried out prior to the application of periodograms, since a meaningful frequency should be calculated only for seasonal time series. If this is not done, the forecasting accuracy can be severely reduced.

Note that the frequency of a time series can also be approximated by autocorrelation when iterating over the lag parameter k (cf. Section 2.2.1). However, autocorrelation has the same problem of distinguishing seasonal from cyclic time series. Furthermore, spectral analysis typically provides more accurate results and should therefore be preferred over autocorrelation.

2.2.4 Time Series Decomposition

As already mentioned in Section 2.2.1, a time series can be thought of as a combination of different components. In time series analysis, a common approach is to reverse this composition and decompose a time series into its individual components. Although several decomposition models have been proposed in the literature, we will focus only on the *structural time series model* (STSM). Based on the STSM, a time series X consists of three main components, namely season S, trend T, and irregular I, which is also often referred to as remainder. However, depending on the type of composition, the original time series is either composed as the sum of the individual components, i.e.,

$$X = S + T + I, \tag{2.10}$$

or as the product of the individual components, i.e.,

$$X = S \cdot T \cdot I. \tag{2.11}$$

The first case is called *additive composition*, while the second case is called *multiplicative composition*. An additive composition is present if the amplitude of the seasonal fluctuations remains the same regardless of the trend. In contrast, a time series exhibits a multiplicative composition if the amplitude of the seasonal pattern changes with respect to the trend level. An example of a multiplicative time series is shown in Figure 2.2. Here, the amplitude of the seasonal pattern increases considerably with rising trend.

A commonly used method for dividing a time series into these components is *Seasonal and Trend decomposition using Loess* (STL) [CCMT90]. The main advantages of STL over other time series decomposition methods based on STSM are that it allows for arbitrary seasonal frequencies, is able to extract changes in the seasonal pattern over time, and allows the user to control the smoothing of the trend component. The main procedure of STL is performed in two nested loops. The inner loop is used to smooth the seasonal and trend components, while the outer loop is used to calculate the irregular component as the difference between the actual observations and the smoothed seasonal and trend components. In addition, each time point is assigned a robustness score with respect to the corresponding value of the irregular component. These robustness scores are then used for the next iteration of the inner loop. Yet, this is only a brief insight into the STL algorithm. For further insights, please refer to the original publication by R. Cleveland *et al.* [CCMT90].

Although STL can only handle additive decomposition of time series, a multiplicative time series can easily be transformed to show an additive composition.

For this purpose, the logarithm is applied to the original time series. Note that for applying the logarithm to the time series, the time series might need to be shifted along the vertical axis so that there are no values less than or equal to zero. To prove the equivalence of the multiplicative decomposition and the additive decomposition on logarithmized time series, it can be shown that

$$\begin{aligned} & X = S \cdot T \cdot I \\ \iff \quad & \log(X) = \log(S \cdot T \cdot I) \\ & \qquad = \log(S) + \log(T) + \log(I). \end{aligned} \tag{2.12}$$

Figure 2.6 illustrates the result of the STL decomposition on the logarithmized monthly international airline passenger time series (cf. Figure 2.2). The horizontal axes always represent the time, while the logarithmized time series and the extracted components, i.e., logarithmized season, logarithmized trend, and logarithmized irregular, are displayed in the different subfigures on the vertical axes from top to bottom. The logarithmized time series shows that the multiplicative behavior has been successfully transformed into an additive composition type. Furthermore, a clear seasonal pattern and a monotonic trend are extracted by STL, leaving only a small irregular component[1].

Once the components are determined, the time series can also be adjusted by removing certain components of the time series. The most common combinations are de-seasonalization, de-trending, and de-trending together with de-seasonalization. The *de-seasonalized time series* X_{-S} is defined by

$$X_{-S} := X - S = T + I. \tag{2.13}$$

By contrast, the *de-trended time series* X_{-T} is defined by

$$X_{-T} := X - T = S + I. \tag{2.14}$$

Finally, the *de-seasonalized and de-trended time series* $X_{-S,-T}$ is defined by

$$X_{-S,-T} := X - S - T = I \tag{2.15}$$

and, thus, equals the irregular component.

[1] Note that the gray bar represents the same area for all four subfigures. Thus, a large gray bar of a component shows that the respective component accounts for only a small portion of the time series.

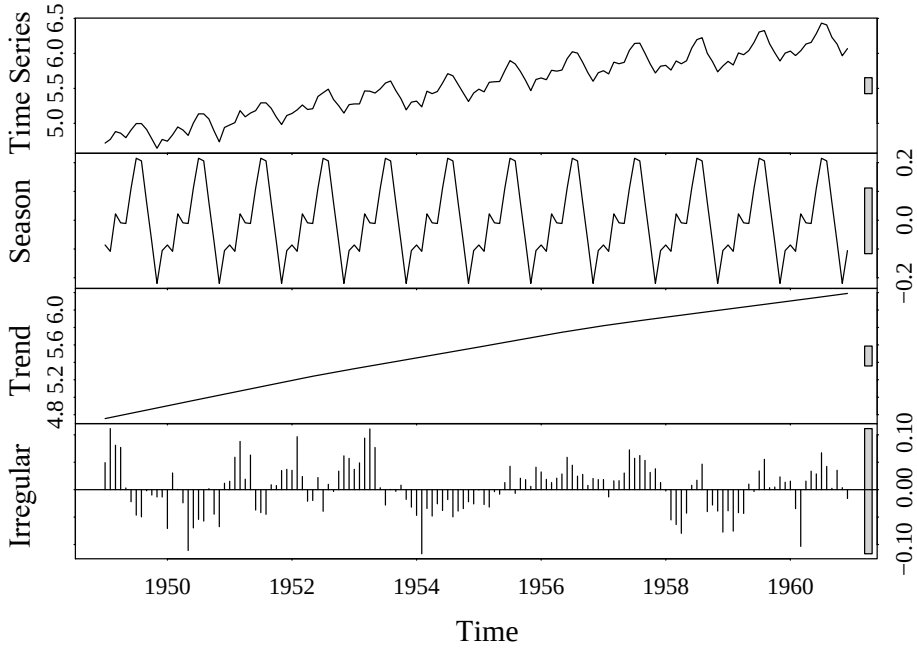


Figure 2.6: The resulting STL decomposition for the logarithmized monthly international airline passengers time series (cf. Figure 2.2).

2.3 Time Series Forecasting

"Prediction is very difficult, especially if it's about the future" is a famous quote attributed to Niels Bohr. This statement proves correct, as there is an entire field of research that focuses exclusively on the study of historical data in order to make predictions on future values, namely the research field of *time series forecasting*. Although sometimes referred to as predictions, the process of analyzing time series, especially univariate time series (cf. Section 2.2), to make estimates on future observations is generally referred to as forecasting. As time series are ubiquitous in our world, forecasting is an essential pillar in the decision-making process and, therefore, is used in many fields like business, economics, finance, science, and engineering.

Generally speaking, time series forecasting consists of three main steps.

1. Acquisition and examination of historical data, also referred to as time series observations.

2. Fitting a model that captures the properties and structure of the historical time series observations.

3. Provision of estimates on future observations, i.e., the forecast, and their evaluation as soon as the actual values are available.

Such forecasts can either estimate a single future value, which is referred to as a *one-step-ahead* forecast, or provide estimates for several successive time points in a single step, which is referred to as *multi-step-ahead* forecasts.

As time series forecasting is of major importance for many different practical applications and the "No Free Lunch Theorem" [WM97] also holds true for the field of time series forecasting, a wide variety of time series forecasting methods have been developed. To provide a brief overview of commonly used time series forecasting methods, the following subsections present a number of basic as well as state-of-the-art individual forecasting methods.

2.3.1 Naïve Forecasting

The simplest and most intuitive individual forecasting method is the so-called *Naïve* forecast. This method assumes that the last known observation x_n provides the most information on future values and, consequently, this last known value is forecast for the entire forecast horizon. Formally, the Naïve forecast can be expressed as

$$\hat{x}_{n+k} = x_n, \tag{2.16}$$

where $\hat{x}_{n+k}$ is the forecast for k time points in the future.

For seasonal time series, in contrast, the Naïve forecast does not forecast the last known observation, but the last known observation with distance equal to a multiple of the time series frequency. This method is called *sNaïve* forecast and is defined by

$$\hat{x}_{n+k} = x_{n+k-f(p+1)}, \tag{2.17}$$

where f denotes the frequency of the time series and $p = \left\lfloor \frac{k}{f} \right\rfloor$ represents the number of periods to look back to the known observation.

Finally, an extension of the Naïve forecast is presented by the so-called *Random Walk* model, which is also known as ARIMA(0,1,0). Here, the time series is assumed to exhibit random behavior and, thus, the Random Walk model forecasts the last known value together with a cumulative sum of randomly sampled values. This process is defined by

$$\hat{x}_{n+k} = x_n + \sum_{i=1}^{k} r_i, \tag{2.18}$$

where r_i are observations of an *independent and identically distributed* (i.i.d.) process. In the domain of time series forecasting, such a process is also referred to as ARIMA(0,0,0) (cf. Section 2.3.4).

The main advantage of Naïve and Random Walk forecasts is quite intuitive: The application of these methods incurs hardly any computational costs and the forecasts can therefore be delivered extremely fast. However, since the forecasting accuracy is often rather poor, these methods are only used as a baseline in real applications.

2.3.2 Exponential Smoothing State Space

A well-known approach to extrapolating data is the use of exponential smoothing. The general concept of *Simple Exponential Smoothing* (SES) was introduced by R. Brown in 1956 [Bro56]. The SES model extrapolates a future value by assigning a weight to each observation in the history and, then, computing the sum of the weighted observations. Therefore, the one-step-ahead forecast of an SES model is defined as

$$\begin{aligned}\hat{x}_{n+1} &= \alpha x_n + (1-\alpha)\,\hat{x}_n \\ \hat{x}_{n+1} &= \alpha x_n + (1-\alpha)\,(\alpha x_{n-1} + (1-\alpha)\,(\alpha x_{n-2} + (1-\alpha)\,(\dots))) \\ &= \alpha x_n + \alpha\,(1-\alpha)\,x_{n-1} + \alpha\,(1-\alpha)^2\,x_{n-2} + \cdots + \alpha\,(1-\alpha)^{n-1}\,x_1,\end{aligned} \tag{2.19}$$

where $0 < \alpha < 1$ represents the smoothing factor of the SES model. The common assumption in time series forecasting is that more recent observations are more relevant and represent future observations better. Therefore, α is typically set quite high. However, the drawback of this approach is that it does not consider trend or seasonal components of the time series. To address this drawback, several adaptations of the SES model have been proposed. In 1957 and 1960, C. Holt and P. Winters presented approaches that took into account the seasonal and trend components [Hol57, Win60]. However, since each component (i.e., trend and season) can be either nonexistent, additive, or multiplicative, there are nine possible combinations. To provide assistance on when to choose which combination, several researchers have provided articles with guidelines [Peg69, GJ85]. Subsequently, R. Hyndman *et al.* [HKOS08] introduced a framework for modeling time series based on the *Exponential Smoothing State Space Model*. The model distinguishes between the three components error, trend and season, which is also the reason why the framework is called *ETS*. The selection of the component combination in ETS can either be set manually or left to the framework. In the latter case, ETS determines the

most appropriate combination based on information criteria, such as Akaike's Information Criterion (AIC) and Bayesian Information Criterion (BIC).

2.3.3 Autoregressive Moving Average

Another class of individual forecasting methods are the *Autoregressive Moving Average* (ARMA) models. ARMA models, as the name implies, consist of two submodels, the autoregressive AR(p) model and the moving average MA(q) model. On the one hand, the AR part forecasts future values as a linear combination of p historical observations x_{t-i}, referred to as order p, their weights φ_i, also referred to as AR coefficients, a white noise random error ϵ_t, and a constant term c. Thus, the AR(p) model is defined as.

$$x_t = \sum_{i=1}^{p} \varphi_i \cdot x_{t-i} + \epsilon_t + c. \tag{2.20}$$

On the other hand, besides the white noise random error ϵ_t and a constant term c, the MA part of ARMA takes the q past errors ϵ_{t-j} and their weights Θ_j, referred to as MA coefficients, to forecast the future value. Formally, the MA(q) model is defined by

$$x_t = \sum_{j=1}^{q} \Theta_j \cdot \epsilon_{t-j} + \epsilon_t + c. \tag{2.21}$$

Finally, the entire ARMA(p,q) model is defined by

$$x_t = \sum_{i=1}^{p} \varphi_i \cdot x_{t-i} + \sum_{j=1}^{q} \Theta_j \cdot \epsilon_{t-j} + c + \epsilon_t \tag{2.22}$$

Note that the random errors ϵ_t are assumed to be i.i.d. and follow the standard normal distribution. The parameters p and q of an ARMA model can be estimated using the Box-Jenkins method [BJ70].

2.3.4 Autoregressive Integrate Moving Average

The *Autoregressive Integrating Moving Average* (ARIMA) model [BJ70] is a common choice for individual forecasting methods and is an extension of the ARMA model introduced in Section 2.3.3. The main drawback of ARMA models is their strong assumption of stationary time series. In time series analysis, stationarity means that the properties of the time series do not depend on the time. For this reason, time series exhibiting a trend or seasonal pattern are non-stationary,

since the observed values depend on the point in time. In contrast, an example of a stationary time series would be white noise, because the time of observation has no effect on the monitored value. To overcome this shortcoming of ARMA models, ARIMA models transform a non-stationary time series into a stationary time series by differencing. Therefore, ARIMA models contain the same parameters as ARMA models (i.e., the order of the AR model p and the order of the MA model q) together with an additional parameter, namely, the order of the integrated part of the model d. Note that in the case of $d = 0$, the ARIMA(p,d,q) model is equivalent to an ARMA(p,q) model.

However, the ARIMA model is also based on two assumptions. First, it assumes that the time series under consideration is linear and second, it further assumes that the time series follows a certain known statistical distribution. Note that the first assumption is extremely stringent, since real-world data are hardly only linear. Therefore, the main drawback of ARIMA models is the linear shape assumption of the model [Zha03]. To address the limitations of the normal ARIMA models, numerous variants have been proposed. For the particular assumption of linearity, the *seasonal ARIMA* (sARIMA) model was developed for seasonal time series. In contrast to the ARIMA model, the sARIMA model is defined by seven parameters. In addition to the three parameters of ARIMA, sARIMA requires the same parameters for the seasonal components along with the length of a seasonal pattern. In the following, we use the term ARIMA models to refer to the set of ARIMA and sARIMA models, unless we specify precisely that seasonality is excluded.

Finally, similar to ARMA models, the parameters of an ARIMA model and its variations can also be estimated using the Box-Jenkins method [BJ70].

2.3.5 Trigonometric, Box-Cox Transformation, ARMA Errors, Trend and Seasonality Model

The main shortcoming of exponential smoothing models, namely their poor ability to fit complex seasonal patterns, is addressed by the so-called *Trigonometric, Box-Cox Transformation, ARMA Errors, Trend and Seasonality* (TBATS) model [DLHS11]. As the name implies, this method combines several existing techniques, namely Fourier terms to model different potential seasonalities, the Box-Cox transformation [BC64] to reduce the variance in the time series and, consequently, to deal with non-linearity, and ARMA error corrections. To determine the most suitable model, TBATS fits numerous models with different parameter settings, such as whether or not to apply the Box-Cox transformation, whether seasonality should be used, or if there is a significant trend. Finally, Akaike's Information Criterion is applied to determine the best model.

2.3.6 Neural Network Autoregression

A special neural network design called *Neural Network Autoregression* (NNetAR) can be used to forecast time series. The architecture of NNetAR is a Feed-Forward Neural Network with a single hidden layer [HAB+18]. For more information on Feed-Forward Neural Networks, please refer to Section 2.5.1. As input, NNetAR receives lagged values of the time series. Therefore, the NNetAR model is defined by two parameters. The first parameter p corresponds to the order of the AR model and, thus, denotes the number of lagged inputs. By contrast, the second parameter k indicates the number of nodes in the hidden layer. In other words, the input layer has the size of the AR order p, the hidden layer consists of k nodes, and the output layer is set to a single node to forecast one value. To forecast multiple steps at once, an iterative process is used that integrates the forecast of the previous iteration.

The two-parameter NNetAR model can also be extended for seasonal time series by adding the parameters f and P, which denote the frequency, i.e., the length of a seasonal pattern, and the number of seasonally lagged inputs, respectively [HA18].

2.4 Machine Learning Methods

Unlike univariate time series forecasting methods, machine learning methods require features to learn a relationship between the desired target and the input. Therefore, machine learning models cannot directly learn the temporal dependency between successive observations. To solve this problem, on the one hand, features can be forecast using the techniques presented in Section 2.3. Then, these feature forecasts can be passed to machine learning methods to predict future observations. Similarly, the original observations can be lagged and passed as features to the machine learning algorithm (cf. Section 2.3.6). However, only a limited number of predictions can be made using this technique, i.e., only as many as the minimum of the applied lags. On the other hand, targets can be created so that the time component is inherently included.

In general, a machine learning task f can be formulated as a learning problem, which involves mapping a matrix of features $\mathbf{X}$ to a vector of targets $\mathbf{Y}$

$$f : \mathbf{X} \mapsto \mathbf{Y}, \tag{2.23}$$

where Y can be either a finite set of known classes (*classification*), any real-valued number (*regression*), or a set of unknown classes (*clustering*).

With respect to critical event prediction, the target can represent the remaining time until the next critical event as an arbitrary number. In this case, the

problem would be modeled as a regression task. However, if the remaining time is not required to be this fine-grained, the remaining time can also be discretized into bins, resulting in a classification task. For both approaches described, the actual target must be known in order to learn the regression or classification model and evaluate the prediction quality. Therefore, such problems are called *supervised learning*.

In contrast, if the targets are not known, clustering can be applied to identify patterns in the data and group similar instances into the same cluster while leaving dissimilar instances in different clusters. Such a problem is referred to as *unsupervised learning*.

This section provides a brief introduction to commonly used machine learning methods for clustering, classification, and regression. However, since a vast number of approaches have already been proposed, we focus on the ones employed in the contributions of this thesis. For more details on clustering methods, we refer the reader to the book by L. Rokach and O. Maimon [RM05]. In addition, the book by T. Hastie *et al.* [HTF09], provides useful information on classification and regression methods.

2.4.1 K-Means Clustering

In order to partition a set of instances $\mathbf{X}$, each represented by a feature vector $\mathbf{x}_i$, into a certain number of groups that show high cohesion within groups and high separation between groups, clustering algorithms can be applied. Thus, the objective of clustering is to ensure that instances in the same group are similar to each other and dissimilar to instances in other groups. A popular method in this area is the *k-Means* algorithm. As input, the k-Means algorithm requires only the feature vectors $\mathbf{x}_i$ and the desired number of clusters to be found. The k-Means algorithm goes back to several researchers. However, it is most commonly attributed to S. Lloyd [Llo82] and E. Forgy [For65]. Formally, the general goal of the k-Means algorithm is defined by

$$\arg\min_{S} \sum_{j=1}^{k} \sum_{\mathbf{x} \in S_i} ||\mathbf{x} - \mu_j||^2 = \arg\min_{S} \sum_{j=1}^{k} |S_j| \mathrm{Var}\left(S_j\right), \tag{2.24}$$

where S is the set of clusters with S_j being the j-th cluster and μ_j is the mean feature vector for the instances in cluster S_j.

To achieve this goal, the k-Means algorithm follows a two-step approach, after an initial random placement of the cluster means μ_j in the feature space. Subsequently, each instance $\mathbf{x_i}$ is mapped to the cluster S_j with the closest mean μ_j. For this purpose, the squared Euclidean distance is employed. In the

second step, the cluster means μ_j are updated as centroids of the respective cluster S_j. These two steps are repeated iteratively until the assigned instances no longer change between clusters.

2.4.2 Hierarchical Clustering

Contrary to the k-Means algorithm, *hierarchical clustering* does not start with the desired number of clusters, but uses either a bottom-up or a top-down approach. The bottom-up approach, also referred to as *agglomerative*, starts with as many clusters as there are instances $\mathbf{x}_i$ in $\mathbf{X}$. That is, each instance has its own cluster. Then, in each step, two clusters are merged to reduce the number of clusters until only one cluster is left, representing the set of all instances $\mathbf{x}_i$. In contrast, the top-down approach, also referred to as *divisive*, works the other way round. More precisely, the algorithm starts with a single cluster containing all instances $\mathbf{x}_i$, and splits one cluster into two separate clusters in each step until $|S_j| = 1$, i.e., each cluster contains only a single instance. Both approaches result in a tree structure, referred to as a *dendrogram*. A dendrogram typically shows the root cluster, which contains all instances, at the top and the individual instance clusters as leaf nodes.

Similar to the k-Means algorithm, the distance between instances must be computed to decide which clusters to merge or split. However, in hierarchical clustering, a large number of different distance metrics can be used, such as the (squared) Euclidean distance, the Manhattan distance, or the Hamming distance. Apart from the distance metric, however, hierarchical clustering also needs a measure to describe the dissimilarity between two clusters. For this purpose, so-called linkage criteria can be used, such as the maximum (complete-linkage), the minimum (single-linkage), or the average (average-linkage) of all pairwise distances between the two clusters or the distance between the centroids of the two clusters (centroid-linkage).

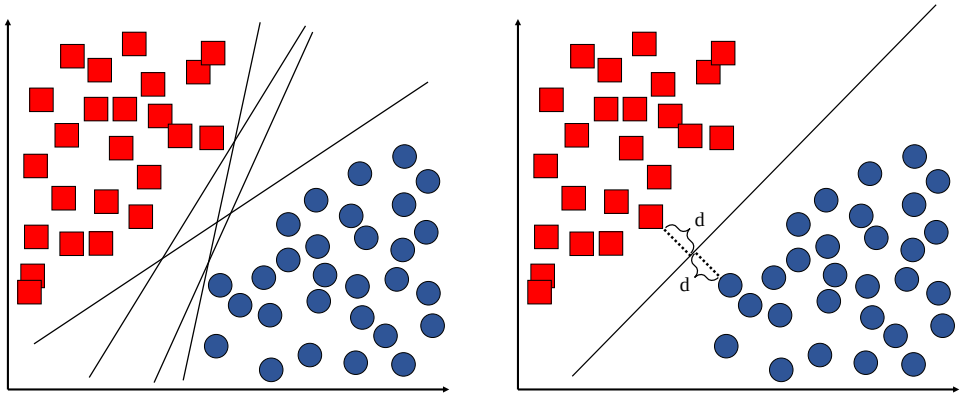
Once the dendrogram is generated, a cutting point must be defined to obtain the final clustering result. This can be done either by specifying a certain number of clusters or by defining a maximum distance. However, the latter case requires profound expert knowledge.

2.4.3 Support Vector Machine

Unlike the previously described clustering methods, *Support Vector Machines* (SVM) [CV95], which are based on the so-called *structural risk minimization* principle, are designed for classification tasks. Therefore, SVMs intend to transform the training data by a non-linear mapping into a higher dimensional

feature space that has a stronger generalization capability. For this non-linear mapping, the so-called *kernel trick* is applied. Once the input space is mapped to the higher dimensional feature space, a decision boundary is sought to distinguish between classes. In terms of SVMs, this decision boundary is referred to as a *hyperplane*. For a p-dimensional feature vector, an SVM tries to find an optimal hyperplane of dimension $p - 1$ that maximizes the separation between classes. Finally, training an SVM is equivalent to solving a linearly constrained quadratic optimization problem, which means that the solution is always unique and globally optimal.

An example of a two-dimensional feature space with two different target classes is shown in Figure 2.7. Figure 2.7a shows several potential hyperplanes that would linearly separate the two classes (namely the red squares and the blue circles) and, therefore, would yield the same accuracy on the training data. In fact, the number of potential hyperplanes is even infinite. However, to improve the generalization capability, an SVM optimizes the margins to the so-called *support vectors* of the two classes. A support vector is the instance with the shortest distance to the hyperplane. By maximizing this margin with respect to both classes, the same maximum margin d is obtained for the support vectors of both classes. Such an optimal hyperplane is shown in Figure 2.7b. Here, the red square and the blue circle that are connected to the hyperplane by a dashed line, which indicates the maximum margin d, represent the support vectors of the two classes.



(**a**) Feature space with several hyperplanes.

(**b**) Feature space with optimal hyperplane and maximum margin d.

Figure 2.7: Two-dimensional feature space containing multiple instances of two different classes.

In order to solve regression problems, an adaptation of SVM, the so-called *Support Vector Regression* (SVR), was developed [DBK+97]. However, in the following, we refer to this version also as SVM.

2.4.4 Random Forest

Random Forest is a machine learning method based on the concept of ensemble learning, which was first introduced by L. Breiman and A. Cutler [Bre01] and builds upon an earlier version of T. Ho [Ho95]. The general idea of ensemble learning is the concept of the "wisdom of the crowd". More specifically, ensemble methods combine a large number n of uncorrelated models to infer a final result. Typically, ensemble learners train numerous weak learners and aggregate their results to obtain a model with high predictive power, i.e., a strong learner. In this context, a weak learner is a classification method that correlates rather weakly with the true classification, while a strong learner correlates very well with the true classification. The two most common methods for generating and combining weak learners are *boosting* and *bagging*. While boosting aims at iteratively improving models by targeting instances that were misclassified in the previous iteration (cf. Section 2.4.5), bagging builds many individual models and derives a result by majority vote of the models' predictions [Bre96]. Random Forest falls into the category of bagging ensemble learners using decision trees as weak learners. Such a decision tree is a directed, acyclic graph in the form of a tree structure, where each node contains a decision rule and the leaves represent the final decision. After creating the set of decision trees, Random Forest combines the individual predictions of the decision trees to obtain a final prediction. This concept is illustrated in Figure 2.8.

By using the ensemble technique, Random Forest overcomes the main disadvantage of single decision trees, namely their tendency to overfit the training data and, as a result, to generalize rather poorly [Ho95]. In order to prevent overfitting, however, Random Forest must ensure that the decision trees do not correlate among each other. A first approach to achieve this goal was proposed by T. Ho [Ho95]. She suggested using only randomly selected features for model learning of the individual decision trees. L. Breiman and A. Cutler [Bre01] built upon this approach when they introduced the bagging approach. In bagging, not only are the features randomly selected, but also the training samples themselves. That is, for each of the n decision trees, a random subset of all the training data is sampled with replacement[2] and, then, the feature space of each sample is also randomly sampled.

[2]Sampling with replacement refers to the concept that instances in the same sample can be selected multiple times.

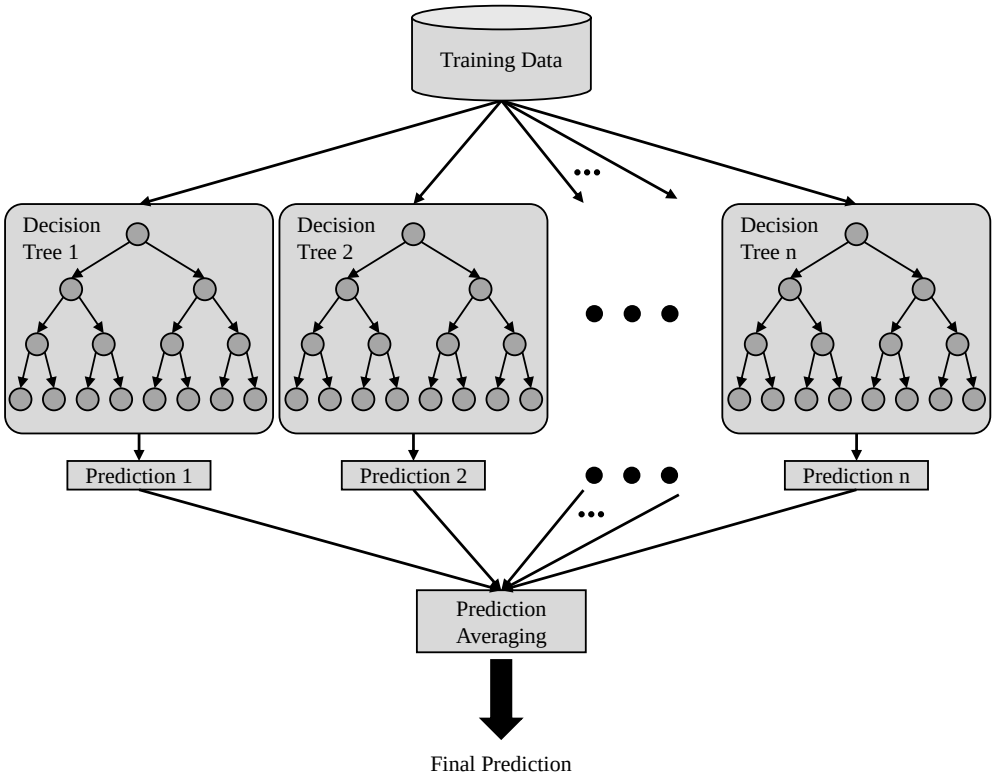


Figure 2.8: A schematic illustration of Random Forest consisting of n decision trees.

Random Forest can be employed for either classification or regression. With respect to a classification task, the final prediction result is the class with the most individual predictions of the decision trees. For regression, the predictions of the individual trees are combined using the arithmetic mean.

2.4.5 eXtreme Gradient Boosting

Similar to Random Forest, *eXtreme Gradient Boosting* (XGBoost) [CG16] also builds upon ensemble learning using decision trees. However, in contrast to Random Forest, XGBoost employs boosting instead of bagging. Generally speaking, XGBoost is a particular implementation of gradient boosted decision trees. The goal of XGBoost is fast execution speed and high model performance, both for classification and regression tasks.

Boosting as an ensemble learning method using decision trees was proposed by Y. Freund and R. Schapire [FS97]. The general idea is to fit many decision trees successively to re-weighted versions of the training data. That is, boosting methods train a weak learner, evaluate its accuracy on the training data, and, subsequently, adjust the weights of the incorrectly predicted instances in the training data to focus on those samples. Then, these re-weighted training data are used to learn another weak learner, which is also added to the ensemble. Hence, this approach is also called *additive training*. This iterative process is repeated several times, typically as long as adding new weak learners improves the overall prediction quality. The final result is obtained by a weighted average of the weak learners' predictions. Thus, the weight computation is based on the weak learner's accuracy during training.

In general, boosting can also be considered as an optimization problem. In terms of XGBoost, the cost function can be described as the sum of the training loss and the regularization term, which in turn describes the tree complexity. In other words, XGBoost penalizes large and complex trees, as they tend to overfit to the training data. When minimizing the cost function, the gradient descent algorithm is applied to determine which trees to add next.

2.5 Deep Learning Models

Similar to machine learning methods, deep learning models also require features as input, although the features can be the observations of the time series themselves. However, deep learning models typically require many more training instances to learn the relationship between the features and the respective target. In turn, deep learning methods can learn more complex relationships and require less manual feature engineering. Deep learning methods belong to the neural network family, especially to those with a large number of nodes and multiple layers. The challenge in applying deep learning models is to find a suitable network architecture and hyperparameter setting. To this date, many different types of neural networks, such as Feed-Forward Neural Networks, Convolutional Neural Networks, and Recurrent Neural Networks have been proposed and new types are being introduced continuously. Compared to the other types of neural networks, the advantage of Recurrent Neural Networks is that they can explicitly model and learn time dependence.

This section provides a brief introduction to the neural network types employed in this thesis. However, this serves only as a brief overview of neural network types. For more information on deep learning methods, we refer the reader to the book by I. Goodfellow *et al.* [GBCB16].

2.5.1 Feed-Forward Neural Network

The *Feed-Forward Neural Network* (FFNN) is a simple neural network architecture that aims at detecting patterns in data, learning from past observations, and providing generalized results with respect to current knowledge. A key advantage of FFNNs is their ability to capture non-linear patterns without making assumptions about the statistical distribution of observations. The general architecture of an FFNN is shown in Figure 2.9. Such an FFNN consists of an input layer, one or more hidden layers, and an output layer. The main property of a *fully-connected* FFNN is that every node in one layer is connected to every node in the subsequent layer. Note that each connection has a weight assigned to it. The FFNN shown in Figure 2.9 consists of n input nodes, z hidden layers, each consisting of m_i, $i = 1, \ldots, z$, neurons, and l output neurons.

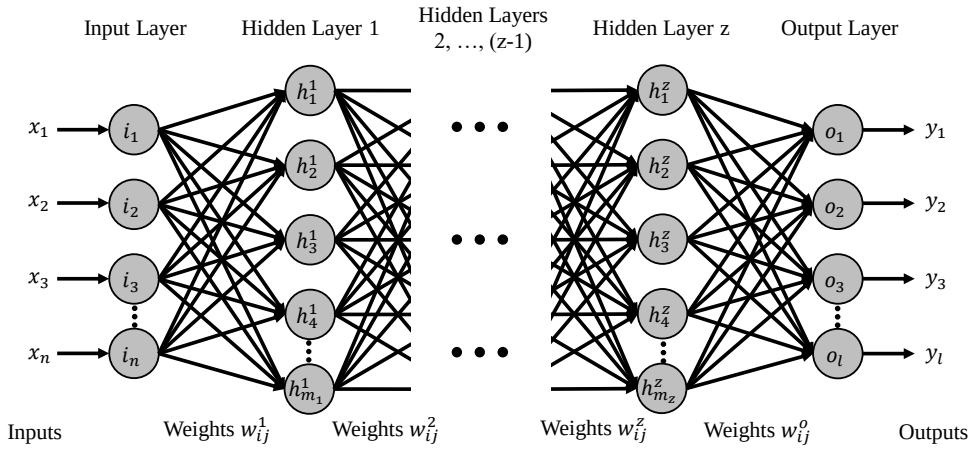


Figure 2.9: The simplified architecture of a fully-connected Feed-Forward Neural Network with z hidden layers.

First, the FFNN receives the input values, namely the features $x_1, x_2, \ldots, x_n$, which are passed to the input neurons $i_1, i_2, \ldots, i_n$. This input is then processed further by the network. For this purpose, the neurons in the first hidden layer $h_1^1, h_2^1, \ldots, h_{m_1}^1$ are first computed by multiplying the input values by the weight w_{ij}^1 of the respective connection and adding a bias value b_j^1. Here, w_{ij}^1 denotes the weights toward the first hidden layer from input neuron i to neuron h_j^1. Formally, the value of h_j^1 is computed as

$$h_j^1 = f\left(b_j^1 + \sum_{i=1}^{n} w_{ij}^1 \times x_i\right), \tag{2.25}$$

where $f(y)$ denotes the applied activation function. The activation function is an essential component of neural networks, as it allows learning of non-linear relationships. However, this requires a non-linear activation function. In general, there are many different activation functions, such as identity, logistic (also known as sigmoid), hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU). As these are the most commonly used activation functions, Table 2.1 contains their formulas, with e denoting Euler's constant.

Table 2.1: Formulas for commonly used activation functions in neural networks.

Name	Formula
Identity	$f(y) = y$
Logistic	$f(y) = \frac{1}{1+e^{-y}}$
Hyperbolic tangent	$f(y) = \tanh(y) = \frac{e^{y}-e^{-y}}{e^{y}+e^{-y}}$
Rectified Linear Unit	$f(y) = \max\{0, y\}$

Next, the information is passed through the hidden layers. To this end, the j-th neuron of the h-th hidden layer is computed as

$$h_j^h = f\left(b_j^h + \sum_{i=1}^{m_{(h-1)}} w_{ij}^h \times h_i^{(h-1)}\right). \tag{2.26}$$

Finally, the value of the j-th output neuron is computed as

$$o_k = f\left(b_j^z + \sum_{i=1}^{m_z} w_{ij}^o \times h_i^z\right). \tag{2.27}$$

As can be seen from the formulas, the learning goal of an FFNN is to update the biases of the neurons b_j^h and the weights w_{ij}^h, $h = 1, \ldots, z$, as well as w_{ij}^o. For this purpose, the *backpropagation* algorithm [RHW86] is typically applied, since it allows learning the dependencies between arbitrary sets of input combinations. Therefore, the gradient of loss is propagated backwards through the network to adjust the weight of each connection. The FFNN repeats this process of input prediction and weight updating until a certain predefined tolerance level is reached or a maximum number of iterations is exceeded.

2.5.2 Recurrent Neural Network

As opposed to Feed-Forward Neural Networks, Recurrent Neural Networks (RNNs) allow so-called *feedback connections*. The most commonly used feedback

connection is direct feedback, where the output of a neuron is linked to its own input. Thus, the output of a previous time step can be used as additional input to the same neuron. Such a direct feedback connection is depicted on the left-hand side of Figure 2.10. The output of the neuron can be considered as internal hidden state that is propagated through all time steps to include information from previous time steps while new data is being received. For this reason, RNNs are specifically tailored to sequential data, such as sentences or time series. The structure of RNNs is composed of so-called *cells*, as shown by the gray box on the left-hand side of Figure 2.10. To better understand the training of RNNs, this is typically visualized over time. That is, the RNN is unfolded over time. The right-hand side of Figure 2.10 illustrates such an unfolding of an RNN with direct feedback for t time steps. To this end, the first input x_1 is initially passed to the RNN and yields the output y_1. The derived hidden state h_1 is then passed to the next time step, where it is used in combination with the new input x_2 to derive the next output y_2. This process continues until the last input x_t is processed in combination with h_{t-1} to derive the last output y_t. However, the major drawback of standard RNNs are the vanishing and exploding gradient problems [KK01]. Concerning the vanishing gradient, during training, gradients decrease drastically from layer to layer, which can cause the weights to stop changing. On the contrary, if the gradients are too large, their multiplication causes superlinear growth, leading to an unstable network and eventually to too large values, which are finally manifested in NaN values. For this reason, RNNs can only model short-term temporal dependencies.

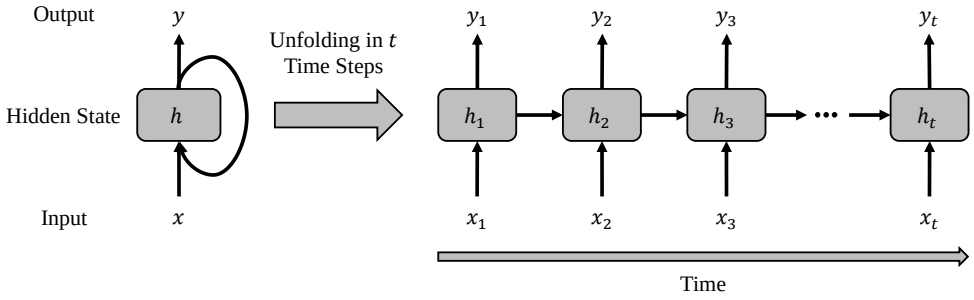


Figure 2.10: A simplified illustration of a Recurrent Neural Network with direct feedback connection unfolded over t time steps.

The most commonly used variations of the standard RNN are the Long Short-Term Memory networks (LSTM) and the Gated Recurrent Units (GRU). In contrast to the standard RNN, these are more capable of learning long-term temporal dependencies via so-called *gates*. That is, these variants of the

standard RNN solve the vanishing and exploding gradient problems [HS97], which is the reason for their popularity. While LSTM uses three gates, namely input, output, and forget gate, GRU uses only two gates, namely reset and update gate. The gates of LSTM and GRU are used to decide which information should be memorized over time and which information should be forgotten. As LSTM involves more gates and parameters, it requires significantly more memory consumption and runtime compared with GRU, but also typically provides higher predictive power. Finally, for training LSTMs and GRUs, the *backpropagation-through-time* [Wer88] algorithm is typically employed, which can be considered a generalization of the standard backpropagation algorithm.

2.6 Evaluation Measures for Forecasting and Classification

In order to assess the quality of time series forecasts and regressions as well as classification tasks, we briefly introduce common evaluation measures. To this end, Section 2.6.1 provides an overview of error measures for time series forecasting. Note that these methods can also be used for regression results. Next, Section 2.6.2 presents commonly used prediction quality measures for binary and multi-class classification.

2.6.1 Forecast Error Measures

A number of measures have been proposed to assess the accuracy of forecasting methods. However, in time series forecasting, these measures represent the error instead of the accuracy. Hence, the lower the value of the measure, the better the forecast. This section is based on the work of R. Hyndman and A. Koehler [HK06], who group forecast error measures into four categories, namely scale-dependent measures, measures based on percentage errors, measures based on relative errors, and scaled error measures. However, we first introduce terms and notations necessary for the following subsections. For this purpose, Table 2.2 summarizes the most important notations.

The evaluation of forecast errors on the forecast horizon, i.e., the future values unknown during training, is also referred to as *out-of-sample* evaluation, while the evaluation of forecast errors on the historical training data, i.e., the historical values known during training, is referred to as *in-sample* evaluation.

2.6.1.1 Scale-dependent Error Measures

The first group of forecast error measures is formed by the *scale-dependent error measures*. As the name implies, the value of the scale-dependent error measures

Table 2.2: Notations for forecast error measures.

Notation	Meaning
n	Length of the time series
h	Length of the forecast horizon
X_t	Observation at time t
$\mathbf{X}$	Vector of observations in the forecast horizon
F_t	Forecast at time t
$\mathbf{F}$	Vector of forecasts in the forecast horizon
$e_t = X_t - F_t$	Forecast error at time t
$e_t^* = X_t - F_t^*$	Forecast error of a baseline method at time t
$p_t = \frac{e_t}{X_t} \cdot 100\%$	Percentage forecast error at time t
$r_t = \frac{e_t}{e_t^*}$	Relative forecast error at time t
$q_t = \frac{(n-1)\, e_t}{\sum_{i=2}^{n} \lvert X_i - X_{i-1} \rvert}$	Scaled forecast error at time t

is highly affected by the scale of the time series. Therefore, these measures should not be used to compare forecasts for multiple time series with different scales. However, scale-dependent measures can be used to compare forecasts from different methods for a single time series. Well-known scale-dependent error measures are the mean absolute error (MAE), which is defined by

$$\mathrm{MAE}\,(\mathbf{X}, \mathbf{F}) = \frac{1}{h} \sum_{t=1}^{h} |e_t|, \tag{2.28}$$

and the root mean square error (RMSE), which is defined by

$$\mathrm{RMSE}\,(\mathbf{X}, \mathbf{F}) = \sqrt{\frac{1}{h} \sum_{t=1}^{h} e_t^2}. \tag{2.29}$$

2.6.1.2 Measures based on Percentage Errors

As the name already reveals, *measures based on percentage errors* use the percentage error instead of the absolute error. Due to this error normalization, these error measures can also be used to compare forecast errors across different time

series with different scales. A commonly used error measure from this group is the mean absolute percentage error (MAPE), which is defined as

$$\text{MAPE}\,(\mathbf{X}, \mathbf{F}) = \frac{1}{h} \sum_{t=1}^{h} |p_t|. \tag{2.30}$$

However, the major drawback of measures based on percentage errors is the assumption of a meaningful zero. As can be seen in Table 2.2, the percentage error is defined as the ratio between the forecast error and the actual observation multiplied by 100%. Thus, the percentage error is undefined if the time series has any zeros in the forecast horizon. In addition, values close to zero also cause bias, as this leads to a highly skewed distribution. A second shortcoming of most measures based on percentage errors is that they penalize overestimates harder than underestimates. This also applies to the example of MAPE. To this end, an alternative version of the MAPE was introduced, the so-called symmetric MAPE (sMAPE), defined as

$$\text{sMAPE}\,(\mathbf{X}, \mathbf{F}) = \frac{2}{h} \sum_{t=1}^{h} \frac{|e_t|}{|X_t| + |F_t|}. \tag{2.31}$$

Nevertheless, sMAPE does not completely solve the drawback of MAPE, since sMAPE penalizes lower forecasts more severely than higher forecasts.

2.6.1.3 Measures based on Relative Errors

While the first two groups of forecast error measures consider only the error produced by the forecasting method under examination, the *measures based on relative errors* incorporate a baseline forecasting method and put the forecast errors in relation to each other. A typical baseline is presented by the Naïve model. As a consequence, measures based on relative errors are easy to interpret and are independent of the scale of the time series. However, when the baseline method performs very well, i.e., the error of the baseline method is close to zero, the value of the measure tends to explode. The most commonly used measure based on relative errors is the mean relative absolute error (MRAE), which is defined as

$$\text{MRAE}\,(\mathbf{X}, \mathbf{F}) = \frac{1}{h} \sum_{t=1}^{h} |r_t|. \tag{2.32}$$

2.6.1.4 Scaled Error Measures

The fourth and final group of forecast error measures is represented by the *scaled error measures*. Unlike the measures based on relative errors, scaled error measures normalize the forecast error by the development behavior of the actual training time series, i.e., by the derivative of the actual training time series. Therefore, scaled error measures are also independent of the scale of the input data. However, since the forecast error is scaled with respect to the in-sample MAE from the Naïve forecast, the scaled error measure is undefined if the time series has zero variance, i.e., if all values in the time series are equal. A common choice for a scaled error measure is the mean absolute scaled error (MASE), which is defined as

$$\text{MASE}\left(\mathbf{X}, \mathbf{F}\right) = \frac{1}{h} \sum_{t=1}^{h} |q_t|. \tag{2.33}$$

2.6.1.5 Coefficient of Determination

A special case among the evaluation measures for forecasting is the *coefficient of determination*. In contrast to the forecast error measures described above, the coefficient of determination R^2 indicates to what extent the forecast agrees with the actual values. Therefore, a higher R^2-score denotes a better forecast, where R^2 exhibits a range of $(-\infty, 1]$. Hence, the coefficient of determination is strictly speaking a quality measure rather than an error measure. Mathematically, the coefficient of determination is defined by

$$R^2\left(\mathbf{X}, \mathbf{F}\right) = 1 - \frac{\sum_{t=1}^{h} e_t^2}{\sum_{t=1}^{h} (X_i - \mu_X)^2}, \tag{2.34}$$

with μ_X denoting the mean of the time series observations. Thus, the squared forecast error is normalized by the squared forecast error of forecasting the mean value for the entire forecast horizon, which relates to measures based on relative errors (cf. Section 2.6.1.3). However, the first term inverts the error measure to a kind of quality measure.

2.6.2 Classification Quality Measures

Various measures can be applied to evaluate the quality of classification models, with different measures capturing different capabilities of the model. Here, we briefly describe the most relevant classification quality measures, which are also used in this thesis. However, in Table 2.3, we first introduce general notations necessary for the further understanding.

Table 2.3: Notations for classification quality measures.

Notation	Meaning
TP	Number of instances correctly predicted as positive
TN	Number of instances correctly predicted as negative
FP	Number of instances falsely predicted as positive
FN	Number of instances falsely predicted as negative

2.6.2.1 Measures for Binary Classification

In a binary classification problem, typically one class is labeled positive while the other class is labeled negative. Thus, the predictions can be presented in a so-called *confusion matrix*. An exemplary confusion matrix for such a binary classification is given in Table 2.4. Here, the rows represent the predicted class, while the columns show the actual observed class.

Table 2.4: An example confusion matrix for binary classification.

Predicted \ Observed	Positive Class	Negative Class
Positive Class	TP	FP
Negative Class	FN	TN

Based on TP, TN, FP, and FN, the following measures can be calculated to characterize the prediction quality.

- *Accuracy*: The accuracy provides the ratio between the number of correctly predicted instances and the total number of predicted instances. Formally, the accuracy is defined as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{2.35}$$

  That is, the sum of the values on the principal diagonal of the confusion matrix is divided by the total sum of all entries in the confusion matrix. Although the accuracy is easy to interpret, it also has a clear drawback. In case the different classes are not equally present in the test set, a high accuracy can be achieved by just predicting the majority class. Therefore, other quality measures are necessary in combination with accuracy.

- *Precision*: The precision indicates the ratio of correctness when predicting the positive class. That is, the precision is defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.36}$$

- *Recall*: In contrast to precision, the recall specifies how many of the actual positive instances were detected. Mathematically, this is expressed as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.37}$$

- *F1-score*: As precision and recall are both very important measures for prediction quality, they are combined in the F1-score using the harmonic mean. Formally, the F1-score is defined by

$$\text{F1-score} = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}. \tag{2.38}$$

- *Matthew's correlation coefficient*: The Matthew's correlation coefficient (MCC) [Mat75] is a kind of correlation coefficient that analyzes the relation between the predicted and observed classes and, therefore, can be used even if the number of instances per class is highly diverse. The MCC ranges from -1 to 1, where a negative value indicates disagreement and a value close to one indicates a perfect agreement. With an MCC close to zero, the prediction quality is similar to a random choice classifier. Mathematically, the MCC is defined by

$$\text{MCC} = \frac{(\text{TP} \cdot \text{TN}) - (\text{FP} \cdot \text{FN})}{\sqrt{(\text{TP} + \text{FP}) \cdot (\text{TP} + \text{FN}) \cdot (\text{TN} + \text{FP}) \cdot (\text{TN} + \text{FN})}}. \tag{2.39}$$

- *Cohen's kappa*: Contrary to the measures presented above, Cohen's kappa is not directly defined by the confusion matrix, but measures the quality of a predictor by comparing its predictions to a random choice predictor based on class frequencies. Therefore, Cohen's kappa does not suffer from class imbalances. Formally, Cohen's kappa is defined by

$$\text{Kappa} = \frac{p_o - p_e}{1 - p_e}, \tag{2.40}$$

  where p_o denotes the accuracy of the learned predictor and p_e represents the expected accuracy of a random choice predictor that uses the class distribution of the training set as the sampling frequency. According to J. Landis and G. Koch [LK77], the kappa value can be interpreted in the steps shown in Table 2.5.

Table 2.5: Interpretation of Cohen's kappa according to J. Landis and G. Koch [LK77].

Kappa	Interpretation
Kappa < 0	Poor agreement
$0 \leq$ Kappa ≤ 0.20	Slight agreement
$0.21 <$ Kappa ≤ 0.40	Fair agreement
$0.41 <$ Kappa ≤ 0.60	Moderate agreement
$0.61 <$ Kappa ≤ 0.80	Substantial agreement
$0.81 <$ Kappa	(Almost) perfect agreement

2.6.2.2 Measures for Multi-Class Classification

In the case of multiple classes, the definition of a fixed positive class and a fixed negative class is no longer possible. However, some quality measures, such as accuracy and Cohen's kappa, can still be calculated in the same way. The accuracy is still the sum of the principal diagonal entries divided by the total sum of all entries in the confusion matrix. Given that kappa builds upon the accuracy, this measure can also be calculated in the same way for multiple classes. For the other measures, the binary measures must be computed class-wise. Accordingly, each class is considered as the positive class, while all other classes are considered as a single negative class. A schematic confusion matrix for a classification task consisting of four different classes is shown in Table 2.6. Here, TP, TN, FP, and FN are denoted for class 3, where $\text{TN} = \sum_i \text{TN}_i$, $\text{FP} = \sum_j \text{FP}_j$, and $\text{FN} = \sum_k \text{FN}_k$. Once the particular measure has been calculated for each class, the results must be aggregated into a single value. For this purpose, there exist three aggregation methods:

- *Macro*: When combining the measure X calculated for each class at the macro level, the simple arithmetic mean is calculated. Therefore, the

Table 2.6: A schematic confusion matrix for a four classes classification with notations for class 3.

Observed / Predicted	Class 1 (negative)	Class 2 (negative)	Class 3 (positive)	Class 4 (negative)
Class 1 (negative)	TN_1	TN_4	FN_1	TN_7
Class 2 (negative)	TN_2	TN_5	FN_2	TN_8
Class 3 (positive)	FP_1	FP_2	TP	FP_3
Class 4 (negative)	TN_3	TN_6	FN_3	TN_9

macro aggregation is defined as

$$\text{macro } X = \frac{1}{c} \sum_{p=1}^{c} X_p, \tag{2.41}$$

where X denotes the measure, X_p the class-wise value of measure X for class p, and c the total number of classes.

- *Weighted*: In contrast to macro aggregation, weighted aggregation considers the sample size of each class. That is, the weighted mean is calculated, with the weights equal to the respective class size n_p. Mathematically, the weighted aggregation is defined by

$$\text{weighted } X = \frac{1}{\sum_{p=1}^{c} n_p} \sum_{p=1}^{c} n_p \cdot X_p. \tag{2.42}$$

- *Micro*: Unlike the other two methods, micro averaging considers all classes at once. That is, TN is equal to TP and FN is equal to FP, resulting in the same values for micro precision and micro recall. Therefore, the micro F1-score is also equivalent to micro precision and micro recall. Finally, all three micro measures are equal to the overall accuracy.

Chapter 3

State-of-the-Art

In order to present the state-of-the-art in the areas covered in this thesis, we organize related literature into two sets. First, we categorize approaches in the literature on hybrid time series forecasting into three groups, explain each group in general, and present representative contributions for each of the three groups in Section 3.1. Subsequently, we introduce several contributions to the general domain of critical event prediction in Section 3.2. Again, we divide this section into three subsections for different aspects of critical event prediction.

3.1 Hybrid Time Series Forecasting Methods

The "*No Free Lunch Theorems*" were proposed by D. Wolpert and W. Macready in 1995 and 1997 and were originally intended for search [WM95] and optimization algorithms [WM97], respectively. In general, the "No Free Lunch Theorem" for optimization algorithms states that there cannot exist an optimization algorithm that outperforms all other optimization algorithms for all use cases. Instead, tailoring an optimization algorithm to meet the requirements of a particular scenario inevitably leads to a degradation in quality for another scenario. Although this theorem was formulated for optimization algorithms, it can be applied to many other areas of mathematics and computer science as well. One such area of research is time series forecasting. To transfer the "No Free Lunch Theorem" for optimization algorithms to time series forecasting, it can be inferred that there is no single forecasting method that achieves the highest accuracy for all time series, regardless of its particular properties. In order to reduce the impact of this "No Free Lunch Theorem" in time series forecasting, many hybrid forecasting methods have been presented in the literature. In this thesis, we refer to hybrid forecasting methods as forecasting methods that combine at least two individual forecasting methods in some form. Thus, the goal of such hybrid forecasting methods is to compensate for the drawbacks of one forecasting method by incorporating another or even multiple forecasting methods that exhibit their strengths specifically in the

weaknesses of the others. To examine the state-of-the-art in hybrid forecasting, we conducted a broad literature review and categorized the approaches into three groups: (I) weighted forecasting method ensembles, (II) forecasting method recommendation, and (III) component-based forecasting.

3.1.1 Weighted Forecasting Method Ensembles

The idea of weighted ensembles of forecasting methods was first introduced in 1969 by J. Bates and C. Granger [BG69] and is therefore historically the first group of hybrid time series forecasting methods. More precisely, the final forecast is generated by a linear combination of forecasts derived from at least two different forecasting methods. For this purpose, a weighting must be assigned to each forecast. Since this weighting process is the core component of such ensemble forecasting methods, J. Bates and C. Granger proposed several methods for weight calculation and compared the accuracy of the resulting forecasts with the accuracy of the individual forecasting methods included in the ensemble. J. Bates and C. Granger have shown that forecast accuracy can be increased by using weighted ensembles, because the ensembles reduce risk, and thus variance, compared to relying only on a single forecasting method.

Twenty years after the proposal of weighted forecasting method ensembles, R. Clemen examined numerous algorithms for weight computation and found that simple approaches, such as equal weights, are not inferior to more sophisticated approaches [Cle89]. In particular, he showed that correlation-based weight calculations should be avoided, since these approaches worsened the forecast accuracy in the conducted study. Finally, he concluded that ensembles are particularly useful when the forecasts of the different methods vary highly.

A further investigation of different weight calculation algorithms was conducted by L. de Menezes *et al.* [DMBT00]. Besides evaluating different weight calculation approaches, L. de Menezes *et al.* provided guidelines on which weight calculation method to choose with respect to the training observations of the time series. To this end, the authors summarized that the so-called outperformance method [Bun75] should be preferred for small training sizes, while for medium training sizes with low correlation, they suggested the so-called optimal method with independence assumption. For large training sizes, they recommended the methods called optimal and constrained regression-based model. L. de Menezes *et al.*, however, also pointed out that equal weights, equivalent to the arithmetic mean, also perform well and should be used if the time series exhibit positive correlation and similar error variances.

R. Adhikari *et al.* [AVK15] proposed an ensemble forecasting method that does not combine all available forecasting methods, but only a certain subset

that is assumed to perform well for the time series under consideration. Therefore, the available time series observations are split into an in-sample training and an in-sample validation part. In a first step, the forecasting models are trained on the in-sample training data only, while their accuracies are calculated on the in-sample validation part. To this end, the forecast error measure mean scaled error is applied. Based on this error measure, the forecasting models are ranked from best to worst. Finally, only the n top ranked forecast models are included in the final ensemble model used for out-of-sample forecasting. The individual forecasts of the selected models are then weighted by the inverse of their error measure on the in-sample validation part. Using a set of nine different forecasting methods and a data set consisting of four time series, R. Adhikari *et al.* showed that this method outperformed all individual forecasting methods as well as the combination of all forecasting models.

Another novel approach to weight estimation for ensemble forecasting was proposed by M. Sommer *et al.* [SSH16]. The authors stated that systems are typically subject to change and, therefore, the systems must deal with unforeseen new situations. Thus, the authors' goal was to automate weight estimation during runtime and keep system designers out of the update loop. To this end, M. Sommer *et al.* proposed the use of the eXtended Classifier System for Function approximation (XCSF) as weight estimation method. XCSF is an evolutionary rule-based machine learning method specialized for function approximation using a combination of gradient-based local learning and a steady-state niche genetic algorithm. Thus, XCSF partitions the problem space into several subspaces and learns linear approximations of the objective function within the corresponding niches. With respect to time series forecasting, XCSF computes coefficients for each potential forecasting method. As individual forecasting methods within the ensemble approach, M. Sommer *et al.* selected different parametrizations of ARIMA (i.e., ARIMA$(0, 0, 1)$, ARIMA$(0, 2, 2)$, and ARIMA$(1, 0, 1)$), ETS, and Random Walk with Drift. Finally, the authors compared their approach on ten time series against three different weight estimation algorithms and the individual methods themselves. The results showed that the performance of the XCSF weight estimation approach depends on the forecast error measure. With respect to both comparisons, namely the individual methods and the other weight estimation methods, the XCSF performed best for some forecast error measures and mediocre or even worst for others. Lastly, the authors concluded that an expanded set of potential forecasting methods tended to improve the forecast accuracy in almost all cases.

Y. Wang *et al.* [WZT+18] introduced an ensemble approach for combining probabilistic load forecasting methods. To this end, the authors proposed a

Constrained Quantile Regression Averaging (CQRA) method in which the CQRA parameters were estimated using a linear program aimed at minimizing the flipper loss, while the CQRA parameters were required to be non-negative and sum to one. More specifically, Y. Wang *et al.* applied three different quantile regression methods, namely Quantile Regression Neural Network, Quantile Regression Random Forests, and Quantile Regression Gradient Boosting. Then, the weights for the combination of these three models were determined by solving an optimization problem. To estimate the quality of the different forecasting methods for out-of-sample forecasting, Y. Wang *et al.* also used an initial subset of the training data for model training, the next part for model validation, and the last part for model testing. These performances of the different methods were then used in the optimization problem to derive suitable weights for the forecasting methods. Finally, the forecasting methods were applied to provide the out-of-sample forecast and their forecasts were aggregated using the weighted average with respect to the weights obtained by the optimization problem. The authors evaluated their proposed CQRA approach on two data sets, showing that their approach reduced the pinball score compared to the best individual probabilistic load forecast. In addition, it also outperformed nine ensemble methods, such as weighted averaging and several quantile regression averaging methods.

A weighted ensemble forecasting approach specifically designed for multi-step-ahead forecasting was proposed by A. Galicia *et al.* [GTLT+19]. In their ensemble, the authors included three machine learning methods, namely Decision Tree, Gradient Boosted Trees, and Random Forest. To estimate the weights for each of these methods, A. Galicia *et al.* used the weighted least square method. However, for h-step-ahead forecasting, each value in the horizon was modeled by a particular model that receives the same inputs, but a different target. This technique results in h models for each forecasting method, which in turn yields weights not only for each method but also for each value in the forecast horizon. The authors explicitly rejected multi-step-ahead forecasting based on rolling history and horizon, claiming that this approach performed worse due to error potentiation. In addition to the static weight estimation approach, which uses the same weights for all time series in the test set, the authors also implemented a dynamic alternative that allows the weights to be updated during runtime. A. Galicia *et al.* compared both ensemble methods to the individual methods with respect to the mean relative error and concluded that both ensemble methods outperformed the individual methods, while the dynamic update also achieved better forecasts than the static weights. Finally, the authors conducted a second experiment to demonstrate the superiority

of the ensemble methods compared to other advanced forecasting methods, namely an Artificial Neural Network approach, a Pattern Sequence-based forecasting approach, and another deep learning approach.

The so-called FFORMA (Feature-based Forecast Model Averaging) was proposed by P. Montero-Manso *et al.* [MMAHT20] and uses a meta-learning approach to estimate the weights of the forecasting methods in the ensemble. Therefore, a large training database of diverse time series is crucial. In a first step, 42 time series characteristics are computed for each time series in the training database. Then, all forecasting methods in the ensemble pool are fitted to the time series individually. For this purpose, the authors included nine time series forecasting methods, namely the Naïve and sNaïve models, Random Walk with Drift, Theta [AN00], ARIMA, ETS, TBATS, NNetAR, and a model based on STL decomposition and autoregression. Then, FFORMA applies a machine learning method, namely XGBoost, to learn the dependencies between the time series characteristics and a proper set of weights for aggregating the forecasts of the different models. In their experiments, P. Montero-Manso *et al.* showed that FFORMA outperforms both the simple arithmetic mean as ensemble method as well as the state-of-the-art individual forecasting methods.

3.1.2 Forecasting Method Recommendation

The second category of hybrid time series forecasting methods deals with the recommendation of certain methods based on quantifiable characteristics of time series. To this end, a rule set is typically generated, which can either be created manually, i.e., based on expert knowledge, or learned automatically.

First approaches toward the recommendation of time series forecasting methods were based on expert knowledge. Therefore, these systems are also referred to as expert systems. Such an expert system was proposed by F. Collopy and J. Armstrong [CA92], in which they integrated four forecasting methods, namely Random Walk, regression, Brown's linear exponential smoothing, and Holt's exponential smoothing. To provide guidelines for method selection, they calculated 18 time series characteristics and derived a rule set consisting of 99 rules in total. Finally, the authors compared the forecast accuracy of their method selection guidelines with simple arithmetic averaging of the forecasting models included in the expert system and showed that their guidelines outperformed the ensemble approach. Although M. Adya *et al.* [ACAK01] introduced an approach to reduce the amount of human intervention required, the proposed expert system was not yet fully automated.

To further automatize the forecasting method recommendation, the second subset of these methods relies on algorithms that derive rules in terms of a

learning objective. For the application of such algorithms, the database must be divided into a training set and a test set. While the algorithms are applied to the training set, the test set is subsequently used to assess the quality of the forecasting method recommendations.

The methodology for deriving recommendation rules for time series forecasting methods automatically was first proposed by B. Arinze [Ari94]. The general procedure consists of five steps. First, a set of possible forecasting methods must be defined, followed by a set of time series characteristics. In the third step, for each time series, a set of features, i.e., the time series characteristics, and a target, i.e., the forecasting method with the highest accuracy, are combined. Then, a rule induction tree is generated based on the previously combined instances. Finally, the selected characteristics are computed for new time series and the respective rules are evaluated. In this work and an incremental work [AKA97], the authors determined six time series characteristics to derive rules for six different forecasting methods. The authors concluded that the recommendation system already yielded very promising results, although the approach was still rather exploratory.

R. Prudêncio and T. Ludermir [PL04] examined two different rule learning-based approaches for forecasting method recommendation and evaluated them in two separate case studies. The first approach was based on seven different time series characteristics and included only two forecasting methods, namely Simple Exponential Smoothing and Time-Delay Neural Network. To infer selection rules from the time series characteristics, the authors used a variation of the well-known C4.5 algorithm (i.e., the J.48 algorithm). With respect to their first case study, the authors showed that this approach improved the forecast accuracy compared to using either of the two individual methods, although the best method was suggested in only about 63% of all cases. For their second case study, R. Prudêncio and T. Ludermir applied another meta-learner to not only select the presumed best method, but to create a complete ranking from best to worst. For this purpose, they employed an adaptation of the NOEMON [KT99] design, using Multi-Layer Perceptrons for the classification. In contrast to the first case study, the authors selected five time series characteristics and the forecasting methods Random Walk, Holt's linear exponential smoothing, and autoregression for the second case study. Again, the results indicated that the recommendation approach outperformed all individual methods.

Two further approaches have been proposed by X. Wang *et al.* [WSMH09]. In their paper, the authors applied clustering and rule learning algorithms to derive categorical and quantitative rules, respectively. Therefore, the authors calculated nine different time series characteristics. In addition, four of them

were also extracted from the de-trended and de-seasonalized time series, resulting in a set of 13 time series features. The time series forecasting methods chosen were ARIMA, ETS, NNetAR, and Random Walk. Their first approach to generating judgmental and conceptual rules used hierarchical clustering and Self-Organizing Maps to group similar time series together and separate dissimilar time series. However, as their second approach aimed at generating quantitative rules, which is more similar to the other approaches in the field of recommending forecasting methods, the authors used the C4.5 algorithm to learn rules that match the extracted time series characteristics with the presumed best forecasting method. This was done by sorting the four forecasting methods for each time series with respect to their achieved forecast accuracy. However, for each time series, only the best forecasting method was assigned class label `1`, while the other forecasting methods were assigned class label `0`. These classes were then used as targets for the rule learning algorithm C4.5. This design results in a rule set that indicates whether or not a forecasting method should be selected for a time series given its particular time series characteristics. X. Wang *et al.* provided all the necessary parameter settings to reconstruct the C4.5 rule learning algorithm as well as the rules they derived. Unfortunately, however, they did not evaluate the forecast accuracy achieved by their conceptual and qualitative rules.

In their work [LG10], C. Lemke and B. Gabrys introduced a different characteristics set compared to X. Wang *et al.* and examined the applicability of different rule generation approaches. Although several time series characteristics were similar to X. Wang *et al.*, they also removed certain characteristics and introduced several others. To this end, C. Lemke and B. Gabrys divided the time series characteristics into general characteristics, frequency domain characteristics, autocorrelation characteristics, and diversity characteristics. First, the authors used this set of time series characteristics to learn a decision tree. The decision tree was chosen because it can be interpreted as a readable set of rules. Thus, the authors provided a set of guidelines for selecting forecasting methods. Second, C. Lemke and B. Gabrys applied three different rule generation methods, namely Feed-Forward Neural Network, Decision Tree, and Support Vector Machine. In this step, the authors applied a technique called zoomed ranking. In this technique, the time series characteristics are first computed and then they are grouped using k-Means clustering. Then, for a new time series, the time series in the cluster with the most similar time series characteristics with respect to a given distance metric are selected for further analysis. Subsequently, a ranking of these time series is created by means of an adaptation of the adjust ratio of ratios, i.e., the achieved forecast error measures

of the two forecasting methods are divided. Finally, the method that has the best rank is selected. The experiments performed by C. Lemke and B. Gabrys suggest that the ensemble forecasting methods studied provided more robust forecasts than individual forecasting methods.

M. Kück *et al.* [KCF16] proposed a forecasting method recommendation approach that considers not only typical time series characteristics but also error-based meta-features. To this end, the authors introduced a novel error-based feature that splits the training time series into an in-sample training part and an in-sample validation part. While the in-sample training part is used to adjust the forecasting method, the adjusted method is applied to the in-sample validation part to generate the forecast. The forecast error is calculated using sMAPE rolling over the entire horizon. In total, M. Kück *et al.* calculated 127 statistical, model-based, and error-based features. In addition, the authors compared several feature sets against each other. As forecasting methods, the approach included only exponential smoothing methods, namely a simple exponential model, a seasonal exponential model, a seasonal-trend exponential model, and a trend exponential model. Then, the authors learned a Multi-Layer Perceptron to select the presumed best forecasting method based on the particular feature set. For this purpose, the problem was modeled as a multi-class classification task. The evaluation shows that the error-based features improved the forecast accuracy and that the Multi-Layer Perceptron-based forecasting method selection outperformed Random Walk, selecting one of the individual methods, and using the best of the four models with respect to the training or validation accuracy.

Although not peer-reviewed, T. Talagala *et al.* have uploaded a technical report introducing their FFORMS (Feature-based Forecast Model Selection) approach [THA18]. In their set of potential forecasting methods, the authors included 15 different methods, namely white noise, ARMA, ARIMA, Random Walk with and without Drift, Theta, six ETS models, STL-AR, sARIMA, and sNaïve. However, not all forecasting methods were considered for all data types. Thus, expert knowledge is required to a certain degree. As features, the authors selected 25 time series characteristics for non-seasonal time series and 30 time series characteristics for seasonal time series. In addition to the previous approaches, FFORMS includes a time series augmentation step. More precisely, ETS and ARIMA models are fitted to the time series in the training data set to generate similar time series, thus increasing the number of training instances. To automatically select the presumably best forecasting method, T. Talagala *et al.* used Random Forest in terms of multi-class classification. To evaluate FFORMS, they compared the forecast accuracy with all individual

forecasting methods included in FFORMS. The results showed that FFORMS often outperformed the individual methods. Yet, in some cases, the individual methods also outperformed FFORMS.

3.1.3 Component-based Forecasting

The last category of hybrid forecasting methods explicitly leverages the strengths of individual forecasting methods for particular fields by applying different forecasting methods to specific parts of the time series. In general, these component-based forecasting methods can be further divided into two groups. The first group of approaches applies multiple forecasting methods sequentially to the residuals of the previous method. Therefore, it is crucial for these methods that the subsequent forecasting methods have different advantages compared to the previous forecasting methods. Otherwise, the subsequent methods would not be able to extract additional information from the residuals of the previous method. In contrast, the second group of component-based forecasting methods employs time series decomposition directly in advance. Then, a particular forecasting method is chosen for each time series component.

G. Zhang [Zha03] proposed an approach belonging to the first category. That is, the author combined ARIMA with a neural network because he wanted to compensate for the main shortcoming of ARIMA, which, according to the author, is the assumption of a linear form of the model. In contrast, Feed-Forward Neural Networks are capable of fitting complex non-linear patterns. However, Feed-Forward Neural Networks tend to overfit to the training data. Therefore, G. Zhang proposed a sequential application of both methods to take advantage of their individual benefits, while compensating for the drawbacks. First, an ARIMA model was fitted to the original time series to model the linear component of the time series. Then, the residuals between the ARIMA model and the original time series observations were calculated. These data were then used to train the Feed-Forward Neural Network. Thus, the basic assumption of this approach is that the residuals contain only non-linear patterns and random errors. To evaluate the model, G. Zhang applied the proposed hybrid model and the two individual methods to three time series. With respect to the mean square error and mean absolute deviation, the results have shown that the hybrid model significantly improved the forecast accuracy compared to the ARIMA and Feed Forward Neural Network models on their own.

A very similar approach to G. Zhang [Zha03] was presented by P. Pai and C. Lin [PL05]. The only considerable difference is the machine learning method chosen. Instead of the Feed-Forward Neural Network used by G. Zhang, the authors of this paper used a Support Vector Machine. Besides this adaptation,

the approach follows the procedure of G. Zhang. To evaluate the proposed hybrid model, the authors compared their approach with the two individual methods and a combination of them. In agreement with G. Zhang, the authors also found that the sequential application of ARIMA and SVM improved the forecast accuracy compared to the individual methods and the simple combination. In addition, the authors pointed out that the hyperparameters of the methods must be tuned in combination rather than model by model.

Another approach to the sequential application of forecasting methods was proposed by R. Adhikari and R. Agrawal [AA14]. Their hybrid forecasting method was developed particularly for forecasting financial time series. The key characteristic of financial time series is that they exhibit remarkably random behavior when considered as univariate time series. Therefore, the authors introduced a combination of Random Walk and an ensemble consisting of a Feed-Forward Neural Network and an Elman Neural Network, which is a special type of Recurrent Neural Networks. Similar to the previous approaches, the statistical forecasting method, i.e., Random Walk, was used to model the linear part of the time series, while the neural network ensemble was applied to learn the non-linear component. The neural network ensemble was aggregated using the simple arithmetic mean. To evaluate their model, the authors used four financial time series and calculated the mean absolute error, mean square error, and symmetric mean absolute percentage error for their proposed hybrid approach as well as for all three individual methods. The results showed that the hybrid approach significantly outperformed the individual methods.

In contrast to these component-based forecasting approaches that apply the methods sequentially to the residuals, other component-based approaches explicitly decompose the time series into components and apply one forecasting method to each component. One such approach was proposed by A. Conejo *et al.* [CPEM05]. The authors applied wavelet decomposition and, subsequently, used ARIMA models to forecast the components. More specifically, their proposed approach was tripartite. In the first step, the time series was split into four components using wavelet transform. This decomposition is performed to reduce the complexity in each time series. Then, an ARIMA model was fitted to each component and, consequently, used to forecast the respective component. Thus, this approach does not explicitly use different forecasting methods, but different parametrizations of one and the same method. However, ARIMA models also cover a wide range of potential models (cf. Section 2.3.4). In the third and final step, the inverse wavelet transform was applied to reconstruct the final forecast by recombining the forecasts of each component time series. In order to evaluate this hybrid approach, A. Conejo *et al.* conducted a case

study consisting of one time series divided into four successive time series. Thus, the method was applied to the four resulting time series and compared with individual ARIMA and Naïve models. The authors pointed out that their hybrid method not only reduced the average forecast error, but also reduced the variance in the forecast error.

Another approach in the same domain was proposed by J. Xie *et al.* [XCCP06], who used time series decomposition together with neural networks. In their work, the authors integrated dynamic unit, specialized units, and multiple-neural networks into Adaptive Time-Delay Neural Networks (ATNN). That is, the proposed neural network architecture, called Dynamic Spline ATNN (DSTNN), contained two types of decomposition units. The first type of decomposition units extracts the seasonal behavior of the time series, while the second type focuses on the trend part. To extract the seasonal information, the approach included a maximum entropy-based spectral analysis technique. For trend estimation, the authors calculated a moving average combined with a polynomial fit. In addition to these decomposition units, the DSTNN also included a dynamic spline interpolation unit based on cubic polynomial spline interpolation. Finally, the authors examined the forecast accuracy of their approach on a single time series using the root mean square error and the mean absolute error as forecast error measures. Compared to a conventional Time-Delay Neural Network, the results demonstrated that the DSTNN approach improved the forecast accuracy for a multi-step-ahead forecast with respect to both forecast error measures.

S. Schlüter and C. Deuschle [SD10] conducted a broad case study evaluating numerous wavelet transforms and comparing their advantages with respect to forecast accuracy with several individual time series forecasting methods. The authors considered the main advantage of wavelet transforms for time series forecasting to be their decomposition into a time-dependent sum of frequency components. These components can capture seasonal patterns with time-varying intensity as well as time-varying period length quite well. Although these frequencies are easier to forecast, the overall complexity of the model nevertheless increases when integrating wavelet decomposition into the forecasting workflow. The authors justified this statement by arguing that an appropriate wavelet function must be selected and the parameters of this wavelet function must be defined, which in turn increases the number of points of error. Therefore, the authors investigated whether or not the trade-off between improved forecast accuracy and increased model complexity in terms of wavelet transforms is worthwhile. To examine the performance of wavelet transforms in the area of time series forecasting, S. Schlüter and C. Deuschle

applied four different wavelet functions to four time series and compared their performance with ARMA, ARIMA with the integration order set to 1, and Census X-12. The authors evaluated both one-step-ahead and multi-step-ahead forecast accuracy using mean square error, mean absolute deviation, and mean absolute percent error. The authors concluded that there is no best combination of wavelet transform and time series forecasting techniques for all time series. Although in most cases there was a combination that outperformed each method, this best combination varied greatly between time series.

N. Liu *et al.* presented a hybrid component-based time series forecasting approach [LTZ+14] based on Empirical Mode Decomposition, Extended Kalman Filter, Extreme Learning Machine with Kernel, and Particle Swarm Optimization. To obtain the individual time series components, the authors applied Empirical Mode Decomposition, resulting in a set of Intrinsic Mode Function components. Then, the authors used either Extended Kalman Filter or Extreme Learning Machine with Kernel on each of the Intrinsic Mode Function components. In addition, they optimized the hyperparameters of the model using Particle Swarm Optimization. As this approach was specifically designed for on-line forecasting, this hyperparameter optimization was performed in an initial off-line step, while there were also periodic hyperparameter updates during runtime. The authors also carried out a case study demonstrating that the hybrid approach provided good forecast accuracy and runtime.

A further component-based approach to time series forecasting was proposed by C. Di *et al.* [DYW14]. Their approach was specifically designed for hydrological time series due to their non-linear, non-stationary, and multiscale characteristics. In general, the proposed approach consisted of four steps. First, the raw time series was denoised using Empirical Mode Decomposition. Then, the denoised time series was decomposed using Ensemble Empirical Mode Decomposition, resulting in a set of Intrinsic Mode Function components and a residual component. In a third step, all components were forecast using a modified version of a Radial Basis Function Neural Network (RBFNN). In the final step, the authors learned a Linear Neural Network (LNN) to combine all forecasts of the Intrinsic Mode Function components and the residual component. C. Di *et al.* evaluated their approach on six time series with respect to mean relative error, mean absolute error, and root mean square error. To this end, the authors compared their approach with individual time series forecasting methods, namely RBFNN and ARIMA, their proposed hybrid approach either without denoising or without decomposition, and the application of other methods within the four-stage approach, i.e., wavelet transforms instead of Ensemble Empirical Mode Decomposition, ARIMA instead of RBFNN, and sim-

ple addition instead of LNN. The results revealed that the proposed approach using all four steps and the particular methods described above performed best for all six time series with respect to all three forecast error measures.

C. Bergmeir *et al.* [BHB16] proposed another approach based on bootstrap aggregation of exponential smoothing models. More precisely, the approach consisted of five steps. First, Box-Cox transformation was applied to the time series to remove multiplicative effects (cf. Section 2.2.4). Subsequently, the transformed time series was decomposed into season, trend, and remainder components using STL decomposition. Then, the core process of the approach was performed. That is, the remainder was bootstrapped to construct a large number of time series that exhibit the same trend and seasonal behavior as the original time series. Although the bootstrapped time series exhibit a similar random behavior, it is not identical. To this end, the authors sampled continuous fixed-length sequences from the original remainder and sequentially assembled them into a new remainder. Then, the seasonal and trend components were combined with the newly created remainders and the inverse Box-Cox transformation was applied. After augmenting the original time series with this set of bootstrapped time series, an ensemble of exponential smoothing models was fitted to these time series. Finally, the forecasts from this ensemble were aggregated using their median to obtain the final forecast for the original time series. In order to evaluate their bootstrap approach, C. Bergmeir *et al.* applied the hybrid model to the M3 data set [MH00] and compared it to a variety of forecasting methods. Although ETS is known to perform very well on the M3 data set, the bootstrap approach outperformed ETS as well as many other forecasting methods. In fact, on certain subsets of the M3 data set, the bootstrap approach even performed best.

3.2 Critical Event Prediction

This section provides an overview of existing approaches to critical event prediction. Therefore, we first present approaches based on time series forecasting methods and, subsequently, describe model-based approaches that rely on multivariate monitoring data and do not involve explicit time series forecasting tasks. Note that some approaches in the first category also apply multivariate models after forecasting each time series in the multivariate data set. Finally, we present several approaches that focus on re-training machine learning models during runtime. For a general overview of methods for predicting critical events in terms of failures, we refer the reader to the comprehensive overview by F. Salfner *et al.* [SLM10]. The planning of countermeasures is hardly addressed

here, since this thesis focuses on the prediction of critical events, while the planning of appropriate countermeasures is beyond the scope of this thesis.

3.2.1 Critical Event Prediction using Time Series Forecasting

S. Oe *et al.* [OSN80] proposed an approach to critical event prediction of lathe tools in terms of increased flank wear. That is, the authors modeled critical events of machine equipment, i.e., their transition from a normal state to an anomalous failure state, as a degradation process described by a time series of acoustic monitoring data. Using this time series, an autoregression model of order p was trained, where p was estimated using Akaike's Information Criterion. To detect degradation, the authors compared the actual observations with modeled observations of a normal machine condition. Next, the authors applied four measures indicating the difference between the models and, therefore, the probability of a faulty machine condition. Their model included two thresholds, with the first threshold implying a warning and the second a severe failure. Finally, they applied their model to real-world acoustic monitoring data of a lathe tool and proved the effectiveness of their model.

J. Hellerstein *et al.* [HZS99] introduced an approach to predict critical events in the form of threshold violations for HTTP requests. To this end, the authors modeled non-stationary and stationary parts of the workload time series using multiple forecasting models, each capturing a different relevant part of the time series. For evaluating their approach, J. Hellerstein *et al.* used data captured from a production web server and concluded that the proposed approach was able to predict threshold violations well if the prediction horizon was not too long and the actual values were not too close to the threshold.

A similar approach was suggested by R. Vilalta *et al.* [VAH+02]. In addition to regression-based long-term forecasts of performance measures and categorical target prediction of specific events, the authors presented short-term prediction of threshold violations. Similar to the work of J. Hellerstein *et al.* [HZS99], R. Vilalta *et al.* used workload data describing HTTP requests arriving at a web server. R. Vilalta *et al.* also applied different forecasting models for different parts of the time series and employed an autoregression model of order two (i.e., AR(2)) for temporal dependencies. Furthermore, the authors included breakpoint detection by means of the Generalized Likelihood Ratio algorithm.

R. Sahoo *et al.* [SOR+03] analyzed event logs and system performance counters of a 350-node cluster system monitored for one year. First, the authors filtered the monitoring data to remove redundant or misleading information. Then, R. Sahoo *et al.* applied numerous time series forecasting methods to estimate future values of critical system performance counters, such as system

utilization, idle time, or network I/O. The investigated forecasting methods include mean, multi-step-ahead Naïve forecast, i.e., forecasting the last observed value for the entire horizon, mean of a pre-defined window of the last observations, autoregression, moving average, and autoregressive moving average. The results showed that the multi-step-ahead Naïve forecast performed best, because performance counters often change only slowly. Moreover, the ARMA model performed poorly with little training data, but improved when the size of the training data was increased. Furthermore, the authors applied a rule-based classification algorithm to predict rare critical events captured in the event logs. Here, the results showed that the approach was able to predict these critical events with an accuracy of up to 70%.

An adaptation of the well-known ARIMA model was introduced by W. Wu *et al.* [WHZ07] with the objective of forecasting the development of a machine's state of health. The authors stated that the main shortcoming of ARIMA is its poor performance for long-term forecasting. Therefore, the authors proposed an adapted version of ARIMA that remedies this shortcoming by reducing the accumulation of systematic errors within ARMA models and, consequently, within ARIMA models for multi-step-ahead forecasting. In addition, their approach automatically updated the parameters of the improved ARIMA model during runtime to allow for adjustments as time series characteristics change. Finally, the authors evaluated their model on real-world vibration data of a rotating machine. To this end, W. Wu *et al.* calculated the root mean square of the vibration data as an indicator of the current health status of the machine. The authors showed that their adapted ARIMA version achieved lower forecast errors than the original ARIMA model for the 50-step-ahead forecast on two time series with respect to the forecast error measures root mean square error, mean absolute error, and mean absolute percentage error.

J. Zhao *et al.* [ZXL07] proposed a modification of the ARMA model to forecast machine downtime using transformed monitoring data. That is, the monitoring data were transformed into an indicator variable for machine downtime. The derived time series was then preprocessed using a moving average with window size eight. The authors state that this step was performed to increase the robustness of the time series and to handle non-linear and non-stationary time series. Next, the authors applied an ARMA model to the residuals of the preprocessed time series. Based on the values obtained by the sample autocorrelation function and the partial autocorrelation function, the authors selected an AR, MA, or ARMA model to forecast the time series. To this end, the parameters of the models were estimated using Akaike's Information Criterion and Bayesian Information Criterion. In addition, J. Zhao *et al.* verified whether

the residuals resulting from the application of the model to the historical data were white noise or still had autocorrelation patterns. The final model was used to forecast the indicator variable for machine downtime. To evaluate the modified ARMA model, the authors used a time series obtained from a semiconductor ATM factory. The modified ARMA model was compared with an ordinary ARMA model, where the results showed that the proposed modification outperformed the ordinary ARMA model. However, a comparison with ARIMA is missing, although ARIMA addresses the same shortcoming of ARMA models as the proposed approach.

A critical event detection model for failure detection of point mechanisms in railroad networks was described by F. Garcia *et al.* [GPR10]. The general idea of their approach was to model the normal behavior of a point mechanism and compare it with the actually observed behavior. To this end, the authors applied Vector Autoregressive Moving Average (VARMA), which can be viewed as a modification of ARMA models for multivariate time series, to forecast the time required for the next movement of the point mechanism along with its 95% confidence interval using the last 50 fault-free executions to learn the VARMA model. They estimated the parameters of the AR and MA submodels using multivariate autocorrelation and multivariate partial autocorrelation. In addition, the authors manually differenced the time series with order one to achieve stationarity. After forecasting the assumed time of movement, they used multiple Harmonic Regression models for different regions of the 95% confidence interval to forecast the signal. The authors then selected the forecast with least variation. The final step of failure detection was the comparison of the actually observed values with the predicted values. F. Garcia *et al.* demonstrated the applicability of their model using real-world data from an operating point mechanism at a British railroad junction.

I. Lahyani *et al.* [LMC12] presented an approach to predict Quality of Service degradations in publish/subscribe networks. For this purpose, the authors employed an autoregression to forecast the latency between two adjacent brokers. Based on exponentially weighted moving average, the authors calculated thresholds for the Quality of Service. Using the forecasts provided by the autoregression model and the calculated thresholds, Quality of Service degradations were assumed when the forecast exceeded the threshold. I. Lahyani *et al.* evaluated the quality of their approach using simulation data and concluded that their approach predicted the simulated Quality of Service degradations well with an accuracy of 75% and a recall of almost 86%.

T. Chalermarrewong *et al.* [CAS12] described the application of ARMA and Fault Tree Analysis for the prediction of hardware failures in data centers. To

this end, the authors used ARMA to forecast future values of system performance counters, such as memory utilization, processor utilization, and processor temperature. In addition, their model included automatic re-training of the ARMA model. Then, thresholds were used to convert the ARMA forecasts into binary variables, which in turn were used as input for the Fault Tree Analysis. In combination with the multi-step-ahead forecasts of the ARMA model, the Fault Tree was utilized to estimate the time remaining until the failure occurs and to suggest possible migration actions. A cluster simulation was used by the authors to evaluate their presented model. The results have shown that the model was able to predict the failures with an accuracy of 97%. However, the model also produced many false positives, resulting in a precision of only 53%.

In contrast, W. Liao *et al.* [LWP12] focused on estimating and forecasting a machine condition index to minimize the long-term costs in terms of maintenance activities and operating costs. To estimate the machine condition index, Principal Component Analysis (PCA) was carried out first to extract features and reduce the dimensionality of the data. Subsequently, the dominant feature was clustered to reveal different patterns in the data. Then, chi-square test was performed to determine the health index of the considered machine. To estimate the future development of the machine's health index, an ARMA was employed. When the health index of the machine exceeded a pre-defined threshold, a critical event was assumed and a maintenance action was triggered. Experimental results based on drilling machine data indicated that the model was able to reproduce the actual degradation process very well.

C. Zhao and F. Gao [ZG15] introduced a three-stage approach to failure prediction. First, they extracted critical fault effects using a combined relative analysis based on Principal Component Analysis. A fault index was constructed by means of Mahalanobis distance between the considered data and the training data of the normal condition, allowing identification of diverging patterns. Finally, a Vector Autoregressive (VAR) model was applied to forecast the identified error effects. For this purpose, a threshold value of the fault index was set as a critical event alert limit. C. Zhang and F. Gao evaluated the prediction performance of their model in two case studies, where they also analyzed the influence of the prediction horizon on the mean absolute deviation. With respect to their case studies, the authors demonstrated the effectiveness of their approach, although they did not compare their model with other approaches.

Another ARMA-based approach to critical event prediction was proposed by M. Baptista *et al.* [BSdM$^+$18]. They focused on predicting failures for aircraft engines using ARMA models in combination with data-driven methods. For this purpose, the approach required a time series of past failure events as input.

Based on this time series, the ARMA model was trained to forecast the time of the next failure event. In addition, various statistical features were calculated on the time series of failure events. Then, Principal Component Analysis was applied to the set of both features to transform the feature set into a new feature set consisting of linearly uncorrelated features. Using this PCA feature set, five different data-driven regression methods were trained, namely k-Nearest Neighbor, Random Forest, Neural Network, Support Vector Regression, and Generalized Linear Regression. To assess the proposed model, the authors used a case study of real-world aircraft engine data. To rank the data-driven methods, the authors calculated eight forecast error measures and concluded that Support Vector Regression performed the best on average, although the performance of each method was highly dependent on the error measure considered. Furthermore, the authors showed that their approach was superior compared to typical failure prediction based on the Weibull distribution.

M. Narayan and A. Fey [NF20] introduced an approach for predicting critical events in robotic forces. The authors consider the main drawback of state-of-the-art forecasting methods to be the high runtime for model learning. Therefore, the authors presented a fast forecasting approach for forces. For this purpose, the authors employed Pseudo Partial Derivative to transform the non-linearity within a time series into a single time-varying scalar. After this transformation, a dynamic linear model was used to forecast future values. However, this model is only capable of forecasting one-step-ahead. Based on the resulting forecasts, critical events can be detected. In order to investigate the quality of their proposed model, the authors used simulation data and compared their model with ARIMA using mean square error as forecast error measure. The results revealed that the proposed model outperformed ARIMA by 11% in terms of prediction error and exhibited a much shorter maximum runtime.

An approach to predict critical events for smart manufacturing was presented by K. Villalobos *et al.* [VSI21]. Their approach included two main steps, namely, forecasting sensor measurements and analyzing whether or not the sensor measurement forecasts are anomalous. As time series forecasting methods, the authors evaluated ARIMA, Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM). Their experimental results suggested that on their use cases, the ARIMA model seemed superior for short forecasting horizons, while the LSTM model outperformed the ARIMA model for longer forecasting horizons. Therefore, the authors chose LSTM as the forecasting method for their model. Furthermore, the authors implemented concept drift detection for their forecasting model. That is, the model observed the forecast error over time and triggered a re-learning of the model if the forecast error

exceeded a certain threshold. The analysis of the forecasts was performed using Residual Neural Networks. K. Villalobos *et al.* evaluated their model for three critical event types and created a specific Residual Neural Network for each critical event type. In their case study, only temperature sensor values were recorded from a real-world production plant. The first critical event type was only a fixed threshold value of a particular temperature sensor. Therefore, the proposed model was able to achieve an area under the receiver operating characteristic curve (AUC-ROC) of 99.9%. The second critical event type could have also been modeled using thresholds, although multiple thresholds would have been required. Again, the proposed model achieved a very high AUC-ROC of 99.2%. Finally, the third critical event type could have not been modeled using thresholds. Nevertheless, the model of K. Villalobos *et al.* resulted in an AUC-ROC value of 97.3%.

3.2.2 Critical Event Prediction using Multivariate Learning Models

One category of models that can be used to detect failures in machines are statistical models. N. Gebraeel [Geb06], for instance, focused on analyzing the degradation of components based on vibration data to predict the health of machines. He proposed a stochastic degradation modeling framework to model the remaining life of already partially degraded equipment. However, the model focused only on exponentially degrading components. In addition, N. Gebraeel presented two sensory update techniques. To evaluate the proposed statistical approach, he conducted experiments with real-world vibration data from rolling element thrust bearings. The results showed the strength of the degradation models updated with the proposed sensory update techniques.

H. Cai *et al.* [CJF+20] proposed a so-called similarity matching procedure using the kernel two sample test to find the most similar instances in the training data set. Using this statistical matching, they provided an estimate of the remaining useful life of the examined machine along with its probability distribution by using Weibull analysis. The authors used this to provide a confidence interval on top of the actual prediction. In order to demonstrate the advantages of their proposed model, H. Cai *et al.* used a publicly available data set of aircraft engines.

Typically, however, recent approaches use machine learning or deep learning approaches for machine failure detection and prediction. Y. Liang *et al.* [LZXS07] presented a failure prediction workflow for event logs from a supercomputer, namely the IBM BlueGene/L. First, the authors transformed the event logs into data interpretable for classification algorithms. Next, the authors applied three common classification algorithms, namely a rule-based

classifier (i.e., RIPPER), a Support Vector Machine, and k-Nearest Neighbor. Furthermore, the authors also implemented a modified version of the k-Nearest Neighbor algorithm. The authors compared the performance of the different algorithms with respect to recall, precision, and F1-score. While all algorithms performed well at a prediction horizon of 12 hours, the prediction quality dropped drastically for shorter prediction horizons. Nevertheless, the modified k-Nearest Neighbor algorithm achieved the best results. For the 12-hour prediction horizon, the modified k-Nearest Neighbor algorithm provided an F1-score, precision, and recall of about 70%, 60%, and 80%, respectively.

J. Sikorska *et al.* [SHM11] conducted a thorough review of different types of models for predicting the remaining useful life of technical equipment. Therefore, the authors described four categories of classification models, namely knowledge-based models, life expectancy models, artificial neural networks, and physical models. For each of these models, the authors summarized a list of benefits and drawbacks with respect to several criteria, resulting in a set of manually generated guidelines for selecting remaining useful lifetime prediction models for both industrial practitioners and researchers. The main criteria considered by J. Sikorska *et al.* were the size and quality of the available training data, the impact of noise, the type of maintenance, whether multiple failure patterns can be modeled in parallel, and whether the model can handle novel failure patterns. Finally, the authors pointed out many further selection criteria, such as the mathematical knowledge of the operator.

Z. Tian [Tia12] proposed a time-to-failure prediction approach for bearings using a Feed-Forward Neural Network. To this end, Z. Tian introduced a fitting function based on the Weibull distribution to transform the historical monitoring data into a shape that exhibits less noise effects. After applying this function to the input data of the Feed-Forward Neural Network, the neural network predicted the health of the machine in terms of lifetime percentage. The input data of the Feed-Forward Neural Network was the age of the bearing along with several monitoring features. However, not only the observations of the current measurement were passed to the Feed-Forward Neural Network, but also those of the previous measurement. In this way, deterioration between successive measurements was supposed to be better detected. The Feed-Forward Neural Network proposed by Z. Tian consisted of two hidden layers with three and two nodes in the first and second hidden layer, respectively. A single output node was used, whose value was interpreted as the lifetime percentage. The proposed approach was evaluated using real-world vibration data from pump bearings. A comparison with another Feed-Forward Neural Network showed that the proposed model performed superior.

An approach tailored to remaining useful life prediction of lithium-ion batteries was introduced by Q. Miao *et al.* [MXC+13]. Their approach extended the standard Particle filter algorithm by an additional preprocessing step. In a first step, the authors applied the unscented Kalman filter to estimate the proposal distribution for retrieving the posterior probability. Subsequently, they applied the standard Particle filter to obtain the prediction. Therefore, they named their modified Particle filter algorithm unscented Particle filter. The authors used their proposed approach to predict the remaining useful lifetime of a lithium-ion battery in terms of cycles and compared the results with the standard Particle filter algorithm with respect to percentage error and root mean square error. The evaluation indicated that the unscented Particle filter outperformed the standard Particle filter and that the prediction quality of both methods improved with increasing degradation of the battery.

T. Pitakrat *et al.* [PVHG13] compared numerous machine learning methods in terms of their hard disk drive failure prediction performance. For this purpose, they used a data set consisting of S.M.A.R.T. measurements of 369 hard disk drives. This data set was first introduced by J. Murray *et al.* [MHKD05]. In order to detect not only currently failed hard disk drives, but also to predict future failures, the authors marked all instances as "failed" that failed within the next seven days, while all others were marked as "healthy". Based on the S.M.A.R.T. feature set and the derived failure targets, the authors trained 21 machine learning methods as binary classifiers. To compare the binary classifiers, the authors considered not only the predictive power of the classification models, but also their required runtime for training and prediction. As error measures, the authors chose precision, recall, false positive rate, accuracy, and F1-score. Finally, the authors concluded that there was no significantly superior method, but that the selection of an appropriate machine learning method depends on the requirements and constraints of the particular application.

The application of Feed-Forward Neural Networks for hard disk drive failure prediction, by contrast, was proposed by B. Zhu *et al.* [ZWL+13]. Their network consisted of a single hidden layer and used backpropagation for weight updating. As input, the authors passed both the normalized and raw S.M.A.R.T. measurements to the network. In addition, the authors also calculated the change rates of the S.M.A.R.T. features and used them as additional input to their Feed-Forward Neural Network. Moreover, the authors also trained a Support Vector Machine with the same set of features. Their evaluation data set included 23,395 hard disk drives, with less than 2% of the devices failing. For predicting hard disk drive failures, B. Zhu *et al.* compared the prediction performance for different prediction horizons, with all models trained as bi-

nary classifiers. The results showed that the Support Vector Machine achieved the lowest false alarm rate. However, the failure detection rate of the Feed-Forward Neural Network was significantly better compared to the Support Vector Machine, while it also maintained a low false alarm rate.

G. Susto *et al.* [SSP+14] proposed an approach to the field of predictive maintenance. The proposed approach relied on multiple classification models with varying prediction horizons to reduce the total operation costs. To this end, each classifier was trained as a binary classification model where all instances that suffered a failure within the defined prediction horizon were marked as failed. Thus, the only difference between the classifiers was the length of the prediction horizon. Next, the predictions were analyzed with respect to the total operating costs incurred. These costs included both the costs of unexpected failures and the costs of underused machine life. Finally, the proposed approach triggered maintenance when the classifier that achieved the lowest total cost in the decision system predicted an imminent failure. As machine learning methods, the authors compared the performance of Support Vector Machine and k-Nearest Neighbor. In order to assess their approach, G. Susto *et al.* applied a Monte Carlo simulation based on data from tungsten filaments used in ion implantation. The results showed that the proposed approach using Support Vector Machines as multiple classifiers outperformed a single Support Vector Machine and also achieved less total costs and fewer unexpected failures compared to the proposed approach using k-Nearest Neighbor classifiers.

A framework for generating machine learning-based time-to-failure predictions for software systems was introduced by A. Pellegrini *et al.* [PDSA15]. As their framework assumed only system-level features, namely the number of active threads combined with memory, swap space, and CPU features, their framework was application-independent. To reduce the dimensionality of the feature space, the authors incorporated a feature selection mechanism based on Lasso regularization. For the prediction of time-to-failure, the six machine learning methods linear regression, M5P decision tree, REP-Tree, Lasso predictor, Support Vector Machine, and Least-Square Support Vector Machine were integrated. Furthermore, the framework collected four error measures as well as two measures regarding the runtime required for model learning. For the evaluation of their framework, the authors gathered monitoring data using two virtual machines. As a basis, the e-commerce benchmark TPC-W was chosen. Finally, the authors concluded that the decision tree-based methods M5P and REP-Tree yielded the best results for their particular case study.

C. Xu *et al.* [XWL+16] predicted the health status of hard disk drives using a Recurrent Neural Network. Their network architecture used a single hidden

layer and employed backpropagation for weight optimization. In addition, the authors manually selected relevant features through reverse arrangement test, rank-sum test, and Z-scores. The authors applied their Recurrent Neural Network to three data sets and compared its prediction quality with Hidden Markov Models, Binary Neural Networks, Classification Trees, Multi-Class Neural Networks, and Conditional Random Fields. The results indicated that the proposed Recurrent Neural Network outperformed the other models with respect to failure detection rate and false alarm rate.

The work of M. Botezatu *et al.* [BGBW16] also dealt with the prediction of hard disk drive failures. To this end, the authors first conducted feature selection using statistical hypothesis testing, assuming that the time series of feature measurements should exhibit a permanent change prior to the occurrence of the failure. Subsequently, they applied exponential smoothing to flatten the feature time series. As the classes were highly imbalanced, the authors also performed informative downsampling to balance the number of classes. To predict the degradation of hard disk drives, the authors tested several machine learning models, namely Decision Tree, Gradient Boosted Decision Tree, Logistic Regression, Random Forest, Regularized Greedy Forest, and Support Vector Machine. Finally, the authors also introduced a transfer learning technique to apply a model learned on a particular hard disk drive model to another model. For the evaluation, the authors used a data set of more than 30,000 hard disk drives collected over a 17-month period and predicted whether the hard disk drive will fail the next day. The prediction quality was evaluated in terms of precision, recall, and F1-score. Their results showed that Regularized Greedy Forest provided the best predictions with an F1-score of up to 98%, even though Gradient Boosted Decision Trees provided comparable results. In addition, M. Botezatu *et al.* demonstrated the importance of transfer learning when different hard disk drive models are present in the test set than in the training set.

Y. Lei *et al.* [LLG+16] focused on predicting the remaining useful lifetime of rotating machines and presented a novel two-stage approach for this purpose. First, the authors introduced a novel health indicator, namely the weighted minimum quantization error. This health indicator was built upon the minimum quantization error introduced by H. Qiu *et al.* [QLLY03] by adding a weighting step. To compute the health indicator, the authors first calculated numerous typical features and derived the so-called trendability by calculating the Spearman correlation coefficient of the features with time. Features with a trendability below a certain threshold were discarded, while the others were grouped using correlation clustering to remove redundant features. The remaining features were then fused by means of a Self-Organizing Map. However,

the Self-Organizing Map was trained using only data from healthy machine states. Finally, the weighted minimum quantization error was calculated as the difference between the features of a healthy model and the actual features, with higher weights given to the features with high trendability. The second stage of the proposed approach employed this health indicator to model the degradation process and predict the remaining useful lifetime. The modeling part was performed using the Paris-Erdogan model, while a Particle filter was applied for the prediction. Here, the parameters of the models were estimated using maximum likelihood estimation. The authors evaluated their proposed approach on the publicly available PRONOSTIA data set, which consists of several rolling element bearings. As base-level features, the authors calculated 28 features from the time domain, the time-frequency domain, and based on trigonometric functions. Finally, Y. Lei *et al.* compared the percentage error of their proposed approach with two other approaches that used the same data set. The authors concluded that their proposed approach outperformed the other two in terms of achieving the smallest percentage error as well as the smallest variation in percentage error.

Similar to Y. Lei *et al.*, L. Guo *et al.* [GLJ+17] also proposed a remaining useful lifetime prediction approach specifically tailored to bearings. The overall procedure of their approach also followed that of Y. Lei *et al.* That is, the authors calculated various features from the raw measurements and selected only the most relevant for further processing. Thereby, the authors used six related similarity features from the time and frequency domains as well as eight statistical features from the time-frequency domain. The feature selection was based on the correlation with time and the monotonicity of the particular feature. However, unlike Y. Lei *et al.*, L. Guo *et al.* did not explicitly derive a health indicator, but instead passed all remaining features to a Long Short-Term Memory neural network. The authors evaluated the percentage error of their proposed model on the PRONOSTIA data set and in an industrial case study of wind turbine analysis. For both evaluation scenarios, the authors compared their proposed approach with an approach employing a Self-Organizing Map instead of an LSTM. Here, L. Guo *et al.* found that their approach considerably outperformed the Self-Organizing Map for both applications.

J. Li *et al.* [LSW+17] described the application of two tree-based machine learning methods for hard disk drive failure prediction. The models employed were Decision Trees and Gradient Boosted Regression Trees. While the first method directly classified hard disk drives as "good" or "failed',' the second method provided a health score between 0 and 1. To derive an output class, the authors set a threshold of 0.2 and marked instances as "failed" if the Gradient

Boosted Regression Trees predicted a value below that threshold. As input, the authors passed S.M.A.R.T. features to the models. However, they first selected the most relevant features using quantile functions. That is, the authors visualized the quantile functions of each S.M.A.R.T. feature for healthy and failed hard disk drives and manually selected strong discriminator features. For the evaluation of the tree-based models trained with the manually selected S.M.A.R.T. features, the authors used three different data sets consisting of a total of 121,698 hard disk drives. As prediction measures, J. Li *et al.* chose failure detection rate, false alarm rate, and time in advance. They also compared their tree-based models with the Feed-Forward Neural Network using backpropagation by B. Zhu *et al.* [ZWL+13]. The result indicated that both tree-based models were superior to the neural network and that the Decision Tree-based model best predicted the hard disk drive failures with a failure detection rate of about 93%, while it maintained a false alarm rate of less than 0.01%. Finally, the authors analyzed the potential cost reduction of integrating their hard disk drive failure prediction models into RAID-6 systems.

The prediction of hard disk drive failures was also addressed by the work of I. Chaves *et al.* [CdPL+18]. However, the authors did not distinguish between good and failed hard disk drives, but predicted the remaining useful lifetime. Their approach also used S.M.A.R.T. features as health indicators of hard drive failures, but they used a Bayesian Network as failure prediction method. To select only meaningful features for the regression task, the authors applied recursive feature elimination with Random Forest as baseline prediction method. For each of the remaining features, the authors fitted a linear model to a fixed window of the feature time series and used the slope of the linear model as an additional trend feature. As a final step in feature preprocessing, they applied a binning technique to discretize the features into categories. The Bayesian Network structure contained nodes for each feature and a parent node representing the operating time of the hard disk drive, while the parameters of the Bayesian Network were estimated using Laplace Smoothing. To assess the prediction quality of their proposed model, the authors used a data set that encompassed 49,056 hard disk drives. I. Chaves *et al.* compared their Bayesian Network with a standard reliability method as well as with two approaches from other research teams, namely another Bayesian Network-based approach and a Recurrent Neural Network-based approach. The authors demonstrated that their Bayesian Network outperformed the standard reliability method and the other Bayesian Network with respect to the measures α-λ performance, prognostics horizon, and relative accuracy. As the Recurrent Neural Network-based approach did not provide predictions with uncertainty, the comparison

was made only with respect to relative accuracy, where the Recurrent Neural Network-based approach performed slightly better.

Y. Zhang *et al.* [ZXHP18] presented a Long Short-Term Memory-based approach for predicting the remaining useful lifetime of lithium-ion batteries. The authors pointed out that previous approaches neglected the use of long-term dependencies on capacity degradations, so they decided to use Long Short-Term Memory. Their proposed network architecture included resilient mean square backpropagation for network optimization, dropout to prevent the network from overfitting, and Monte Carlo simulation to model prediction uncertainties. Furthermore, the network consisted of two hidden Long Short-Term Memory layers with 50 and 100 neurons, respectively, and a Feed-Forward output layer with one neuron. Finally, the authors compared their Long Short-Term Memory with a Support Vector Machine, a Particle filtering algorithm, and a basic Recurrent Neural Network. The models were applied to six lithium-ion battery data sets measured under two different temperatures. The results suggested that the proposed Long Short-Term Memory approach outperformed the other models, although the confidence intervals resulting from the proposed Long Short-Term Memory were often comparatively large.

The use of simulation data for model training while applying the learned models to real-world data was analyzed by C. Sobie *et al.* [SFN18]. Their approach was specifically tailored to classify bearings with race faults. Using the simulated data, they extracted statistical features based on angle synchronous averaging and envelope functions. In addition, the authors also derived the energies of the first wavelet decompositions. Then, the features were normalized using Z-score normalization. As classification methods, Sobie *et al.* used standard machine learning methods, namely logistic regression, k-Nearest Neighbor, Random Forest, Support Vector Machine, and Multi-Layer Perceptron, a Convolutional Neural Network with one hidden layer, and first-Nearest Neighbor Dynamic Time Warping. The comparison of model training using simulation data with model training using real-world data indicated that the models trained on simulation data were superior to those trained on real-world data with respect to accuracy. In addition, the Convolutional Neural Network and first-Nearest Neighbor Dynamic Time Warping outperformed the classical machine learning methods based on feature extraction.

J. Shen *et al.* [SWLY18] proposed a part-voting Random Forest approach for hard disk drive failure prediction. To this end, the authors trained a Random Forest model to distinguish between normal hard disk drives and hard disk drives with impending failures. In addition, they used a sliding window on the predictions to correct misclassifications due to noise. More specifically,

they calculated the ratio of "failed" and "good" predictions and if it exceeded a certain threshold, the final prediction was set to "failed". The part-voting strategy was incorporated to select only a particular subset of decision trees from the Random Forest for individual instances based on their S.M.A.R.T. measurements. Here, the authors applied clustering to group similar time series and cluster-wise removed decision trees from the Random Forest based on their performance on the out-of-bag samples. For the evaluation of the part-voting Random Forest, J. Shen *et al.* used two data sets with a total of 64,193 hard disk drives. After manually selecting S.M.A.R.T. features, they applied their part-voting Random Forest model and compared it with Classification Tree, Recurrent Neural Network, and a standard Random Forest. The results proved that the part-voting Random Forest yielded Pareto optimal results with respect to failure detection rate and false alarm rate.

In contrast to J. Shen *et al.*, X. Sun *et al.* [SCH+19] used a temporal Convolutional Neural Network with modified loss function for hard disk drive failure prediction. For this purpose, the authors first normalized the input features based on the failure probability to allow the aggregation of monitoring data from different vendors and also to predict failures for every vendor. Then, they described their temporal Convolutional Neural Network, which consisted of two successive convolutional layers, each combined with a max pooling layer, and two fully-connected layers at the end. The input of the neural network were two-dimensional data, where the dimensions were represented by feature space and time. One-dimensional kernels were used to slide in the time dimension to extract meaningful information. The final output of the second fully-connected layer was the probability of failure. In addition, the authors proposed a novel loss function based on binary cross-entropy to handle imbalanced data sets. That is, the binary cross-entropy was multiplied by different weights depending on whether the sample was correctly or incorrectly classified. In order to evaluate their proposed model, the authors used a data set consisting of 71,839 hard disk drives. First, X. Sun *et al.* selected S.M.A.R.T. features based on change point detection and, then, compared the prediction quality of their model with Random Forest, Long Short-Term Memory, and two Convolutional Neural Networks with different loss functions. Their proposed temporal Convolutional Neural Network utilizing the novel loss function produced the highest values with respect to precision, recall, and F1-score.

D. Knittel *et al.* [KMN19] proposed an approach for the diagnosis of milling machines using feature extraction steps and the application of different machine learning methods. Their use case focused on composite sandwich structures with honeycomb cores. On the basis of raw machine data, the authors derived

41 features from both the time and frequency domains. In addition, D. Knittel *et al.* applied Principal Component Analysis to reduce the dimensionality of the feature space. Thereby, the authors used only the first five principal components. The labels were created using the flatness value. Here, the authors created two classes representing the good and bad machine health conditions. To learn the relationship between the first five principal components and the created machine health state classes, the authors tested k-Nearest Neighbor, Support Vector Machine, and Decision Tree. In their experimental results, the authors relied on accuracy and concluded that the Support Vector Machine provided the best machine health state detection.

A deep learning approach to classify different bearing fault types was presented by D.-T. Hoang and H.-J. Kang [HK19b]. Their approach converted one-dimensional vibration time series into gray-scale vibration images based on the amplitude of the vibration signal. These vibration images were then used as input to a Convolutional Neural Network., which consisted of two convolutional layers, each followed by a subsampling layer, and a fully-connected layer at the end. To evaluate their proposed model, the authors conducted a case study of bearing data in normal condition and three different types of faults. The authors compared their vibration image Convolutional Neural Network with a one-dimensional Convolutional Neural Network and a Stacked Autoencoder. Without the presence of noise, both Convolutional Neural Network-based approaches clearly outperformed the Stacked Autoencoder. Moreover, the proposed Convolutional Neural Network with vibration images proved to be the most robust in the presence of increased noise.

Convolutional Neural Networks have also been employed by T. Han *et al.* [HLYJ19] for the detection of faulty machine conditions. Therefore, the authors pre-trained their Convolutional Neural Network on a large data set and compared three transfer learning strategies for the adaptation to new, unseen data sets. The evaluation was performed using a gearbox fault data set, where accuracy was considered as the measure of quality. The comparison showed that the use of transfer-learning strategies significantly improved the predictive power compared with using the pre-trained model only as well as completely re-training the models. In addition, the results suggested that one transfer-learning strategy performed best, although the different strategies performed closely for some scenarios.

J. Zhang *et al.* [ZZH+20] introduced the application of transfer learning for predicting rare failures of hard disk drives and solid state drives. Here, transfer learning refers to transferring a model trained on a particular drive model to another drive model, of which there is only limited monitoring data. The

approach presented by J. Zhang *et al.* involves two main steps, namely the application of the transfer learning algorithm TrAdaBoost and an algorithm for mapping instances to drives. The first algorithm is an adaptation of the boosting method AdaBoost that allows transfer learning. Here, the authors investigated the use of four different weak learners, namely Support Vector Regression, Gradient Boosted Regression Tree, Regularized Greedy Forests, and Recurrent Neural Networks. The latter component of the proposed approach finally estimated the health state of the drive under consideration. J. Zhang *et al.* used two real-world data sets to compare their transfer learning model with standard machine learning models using the measures failure detection rate, false alarm rate, F1-score, and AUC-ROC value. The evaluation results exemplified the improvement using transfer learning compared to standard machine learning methods with respect to all four weak learners within TrAdaBoost.

Another approach for the detection and prediction of rare critical events was proposed by M. Dangut *et al.* [DSJ20]. The authors presented a two-stage approach by means of deep learning models focusing on aircraft engines. First, an Autoencoder was trained to detect critical event patterns in the monitoring data. To this end, the encoder component of the Autoencoder was trained using data from engines in a healthy state only. Then, critical events were detected based on the reconstruction error after decoding the encoded signal. The assumption was that a high reconstruction error implied a critical event because the encoder was trained only on data from healthy engines. Subsequently, the data was passed to the critical event prediction component only if a critical event was detected. For the prediction task, a bidirectional Gated Recurrent Unit was implemented. In order to evaluate the two-stage model, the authors used real-world data from aircraft engines and compared their approach with two types of Recurrent Neural Networks, namely Long Short-Term Memory and Gated Recurrent Unit. The experimental results revealed that the two-stage approach surpassed the individual Recurrent Neural Networks.

J. Campos and E. Costa [CC20] presented an approach for failure prediction of an operating system based on fault injection data. For this purpose, the authors emulated 16 different types of faults using the so-called SWIFI (Software Implemented Fault Injection) fault injector. The faults were injected in the form of bugs into the Linux kernel of a running operating system. Next, the authors grouped the 16 fault types into four failure classes. Using these classes as well as the normal class, the authors trained six different machine learning algorithms to predict whether a bug will occur in the near future and, if so, to which failure class the bug will belong to. For their experiments, the authors measured three distinct workloads, each with approximately 4500 runs. The

results showed that the best machine learning method for predicting failures in the operating system was XGBoost with an F2-score[1] of up to 98.5%.

S. Haidong *et al.* [HZJH20] proposed a fault diagnosis model for rotary machines based on Autoencoders and transfer learning. To achieve this, they developed a Stacked Autoencoder with scaled exponential linear unit, correntropy, and non-negative constraint to learn the dependencies between measurement data and rotary machine faults. A parameter transfer strategy was deployed to adapt this pre-trained model to new application scenarios. In addition, Particle Swam Optimization was used for automated parameter adjustment. The authors applied their model to data sets of bearings and gears, comparing the prediction quality of their model with different transfer learning Autoencoders and Deep Belief Networks with respect to accuracy. The results showed that Autoencoders yielded the best results, with the proposed model being superior.

A comparison of deep learning (i.e., Feed-Forward and Recurrent Neural Networks) and statistical (i.e., Projection to Latent Structure) approaches for IoT-enabled manufacturing was conducted by D. Shah *et al.* [SWH20]. Additionally, they evaluated the impact of different feature engineering efforts. For their comparison, the authors acquired monitoring data from a pipe flow testbed. Their results demonstrated the necessity of appropriate feature extraction and feature selection. More specifically, merely applying deep learning models to the raw monitoring data yielded only very poor prediction quality. Nonetheless, D. Shah *et al.* concluded that statistical methods combined with feature engineering provided high and robust accuracy.

A digital twin-based approach toward predictive maintenance of Computerized Numerical Control (CNC) machine tools was described by W. Luo *et al.* [LHY+20]. Their proposed framework combined data-driven remaining useful lifetime estimation, physical model-based degradation model, and internal state simulation. The authors evaluated their hybrid framework using a case study focusing on cutting tool failures. Therefore, W. Luo *et al.* compared their hybrid model with stand-alone model-based and stand-alone data-driven approaches, demonstrating that their hybrid framework significantly improved the prediction of remaining useful lifetime with respect to percentage error.

D. She and M. Jia [SJ21] also introduced a remaining useful lifetime prediction approach tailored to bearings. However, the authors not only predicted point estimates, but also provided confidence intervals describing the uncertainty in the predictions. In a first step, their approach computed a health index based on measurement data. Next, the authors trained a bidirectional Gated Recurrent Unit to predict the remaining useful lifetime based on the time series

[1] F2-score is a variation of F1-score that assigns a higher weight to recall.

of health indices. More specifically, their network architecture consisted of five bidirectional GRU layers and fully-connected regression layers. To derive confidence intervals, the authors used bootstrapping. Accordingly, they randomly sampled k times from the original training data to learn k prediction models. These models were merged to obtain a mean and variation of the remaining useful lifetime predictions. To evaluate their proposed model, D. She and M. Jia used a bearing data set on which they compared the predictions of their model with Particle filter model, Gated Recurrent Unit, Long Short-Term Memory, and Fully-Connected Neural Network with respect to root mean square error and mean absolute percentage error. For this experiment, the bidirectional Gated Recurrent Unit proposed by D. She and M. Jia provided the most accurate remaining useful lifetime prediction with respect to both error measures. In addition, it also provided comparatively narrow confidence intervals.

X. Bampoula *et al.* [BSNA21] proposed a critical event prediction method for rolling mill machines in the steel production industry. For this purpose, they used two temperature and two hydraulic force sensors to predict the remaining useful lifetime of rolling mill machines to reduce maintenance actions and, consequently, costs. Their approach employed two stages. First, a Long Short-Term Memory Autoencoder was learned to classify the current health state of the machine. To this end, three health states were modeled, namely good, mediocre, and bad. Next, an individual Long Short-Term Memory Autoencoder was trained for each of the three classes of machine health states. These Long Short-Term Memory Autoencoders were used to regress the remaining useful lifetime with respect to the identified machine state. To evaluate their approach, the authors used real-world data from a rolling mill machine. The authors showed that their approach achieved an F1-score of about 80% and can predict critical events with a lead time of approximately one day.

3.2.3 Update Strategies for Critical Event Prediction Models

One of the first incremental learning algorithms for machine learning models was proposed by J. Schlimmer and R. Granger [SG86]. In this work, the authors first described three evaluation criteria for incremental learning, namely amount of training data, costs of updating, and quality of the learned abstraction. For their proposed model, they used a binary classification model, where each feature was assigned a weight as a base-level predictor. The classification was then based on Bayesian formulas. J. Schlimmer and R. Granger incorporated an updating scheme that adjusted feature weights based on the error type (i.e., false positive or false negative). Finally, they demonstrated the effec-

tiveness of their model using two case studies with different noise levels. As a measure of evaluation, the authors applied accuracy.

Another early approach was described by R. Solomonoff [Sol89]. His incremental learning approach used algorithmic probability theory to emulate the human learning process. Therefore, the approach began with a small set of rules, also referred to as concepts, that represented initial knowledge. During application, the system increased its knowledge by adding new rules and adjusting the probabilities for each rule. Thus, this approach is quite related to the (semi-)autonomic construction of an expert system. However, the author unfortunately did not provide an evaluation of the proposed model.

N. Syed *et al.* [SLS99] introduced an incremental learning approach for Support Vector Machines to deal with concept drifts. As quality criteria, they also referred to those introduced by J. Schlimmer and R. Granger [SG86], but added three more criteria for evaluating the robustness and reliability of the incremental learning model, namely stability, improvement, and recoverability. In terms of stability, the accuracy achieved on the test sets should not differ significantly between incremental learning steps. Moreover, the accuracy of the incremental learning model should improve across the incremental learning steps, as a larger amount of training data should improve the model quality. The recoverability criterion addressed the ability to recover from potential quality degradation via subsequent incremental learning steps. To incrementally learn the Support Vector Machine, the authors stored only the support vectors in the memory so that the model could be easily and efficiently updated incrementally. Finally, the authors compared their incrementally learned Support Vector Machine with a Support Vector Machine that was completely re-trained for each evaluation batch. Here, N. Syed *et al.* were able to show that the incremental Support Vector Machine occasionally outperformed the Support Vector Machine with all data, while it mostly achieved a slightly worse accuracy.

Incremental learning of Support Vector Machines has also been addressed by C. Diehl and G. Cauwenberghs [DC03]. Their approach was based on the Karush-Kuhn-Tucker condition, which divides all training instances into three different sets, namely margin support vectors, error support vectors, and reserve vectors. As updating process, the authors employed adiabatic incremental learning, where for a perturbation of the new vector coefficients, the goal was to identify the required changes in the margin vector coefficients and the bias that maintain the Karush-Kuhn-Tucker conditions. The perturbation parameter ranged from 0 to 1 and was incremented at each perturbation. When the perturbation parameter reached one, all new vectors were assigned to one of the three sets and all vectors satisfied the Karush-Kuhn-Tucker conditions. For

the evaluation, the authors compared their incremental Support Vector Machine with a complete re-training of a Support Vector Machine for a data set with varying hyperparameter settings of kernel width and regularization parameter. However, the predictive power of the models was not assessed, but only their computational cost in terms of the number of floating point operations, the number of kernel evaluations, and the number of perturbations. The results showed that the smaller the kernel width, the less computational time the incremental learning approach saved. However, the higher the regularization parameter was, the higher the computation time savings were.

An adaptation of Extremely Random Forest for incremental on-line learning was proposed by A. Wang *et al.* [WWCL09]. The approach used streaming data to learn a model for classification tasks. To this end, lists of matching instances and their respective classes were stored in each leaf node of the decision trees. When new instances arrived, they were passed through the decision trees and also stored in the leaf nodes. Then, the Gini index was calculated as a measure of the impurity of leaf nodes to decide when to construct a new tree for the instances stored in that leaf node. The authors compared this incremental Extremely Random Forest to a standard Extremely Random Forest with respect to accuracy on multiple typical machine learning classification tasks, such as spam classification. However, the standard Extremely Random Forest outperformed the incremental Extremely Random Forest on most tasks. The authors also applied their model to a video object tracking task and compared the performance achieved with an approach used by another research team. Here, however, the proposed incremental Extreme Random Forest model substantially outperformed the other approach.

J. Guajardo *et al.* [GWM10] investigated the updating of machine learning models for forecasting tasks instead of classification tasks. The general idea of the authors was to use the sequentially arriving ground truth data of the monitored systems as an extension of the training data to update the forecasting model. More specifically, the authors focused on Support Vector Regression as a forecasting method and considered only seasonal time series for their approach. First, J. Guajardo *et al.* applied a wrapper method to select only relevant features and tuned the hyperparameters using a grid-search on the validation data set. To emulate the on-line forecasting procedure, the test data set was divided into several consecutive sequences, each containing a single period of the seasonal pattern. After predicting a period, the actual monitoring values were added to the training data set and used to train a new Support Vector Regression model. This process was repeated until no more test data were available. The authors used four time series from the M1 competition

and five sales time series to evaluate the forecast quality with respect to mean absolute percentage error, mean square error, and mean absolute error. They compared the accuracy of their model with the achieved forecast accuracy of the static Support Vector Regression model trained on the initial training data set only. The reported results demonstrated the improvement of forecast accuracy for almost all evaluation scenarios.

The framework *ADAIN* for incremental model learning from stream data was introduced by H. He *et al.* [HCLX11]. Their framework first learned an initial model using a mapping function. Thereby, the authors recommended non-linear regression models and selected Multi-Layer Perceptron for their later procedure. As new stream data became available, the model was applied to that data to obtain an estimate of the model error for the new data distribution. For the update task, higher weight was assigned to the misclassified instances to focus on those instances where the current learning method encountered difficulties. In this way, the model was refined and the whole process was repeated when new batches of data arrived. The authors analyzed their proposed approach both analytically and experimentally. The experiments demonstrated the effectiveness of their model, while the authors also showed that using Support Vector Regression instead of Multi-Layer Perceptron as mapping function yielded similar results.

M. Pratama *et al.* [PAAL13] proposed a parsimonious network based on a fuzzy inference system called *PANFIS*. Here, PANFIS started with an empty rule set and iteratively updated the set of fuzzy rules in five steps. First, the rules were updated based on extended Self-Organizing Map theory. Next, new fuzzy rules were identified using the so-called datum significance concept. Using the ϵ-completeness criterion, the antecedent fuzzy rules were selected. Then, similar rule sets were merged and inconsistent rules were pruned. Finally, the parameters were adjusted to the new conditions. To evaluate this update procedure, the authors used both synthetic as well as real-world data sets. The authors compared their model with state-of-the-art evolving neuro-fuzzy methods with respect to root mean square error and non-dimensional error index, showing that PANFIS was able to keep up with state-of-the-art approaches. For some scenarios, PANFIS even surpassed these approaches.

A neural network-based ensemble model with model updating based on concept drift detection was introduced by S. Xu and J. Wang [XW16]. To classify data streams, they employed Extreme Learning Machines, i.e., a Feed-Forward Neural Networks with one hidden layer. As the main parameters of such an Extreme Learning Machine are the size of the hidden layer and the selected activation function, the authors automatically tested different

numbers of neurons on the hidden layer and selected the setting that resulted in the highest validation accuracy. In addition, they learned an ensemble of Extreme Learning Machines to improve the robustness of the predictions by randomly selecting the activation function used for each of the Extreme Learning Machines. The final prediction was computed as a weighted majority vote, where the weight of each Extreme Learning Machine was derived from the accuracy achieved on the validation set. To detect concept drifts, S. Xu and J. Wang used the Hoeffding bound. If no concept drift was detected, the classifiers were incrementally updated by adjusting the number of neurons in the hidden layer. However, if concept drift was detected, the classifiers with the smallest weights and, thus, the worst accuracy on the validation set were discarded. Finally, the proposed incremental Extreme Learning Machine approach was compared to a standard Extreme Learning Machine, a Backpropagation Neural Network, and other related approaches from other research teams, using both artificial and real-world data sets. The authors showed that their approach achieved the highest accuracy and the shortest runtime in most cases.

F. Castro *et al.* [CMJG+18], by contrast, focused on incrementally adding new classes during runtime for image classification tasks. Their approach was based on deep neural network architectures. In this approach, each classification layer was responsible for a particular feature in the image and output logits that were then used to compute a softmax function for the final classification. The novel contribution of the authors was the introduction of a cross-distilled loss function that preserved the already learned knowledge about old classes, while cross-entropy loss was utilized to dynamically learn relationships about the newly added classes. To retain knowledge about all classes, only representative examples per class were stored. These representative examples were identified via herding selection, which considers the deviation of examples from the average over all examples of the same class. The authors evaluated their proposed model using two real-world image classification data sets available online. Thereby, the authors applied a Convolutional Neural Network as classification model. The results found that the proposed model outperformed the current state-of-the-art approaches for image classification.

M. Islam *et al.* [IHL18] also dealt with image processing, but considered updating prediction error parameters of object tracking models. For this purpose, the authors used a kernelized correlation filter tracker as the base-level method. However, instead of the commonly used fixed learning rates, they employed dynamic learning rates for prediction error updating. For dynamically adjusting the learning rate, M. Islam *et al.* computed the peak-to-sidelobe ratio to handle sudden illumination variations, deformations, abrupt motion changes,

and occlusions. To evaluate the proposed update procedure, the authors used three object tracking data sets and compared the distance precision score, overlap success score, and average running speed of their model with nine other approaches. By doing so, the authors showed that their model achieved an effective trade-off between object tracking quality and speed.

The binary prediction of impending hard disk drive failures was analyzed by J. Xiao *et al.* [XXW+18]. For this purpose, they applied an Online Random Forest, where the quality of splits was assessed using Gini impurity. The individual decision trees in the Online Random Forest were updated based on two Poisson distributions, discarding outdated trees to unlearn old and, thus, potentially no longer useful information. Such outdated trees were determined based on their age as well as their out-of-bag error. For the evaluation of the described Online Random Forest, the authors used two data sets with different hard disk drive models. With respect to the failure detection rate and the false alarm rate, the authors first showed the superiority of Random Forest over a simple Decision Tree as well as Support Vector Machine. Next, they compared Online Random Forest with a static Random Forest model, a monthly re-trained Random Forest model using all data, and a monthly re-trained Random Forest model using a temporal data replacement strategy. The results illustrated the importance of incremental learning, with the described Online Random Forest and the monthly re-training using all available data performing the best.

C. Constantinides *et al.* [CSGK19] described an incremental learning approach for intrusion detection systems. To this end, they built a model using multiple submodels. First, two Self-Organizing Incremental Neural Networks were trained for each type of intrusion attack. The outputs of each pair of two neural networks were then passed to a binary Support Vector Machine. Subsequently, the predictions of the numerous Support Vector Machines were used as input to a multi-class Support Vector Machine for the final prediction. This model was initially learned with only a small amount of data and then incrementally updated. For this purpose, a validation module was integrated to provide feedback to the learning component. At fixed cycles, the models were updated based on new, misclassified instances. Using a typical intrusion detection benchmark data set, the authors demonstrated that the detection accuracy increased over time, thus with increasing training data. Moreover, the incremental approach outperformed a static off-line model.

An extensive comparison between re-training and incremental learning of machine learning models and neural networks in the domain of performance prediction of adaptable software was conducted by T. Chen [Che19]. He used eight machine learning methods and a Multi-Layer Perceptron. During re-

training, all available data were used to learn a new model of the same prediction method, while incremental learning updated the existing models based on newly arrived data. Three data sets were used in the study, each with different characteristics. After analyzing the results, T. Chen concluded that neither strategy performed best with respect to the mean absolute error for all scenarios. Instead, incremental learning yielded lower mean absolute errors for the Multi-Layer Perceptron, while re-training was superior for the Support Vector Machine, for example. Nevertheless, the results suggest that re-training provides more robust predictions than incremental learning. In contrast, the training time for incremental updates was significantly shorter and less varying.

Similar to F. Castro *et al.* [CMJG+18], J. He *et al.* [HMSZ20] also addressed the challenge of new classes at runtime for image classification. However, J. He *et al.* not only focused on integrating new classes, but also added new instances of the existing classes into the update loops to counter concept drift. As classification method, the authors employed the Nearest Class Mean classifier. To integrate new classes into an existing model, J. He *et al.* developed a novel loss function based on cross-distillation with accommodation ratio. For the old classes, only representative sets were stored instead of all instances. New instances of existing classes were used to update the respective class representative set. Apart from incremental learning, periodic off-line re-training was also carried out to handle catastrophic forgetting and concept drift even better. For the evaluation of their proposed model, the authors used three image classification data sets. They compared the accuracy achieved by their model with state-of-the-art approaches and showed that their model performed superior.

W. Rang *et al.* [RYC+20] introduced an update strategy that controls the selection of samples in the training set. Thus, they described a middle layer that processes incoming and already available data. For this purpose, a data life index was calculated by assessing the impact of each sample during the previous update cycle. Based on this data life index, samples were grouped into four categories, with the last category discarding data. The authors demonstrated the effectiveness of their approach using pattern recognition, classification, and recommendation tasks, where it resulted in lower costs and less training time compared to standard periodic updates.

In their framework, Y. Xiang *et al.* [XPL21] combined data-driven and physics-driven prediction models. The physics-based model was constructed using simulation data, while simulation data were used in combination with experimental data to learn the data-driven model. For the data-driven model, Gaussian process regression was used. Both models were then combined by assigning a weight to each of the models. Y. Xiang *et al.* computed these

weights based on the minimum deviation of the probability distributions of the posterior error and another data-driven model. By means of a validation step, the framework identified whether an update was required. In the case of a sufficiently large divergence, updates were performed based on maximum likelihood estimation. Using a case study of temperature data, the authors showcased the effectiveness of their proposed hybrid model.

Part II

Improving Time Series Forecasting

Chapter 4

Telescope: Remainder Learning for Component-based Time Series Forecasting

Beyond its application in the mere estimation of future values, time series forecasting is also an important component of many critical event prediction systems, where forecasts are combined with thresholds to predict impending threshold violations. As a result, time series forecasting has become an important element of decision-making processes, used in many fields including business, economics, finance, science, and engineering [MSR16]. For this reason, time series forecasting represents a highly active research field in the last decades. Many different forecasting methods have been developed during this time, each with its own advantages for specific time series domains. However, there is no single forecasting method that performs best for all time series, as stated by the "No Free Lunch Theorem" [WM97]. Instead, the forecast quality and runtime of each forecasting method benefit from certain time series characteristics, while other characteristics degrade these for that particular forecasting method. For this reason, expert knowledge is required to decide which forecasting method to choose for each time series. Although expert knowledge is useful, this is a time-consuming task that cannot be fully automated. Another approach is trial and error, which is hardly applicable in practice due to its inefficiency. Therefore, hybrid forecasting (also known as mixed or combined forecasting) was introduced to overcome this problem of individual approaches by combining the advantages of at least two individual forecasting methods. However, a serious shortcoming in the literature is that most contributions to hybrid forecasting approaches do not provide runtime evaluations. In general, ensemble forecasting approaches, such as those presented in Section 3.1.1, have a very high runtime due to the fact that multiple forecasting methods must be executed on the entire time series, weights need to be estimated, and the results have to be combined. In contrast, forecasting method recommendation provides a short on-line runtime since only one fore-

casting method is applied, but it requires a large database of time series from the same domain and also a time-consuming training step for rule generation. However, the runtime of component-based forecasting strongly depends on the adopted decomposition method. In addition, these approaches also exhibit other shortcomings. Most hybrid forecasting approaches in the literature based on time series decomposition follow two different approaches. One group relies on decomposition methods and only applies one forecasting method for each time series component. The other group utilizes two forecasting methods, where the second forecasting method is subsequently applied to the residuals of the first forecasting method.

In order to improve forecasting performance not only on average, but also in terms of reliability and runtime, we present a novel hybrid, multi-step-ahead forecasting approach for seasonal, univariate time series based on explicit time series decomposition and three different forecasting methods. We call the proposed hybrid forecasting approach *Telescope* in reference to the analogy with the vision of far-distant objects. However, this approach is also subject to the "No Free Lunch Theorem" in that the following assumptions must be made:

1. The time series to be forecast is univariate.

2. Only seasonal time series (cf. cyclical time series, see Section 2.2.2) with a length of more than two full periods are passed to Telescope for forecasting future values.

3. As only seasonal time series are allowed, the lengths of the periods must be constant. Yet, several overlapping seasonal patterns are permitted.

In addition, several hundreds of historical observations are required to obtain superior results. Although Telescope can also be applied to shorter time series, existing individual forecasting methods can already handle short forecast horizons very well, too. To emphasize the strength of Telescope, longer time series (at least 500 to 1000 observations) and, thus, more values in the forecast horizons should be targeted.

This chapter describes Telescope in detail, with the remainder organized as follows: The overall design of Telescope is described in Section 4.1, followed by in-depth introductions to the different phases of Telescope in Sections 4.2-4.5. Finally, the chapter is summarized and the respective research questions are answered in Section 4.6.

The content of this chapter is based on the master's thesis of M. Züfle [Züf17] and a related publication at the International Work-Conference on Time Series Analysis (ITISE) [ZBH+17]. Extending the approach presented in this thesis,

we further refined particular components of Telescope, resulting in publications at the 11th ACM/SPEC International Conference on Performance Engineering (ICPE) [BZG+20], the 36th IEEE International Conference on Data Engineering (ICDE) [BZH+20a], and the Proceedings of IEEE [BZH+20b]. However, these modifications of Telescope were mainly driven by A. Bauer and are, therefore, part of his PhD thesis [Bau20]. Finally, we made the most recent version of Telescope available on our GitHub page[1] as a fully automated end-to-end forecasting tool that requires only the historical time series observations and the forecasting horizon as input and provides a forecast of the specified length.

4.1 Overall Design of Telescope

The approach of Telescope, a novel hybrid, multi-step forecasting approach for univariate, seasonal time series, is based on time series decomposition, clustering techniques, and three individual forecasting methods, namely ARIMA, NNetAR, and XGBoost. A diagram of the simplified forecasting procedure is shown in Figure 4.1. However, this is only a brief summary of the Telescope workflow to demonstrate the fundamental steps employed to derive forecasts from historical observations.

Telescope's general workflow starts with a preprocessing step. After the raw values are passed to Telescope, the algorithm estimates the frequency of the seasonal time series, i.e., the length of a period, by applying periodograms to the raw time series. Additional information on the mathematical foundations of periodograms are provided in Section 2.2.3. Then, Telescope applies an anomaly detection and removal method based on the generalized extreme studentized deviate test. In the final step of the preprocessing phase, the resulting time series is analyzed to determine its composition type. In the case of a multiplicative time series, a transformation is required and Telescope applies the logarithm to the time series so that the resulting time series is of additive composition type. Subsequently, Telescope performs two independent phases, namely the creation of the categorical information and the decomposition of the time series.

As can be seen on the left side of Figure 4.1, categorical information are generated using clustering. For this purpose, the time series is split into individual periods with respect to the frequency estimated in the preprocessing phase. For each of these periods, statistical characteristics are computed and used to perform k-Means clustering. A brief summary of the k-Means clustering algorithm can be found in Section 2.4.1. Once k-Means has determined the

[1] Telescope repository: `https://github.com/DescartesResearch/telescope`

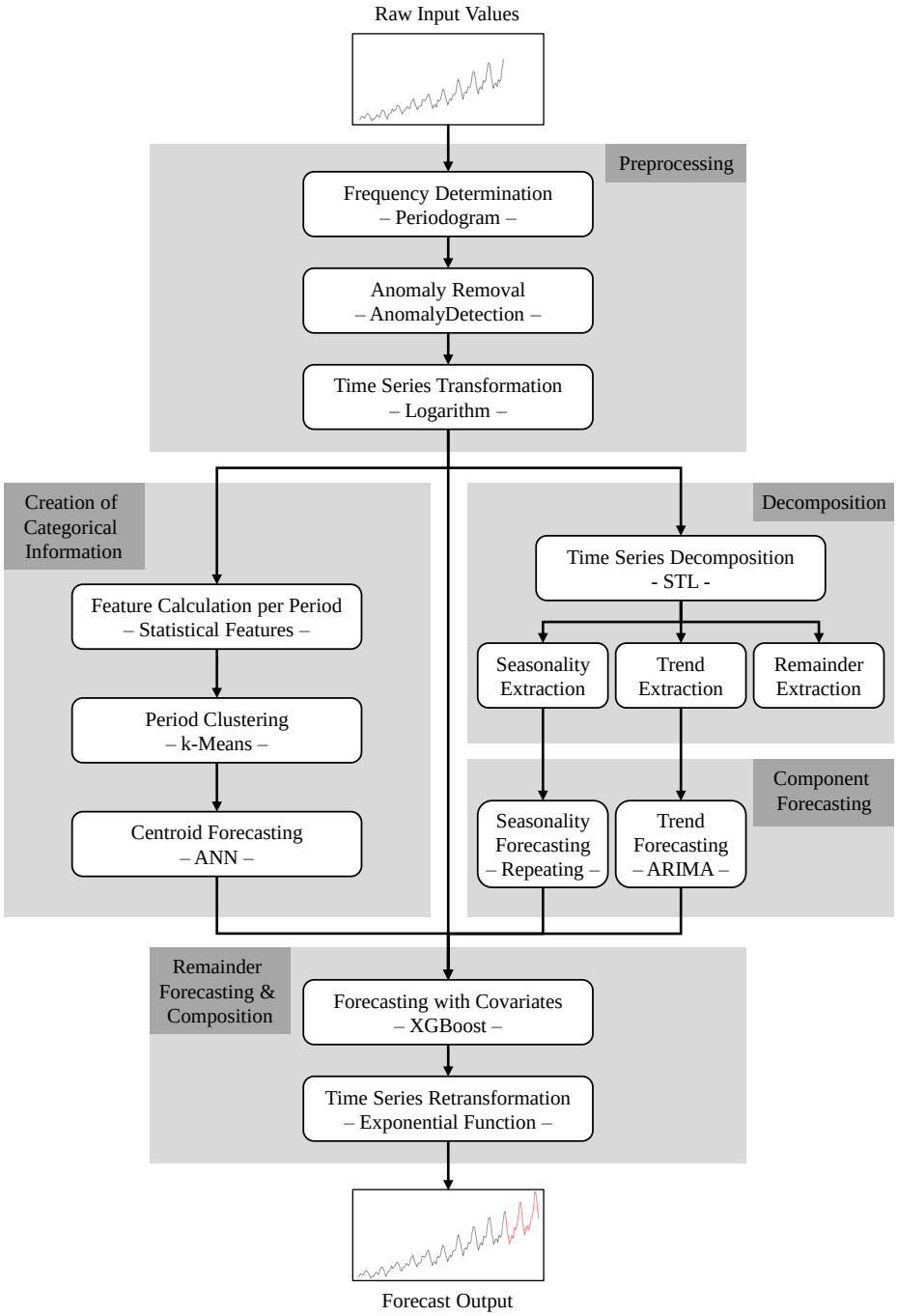


Figure 4.1: A simplified illustration of the Telescope approach.

cluster label of each period, these are forecast by the application of an artificial neural network (ANN), or more precisely by NNetAR (cf. Section 2.3.6).

The other phase (right side of Figure 4.1) decomposes the time series according to the structural time series model. Specifically, STL is used here for time series decomposition. STL splits a time series into three components, namely seasonal, trend, and remainder. Each of these components can be considered as a new time series, with the sum of all components yielding the original time series. By definition, the seasonal component is a repeating pattern and, thus, this pattern is continued for the entire forecasting horizon. In contrast, ARIMA is applied to forecast the trend. Finally, the remainder component contains the part of the original time series that cannot be explained by seasonality or trend. Thus, the remainder component is the statistical part that may be caused by noise or other disturbing factors. Therefore, the remainder is hard to forecast using individual forecasting methods and, hence, is correlated with a high error rate. For this reason, Telescope does not forecast the remainder directly, but learns it indirectly in the final phase.

In the last phase, XGBoost is applied to regress the values of the time series by adding several covariates to the original data. For this purpose, the created categorical information and the time series component forecasts are passed to XGBoost as covariates, while the original values of the time series are the target of the regression. Finally, in case the time series was transformed in the preprocessing phase, the corresponding re-transformation, i.e., the exponential function, is applied and Telescope returns the forecast values of the horizon (displayed in red in Figure 4.1).

In addition to Figure 4.1, Telescope is presented as a more detailed pseudocode in Algorithm 1. Here, the general workflow is the same as shown in Figure 4.1, but Algorithm 1 provides additional details that would exceed the scope of Figure 4.1. For instance, Telescope performs more time series investigations in the preprocessing phase (cf. Lines 1-13). Apart from frequency estimation, removal of anomalies, composition type identification, and determination of the minimum value to transform the scale of the original data into positive values, another test is performed that checks whether the time series exhibits a significant trend (cf. Line 5). Testing whether the time series has a significant trend is an important task of Telescope, since the accuracy of the XGBoost forecasts depends strongly on the trend behavior of the time series. To this end, STL is applied to the original time series to decompose the time series. The trend component is then forecast and its range is compared to the range of the entire time series to determine if the trend represents a large portion of the range of the time series. In particular, the choice of boosting method of

Algorithm 1 Telescope workflow

Input: time series to be forecast x, length of the forecast h
Output: forecast values fc_xgb

determine time series frequency and remove anomalies
f = determineFrequency(x)
$\widetilde{x}$ = removeAnomalies(x, f)
tests for significant trend, multiplicative decomposition, and necessity of shifting
has_trend = testTrend($\widetilde{x}$, f, h)
log_test = testMultiplicativeDecomposition($\widetilde{x}$, f, ACF = TRUE)
min_value = min($\widetilde{x}$)
if (log_test) **then**
  if ($min\_value \leq 0$) **then**
    $\widetilde{x}$ = $\widetilde{x}$ + abs(min_value) + 1
  end if
  $\widetilde{x}$ = log($\widetilde{x}$)
end if
time series decomposition, season, and trend forecasting
$decomp$ = stl(ts($\widetilde{x}$, f), s.window = "periodic", t.window = length($\widetilde{x}$)/2)
fc_season = elongate($decomp$["season"], h)
$model$ = fittingModels($decomp$["trend"])
fc_trend = forecastTrend($\widetilde{x}$, f, $model$, h)
creation of categorical information
$clusters$ = clusterPeriods($\widetilde{x}$, f)
$fc\_clusters$ = forecastClusters($clusters$, f, $\widetilde{x}$, $repeats$, h)
data preprocessing and execution of XGBoost
if (log_test) **then**
  cov = cbind($decomp$["season"], $clusters$)
  $label$ = $\widetilde{x}$ − $decomp$["trend"]
  fc_cov = cbind(fc_season, $fc\_clusters$)
else
  cov = cbind($decomp$["trend"], $decomp$["season"], $clusters$)
  $label$ = $\widetilde{x}$
  fc_cov = cbind(fc_trend, fc_season, $fc\_clusters$)
end if
xgb = doXGB.train($label$, cov, getBooster(fc_trend, $decomp$["trend"], $\widetilde{x}$))
fc_xgb = predict(xgb, fc_cov)
re-transform the scale of the data and return the forecast
if (log_test) **then**
  fc_xgb = exp(fc_xgb) × exp(fc_trend)
  if ($min\_value \leq 0$) **then**
    fc_xgb = fc_xgb − abs(min_value) − 1
  end if
end if
return fc_xgb

XGBoost depends on the result of this test. As a final check of the preprocessing phase, the minimum value of the history is determined. If the decomposition is of the multiplicative type and in addition the minimum value is less than or equal to zero, the minimum value plus one is added to all observation values in the history before the application of the logarithm. Afterwards, these logarithmized values are used for the further workflow of Telescope.

Regarding time series decomposition, Algorithm 1 displays two input parameters, namely *s.window* and *t.window*. The *s.window* is set to periodic for seasonal extraction and the *t.window* is the span of the loess window used for trend extraction. For this reason, *t.window* should be a large value. Here, Telescope sets *t.window* to half the length of the history to prevent seasonal patterns from appearing in the trend component.

Although ARIMA is a state-of-the-art individual time series forecasting method, it cannot handle exponential trend patterns. Thus, the nature of the trend must be determined in advance. For this purpose, STL decomposition is applied to the original and logarithmized data respectively. Subsequently, linear models are fitted and, in the case of the logarithmized data, the exponential function is applied to the linear model to re-transform the scale. Once the models are fitted, the root mean square error is used to identify the model with less residuals. The trend type is determined according to the superior model.

XGBoost not only indirectly forecasts the remainder component, but also combines the forecasts of the individual time series components. However, since XGBoost offers numerous hyperparameters, there is no globally best parametrization of XGBoost for all time series. Again, several tests must be performed to adjust the most important parameters. First, Telescope checks whether the type of decomposition is multiplicative or additive. For time series with multiplicative decomposition, only the cluster labels and logarithmized seasonality are passed as covariates, with the de-trended, logarithmized time series as the regression target. After the forecast is performed, the exponential function is applied to the XGBoost forecasting result and the trend forecast, respectively. Finally, these results are multiplied to derive the final forecasting result. In contrast, if the time series exhibits additive decomposition, XGBoost is applied to the time series values as the regression target and the cluster labels, seasonality, and trend as covariates. A second test deals with the choice of boosting method, where two boosting methods are considered. The first boosting method is for time series with a strong trend pattern, while the second boosting method is for time series with only little or no trend. In a final step, Telescope checks whether the time series has been logarithmized and if the minimum value of the original time series is less than or equal to zero, as in

this case, the time series has been shifted to a positive scale. Consequently, the value added to the time series to make each observation value positive must be subtracted again after forecasting.

4.2 Time Series Preprocessing

The time series preprocessing phase of Telescope consists of three main steps, which are described in more detail in this section. Section 4.2.1 presents the frequency determination algorithm, followed by the detection and removal of anomalies in Section 4.2.2. Finally, the trend tests are explained in Section 4.2.3.

4.2.1 Frequency Determination

The frequency determination is the first preprocessing step as depicted in the topmost gray box *Preprocessing* in Figure 4.1 and follows a two-stage approach. First, a periodogram is employed to derive the most dominant frequency and next, a test is performed to determine whether this frequency is a typical time series frequency. This second step is crucial because periodograms often provide only rough estimates that vary around the actual frequency.

The procedure for the first stage, namely the estimation of the time series frequencies by means of periodograms, is given in Algorithm 2. For this purpose, the time series observations and a counter indicating the iteration number are required as input features. The time series observations must be passed to the algorithm, since the periodogram is computed on them (cf. Lines 3 and 6). However, since the frequency estimation algorithm often has to be applied multiple times because the found frequencies do not match any of the typical time series frequencies in the second stage, the iteration number is used to indicate whether a new periodogram has to be computed or not. As the periodogram is a deterministic computation, the execution of the periodogram on the same data with the same input parameters will always yield the same result. Therefore, the periodogram is calculated only in the first two iterations and stored in the workspace so that it can be re-used in later iterations (cf. Lines 4 and 7). If no common frequency is found in the first iteration, a span is added in the second iteration to smooth the time series (cf. Line 6). In addition, as the iteration counter increases, the algorithm determines and checks the next dominant frequency. To estimate the dominant frequency, the frequency with the highest spectrum is chosen. Generally speaking, the i-th most dominant frequency is the frequency with the i-th highest spectrum. Nevertheless, finding the highest spectrum does not necessarily indicate that the time series is seasonal and

Algorithm 2 Frequency estimation using periodograms

Input: observation values x, iteration number $numIters$
Output: i-th likeliest frequency f

```
1:  ### only determine periodogram for the first try
2:  if (numIters == 1) then
3:      pgram = spec.pgram(x)
4:      writeToWorkspace(pgram)
5:  else if (numIters == 2) then
6:      pgram = spec.pgram(x, spans = 5)
7:      writeToWorkspace(pgram)
8:  end if
9:  ### determine highest spectrum and corresponding frequency
10: max_spectrum = getIHighestSpectrum(pgram, numIters)
11: best_freq = getFrequencyForSpectrum(max_spectrum)
12: ### determine spectrum for the periodogram of diffs with lag best_freq
13: diff_values = diff(x, best_freq)
14: diff_pgram = spec.pgram(diff_values)
15: diff_max_spectrum = getIHighestSpectrum(diff_pgram, numIters)
16: ### accept frequency if the spectrum of the periodogram of diffs is considerably smaller
17: if (diff_max_spectrum < tolerance_level × max_spectrum) then
18:     f = best_freq
19: else
20:     f = −1
21: end if
22: return f
```

that the spectrum found provides the frequency of the time series. In case the spectrum is only slightly higher than for several other frequencies, the time series is likely to be random. To determine if the frequency found is actually a likely frequency of the time series, a second periodogram is computed. For this purpose, the difference between each value in the original time series and its precursor with a distance equal to the estimated frequency is computed (cf. Line 13). After this differentiation, the seasonal pattern with this particular frequency should no longer be present in the time series. Thus, the highest spectrum is determined for the second periodogram as well to compare the highest spectrum of the original periodogram and the one after differentiation.

If the highest spectrum of the second periodogram is smaller than a tolerance value multiplied by the highest spectrum of the first periodogram (cf. Line 17), the particular frequency is returned. Otherwise, the found frequency is discarded and -1 is returned (cf. Line 20). The tolerance level can be adjusted, however, it is fixed to 0.5 in this thesis.

Algorithm 3 shows the procedure of the overall frequency determination. In contrast to the frequency estimation algorithm, the entire frequency determination requires only the observed values of the time series. Also, only this method is called from the Telescope workflow, since it includes the frequency estimation in Line 12. First, a list of all common frequencies must be loaded (cf. Line 2). Afterwards, the frequency estimation presented in Algorithm 2 is applied until either a common frequency is found (cf. Line 15) or the number of maximum iterations is exceeded (cf. Line 10). To check whether the frequency

Algorithm 3 Frequency determination

Input: observation values x
Output: most likely frequency f

```
1:  ### get a list of all common frequencies
2:  common_fs = getCommonFrequencies()
3:  f = -1
4:  numIters = 1
5:  maxIters = 10
6:  ### only leave the loop if the maximum number of iterations is reached or a frequency
    estimated by the periodogram is accepted
7:  while (f == -1) do
8:      if (numIters > maxIters) then
9:          f = 1
10:         break
11:     end if
12:     estimated_f = estimateFrequencyPeriodogram(x, numIters)
13:     ### test whether there is a common frequency close to the estimated frequency
14:     if (nearby(estimated_f, common_fs)) then
15:         f = getClosestFreq(estimated_f, common_fs)
16:     end if
17:     numIters = numIters + 1
18: end while
19: return f
```

estimated by Algorithm 2 implies a common frequency, it is compared with the list of common frequencies. If the frequency matches one of the common frequencies (cf. Line 14), the common frequency matching with least difference is returned as the frequency of the time series (cf. Line 15). Otherwise, the entire frequency estimation and verification loop is repeated up to nine more times, where the i-th most dominant frequency is considered in the i-th iteration. Finally, if no frequency passes the verification tests after 10 iterations, the frequency is set to 1, indicating that the time series is not seasonal. In this case, Telescope terminates as its assumption of a seasonal time series is not satisfied and, consequently, an alternative forecasting method must be chosen.

4.2.2 Anomaly Detection and Removal

Given that anomalies can severely affect the quality of time series decomposition methods and, consequently, the accuracy of forecasts, anomaly detection and removal is a critical task for decomposition-based time series forecasting. Therefore, the detection and removal of anomalies is the second preprocessing step as depicted in the topmost gray box *Preprocessing* in Figure 4.1. Algorithm 4 presents the procedure of the anomaly detection and removal mechanism implemented in Telescope. As input, the algorithm requires the time series observations and frequency. First, the indices of the anomalies are extracted by applying the method of Hochenbaum *et al.* [HVK17] (cf. Line 2). This method performs a modified version of seasonal and trend decomposition with loess (STL) to split the time series into seasonal, trend, and remainder components. To do this, the frequency of the time series is required, which is the reason for the anomaly removal to be performed after the frequency determination. Subsequently, generalized extreme studentized deviate test (ESD) with median instead of mean and absolute median deviation instead of standard deviation is applied to the remainder to identify outliers. According to Hochenbaum *et al.*, substituting these metrics improves the accuracy of outlier detection [HVK17]. However, especially in many human-generated or influenced time series, missing values, i.e., a kind of anomaly, are usually replaced by zeros and the method presented above cannot detect them as anomalies, so the anomaly detection and removal method implemented in Telescope provides an optional input parameter that defaults to false but can be easily enabled. If this Boolean variable is set to true, the algorithm scans the time series for zeros and adds their indices to the indices of detected anomalies (cf. Lines 5 and 6).

Once all anomalies are identified, they can be removed (cf. Line 9). To this end, the positions of the precursor and the successor of each anomaly are determined (cf. Lines 10 and 11). Note, however, that the precursor and

successor are not necessarily the adjacent observations of the anomaly, but the closest observations that are not anomalies. Finally, each anomaly is set to the mean of its precursor and successor to return the revised time series.

Algorithm 4 Anomaly removal

Input: observation values x, frequency f, Boolean indicating whether zeros need to be removed $remove\_zeros$ (default: FALSE)
Output: corrected values $\widetilde{x}$

1: *### get all indices of anomalies*
2: $\widetilde{x} = x$
3: $indicesAnoms =$ AnomalyDetection(x, f)
4: *### set indices of zero values to anomaly indices if $remove\_zeros$ set to TRUE*
5: **if** ($remove\_zeros$) **then**
6: $indicesZeros =$ findIndicesWithValueZero(x)
7: $indicesAnoms =$ sort(append($indicesAnoms$, $indicesZeros$))
8: **end if**
9: *### replace anomaly values by the mean of the two non-anomaly adjacent values*
10: **for all** ($position$ in $indicesAnoms$) **do**
11: $prev =$ getNonAnomalyPrecursor($position$)
12: $next =$ getNonAnomalySuccessor($position$)
13: $\widetilde{x}[position]=$ mean($prev$, $next$)
14: **end for**
15: **return** $\widetilde{x}$

4.2.3 Trend Tests

The trend tests constitute the final step of the time series preprocessing as shown in the topmost gray box *Preprocessing* in Figure 4.1. Subsequently to the time series decomposition, the trend type, namely linear or exponential (exp), must be determined, since ARIMA cannot handle exponential trends. To this end, Algorithm 5 illustrates the procedure for determining the trend type realized in Telescope. As input features, the algorithm requires only the trend component extracted by the STL decomposition. In a first step, the algorithm fits a linear model to the original trend (cf. Line 2) and employs this model to reconstruct the data (cf. Line 3). Next, a linear model is fitted to the logarithmized trend (cf. Lines 5 and 6). Accordingly, this model represents an exponential trend model. Afterwards, the exponential function is applied

to the data reconstructed with the second linear model (cf. Line 7), so that both reconstructions can be compared. Once both models are learned and the data are reconstructed, the residuals of both models are calculated and can be compared. Therefore, the root mean square error (RMSE) is derived for both models based on the original observations (cf. Lines 9 and 10). Finally, the type of trend with smaller residuals is returned. However, since the exponential model often appears almost like a linear model due to the very small exponential growth, a tolerance factor is included in the comparison to avoid such misbehavior (cf. Line 11).

Algorithm 5 Fitting linear or exponential model

Input: trend determined by stl $trend$
Output: linear or exp according to the best fitting model $model$

1: *### determine linear model*
2: $linear\_model = \text{lm}(trend)$
3: $counts\_linear = \text{predict}(linear\_model)$
4: *### determine exponential model*
5: $log\_trend = \log(trend)$
6: $exp\_model = \text{lm}(log\_trend)$
7: $counts\_exp = \exp(\text{predict}(exp\_model))$
8: *### return the model with smaller RMSE*
9: $linear\_error = \text{rmse}(trend, counts\_linear)$
10: $exp\_error = \text{rmse}(trend, counts\_exp)$
11: $model = \text{argmin}(linear\_error, tolerance \times exp\_error)$
12: **return** $model$

The second trend test examines whether the time series exhibits a significant trend pattern since XGBoost with its default settings cannot handle such time series. However, only one parameter of XGBoost needs to be adjusted to improve its performance for such time series, namely the so-called *booster*. To this date, XGBoost offers three different boosting methods, namely gbtree, gblinear, and dart. While gbtree and dart construct trees, gblinear uses a linear function for modeling. Therefore, gblinear is capable of modeling linear trends that exceed the range of input values, while gbtree and dart should not be used for such tasks. Nonetheless, gbtree is preferable for time series without strong linear trends, as it outperforms gblinear for such time series of more stationary nature. To this end, inspecting the proportion of trend in the forecast horizon of the time series is again a crucial component for parameter tuning of XGBoost.

However, since the proportion of trend in the forecasting horizon is unknown at modeling time, it must be estimated. The heuristic implemented in Telescope is shown in Algorithm 6. As input features, the algorithm requires the time series observations, the frequency, and the length of the forecasting horizon. First, the time series is decomposed using STL (cf. Line 2) and the trend component is extracted (cf. Line 3). Here, the parametrization of STL is essential, since Telescope by definition considers only seasonal time series. If *t.window* is not set or is set too short, seasonal patterns often remain in the trend component, which reduces the quality of the decomposition. Next, the first trend test given

Algorithm 6 Test trend

Input: observation values $\widetilde{x}$, frequency f, length of the forecast h
Output: TRUE if there is a strong trend pattern, FALSE otherwise

1: *### decompose time series*
2: $decomp =$ stl(ts($\widetilde{x}$, f), s.window = "periodic", t.window = length($\widetilde{x}$)/2)
3: $trend = decomp$["trend"]
4: *### fit the right model (linear or exponential)*
5: $model =$ fittingModels($\widetilde{x}$, f)
6: *### preprocessing of data and forecasting the trend*
7: **if** ($model ==$ exp) **then**
8: $trend =$ log($trend$)
9: $arima =$ auto.arima($trend$, stepwise = TRUE, seasonal = FALSE)
10: $forecast\_trend =$ forecast($arima$, h)
11: $forecast\_trend =$ exp($forecast\_trend$)
12: **else**
13: $arima =$ auto.arima($trend$, stepwise = TRUE, seasonal = FALSE)
14: $forecast\_trend =$ forecast($arima$, h)
15: **end if**
16: *### determine the proper booster*
17: $booster =$ getBooster($forecast\_trend$, $trend$, $\widetilde{x}$)
18: **if** ($booster ==$ gblinear) **then**
19: $has\_trend =$ TRUE
20: **else**
21: $has\_trend =$ FALSE
22: **end if**
23: **return** has_trend

by Algorithm 5 is applied to determine the trend type (cf. Line 5). In the case of an exponential trend, the time series must be logarithmized, since ARIMA cannot handle such strong trends (cf. Line 8). Subsequently, ARIMA is applied to model and forecast the trend according to the length of the forecasting horizon (cf. Lines 9, 10, 13, and 14). Note that ARIMA is explicitly used with seasonality disabled. Many implementations of ARIMA include variations of the ARIMA model, such as sARIMA for seasonal time series. However, since Telescope decomposes the time series and only considers the trend component here, these variations of ARIMA are not required for this task. Instead, these variations would only increase the runtime, which is the reason that they are explicitly excluded here. Again, in the case of an exponential trend, the exponential function must be applied to the forecast to transform it back to the original scale (cf. Line 11). Once the trend is forecast, the proper boosting method is derived (cf. Line 17). The procedure of booster determination is described in Algorithm 12 in Section 4.5. If the algorithm for booster determination returns gblinear, a strong trend pattern is assumed and the trend test returns true. In contrast, if gbtree is delivered by Algorithm 12, false is returned.

4.3 Creation of Categorical Information

Aiming to further improve forecast accuracy, Telescope extracts additional categorical information from the time series. To this end, Section 4.3.1 describes the partitioning of the time series into individual periods, the extraction of statistical characteristics of these periods, and their clustering. However, as forecasts of these categorical information are also required, Section 4.3.2 presents the procedure for cluster label forecasting.

4.3.1 Clustering of Single Periods

As a general rule, the predictive power of machine learning improves as more covariates are added, up to a certain point where the model overfits and, thus, deteriorates again. Since the core component of Telescope uses XGBoost, a machine learning technique, covariates are desirable. In addition to the seasonal and trend components of the STSM model, further categorical information can be extracted from the time series. However, extracting meaningful categorical information from univariate time series is a challenging task. The approach implemented in Telescope is based on clustering the individual periods of the time series (cf. the two topmost white boxes of the gray box *Creation of Categorical Information* in Figure 4.1). To this end, Algorithm 7 presents the workflow of

the clustering approach. The algorithm takes the time series observations and frequency as input and first divides the time series into individual periods based on the specified frequency, as this represents the length of the seasonal pattern (cf. Line 2). Then, the statistical characteristics variance and range are calculated for each period (cf. Lines 4-8). After normalizing both characteristics to the range of zero to one (cf. Lines 10-11), k-Means clustering is applied to this feature space (cf. Line 12). Given that additional information is hard to estimate, the number of clusters is set to two, as too many clusters might lead to insufficient distinctions between clusters. Finally, the cluster centroids are returned if the cluster has reached a certain quality level.

Algorithm 7 Cluster periods

Input: observation values $\widetilde{x}$, frequency f
Output: cluster labels for all periods $clusters$

```
1:  ### split the time series into single periods
2:  periods = splitIntoSinglePeriods(x̃, f)
3:  ### calculate characteristics for each period
4:  for all (period in periods) do
5:      vals = getValuesInPeriod(period)
6:      var = cbind(var, variance(vals))
7:      range = cbind(range, max(vals) − min(vals))
8:  end for
9:  ### normalize characteristics
10: range = normalize(range)
11: var = normalize(var)
12: clusters = kmeans(range, var, centers = 2)
13: ### only return clustering if it is good enough
14: silhouette = calculateSilhouette(clusters, dissimilarity(var, range))
15: if (silhouette < 0.75) then
16:     clusters = −1
17: end if
18: return clusters
```

However, clustering does not extract valuable information for all types of time series. In some cases, categorical information produced by the application of clustering methods can drastically reduce the forecast accuracy. For instance, this is the case for the time series shown in Figure 4.2, which displays the quarterly gas production in Australia from 1956 to 2010 in petajoules.

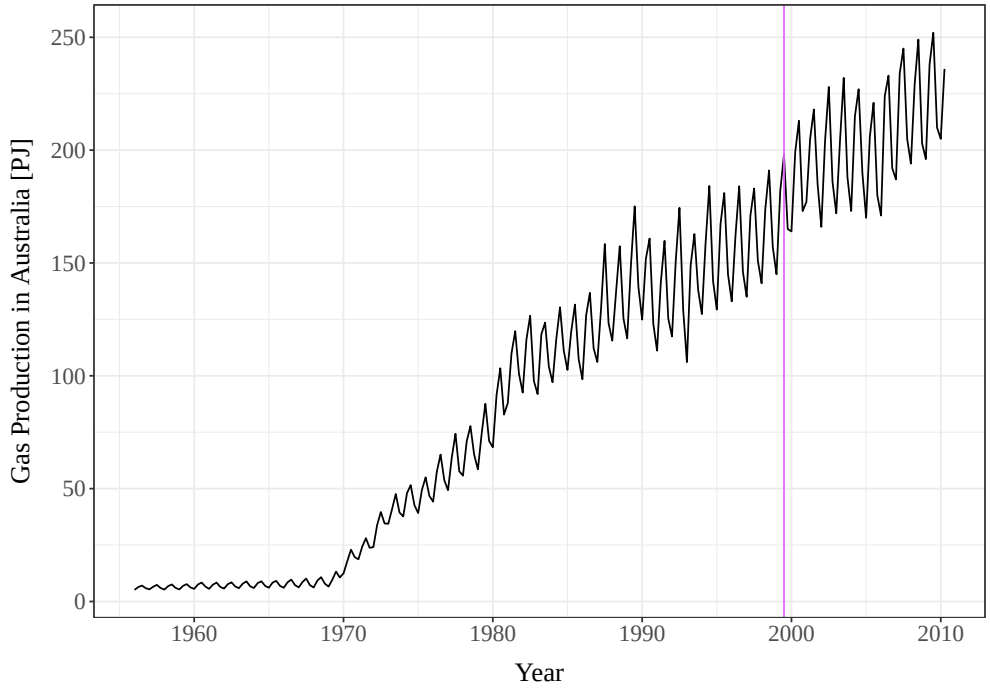


Figure 4.2: The quarterly gas production in Australia [petajoules] from 1956 to 2010.

This time series exhibits seasonal and trend patterns as well as a multiplicative decomposition. Here, the first 80% of the observations are used for model training, while the remaining 20% are used for forecast evaluation. The end of the training part is indicated by the vertical purple line in Figure 4.2. The clustering procedure described above provides the following centroid vector for the training part, where each value represents one period:

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1

As can be seen, the centroids in the history fluctuate, so forecasting this vector of centroids leads to both types of centroids in the forecasting horizon, although it is evident from Figure 4.2 that actually only centroid 1 would be reasonable. For this reason, Algorithm 7 utilizes the silhouette coefficient as a verification that the cluster labels are meaningful (cf. Line 14). The silhouette coefficient is a measure of the consistency of the clustering and ranges from -1 to 1. That is, the silhouette coefficient compares the cohesion, i.e., how similar an object is to its cluster, with the separation, i.e., how dissimilar an object is to the other clusters. If the silhouette coefficient is greater than a certain threshold (i.e., 0.75), the cluster structure can be considered strong. A silhouette coefficient between 0.5

and 0.75 implies a medium clustering quality. If the silhouette coefficient is less than 0.5, the clustering quality is considered insufficient. Hence, Algorithm 7 requires a silhouette coefficient of at least 0.75. Only if the silhouette coefficient reaches this threshold, the clustering is further used. Otherwise, clustering is discarded and only the STSM components are used as covariates for XGBoost.

4.3.2 Cluster Label Forecasting

Following the clustering of each period in the time series history, the centroids must be forecast so that they can be used as covariates for XGBoost, as depicted by the lowermost white box of the gray box *Creation of Categorical Information* in Figure 4.1. The procedure for forecasting cluster labels employed in Telescope is outlined in Algorithm 8. As input, the algorithm assumes the period cluster labels, time series observations, time series frequency, number of repetitions to learn the ANN, and the length of the forecasting horizon. First, the ANN is trained for the specified number of repetitions (cf. Line 2). That is, the ANN is trained with as many different starting weights as the number of repetitions specifies. Their mean is then used to forecast the cluster labels (cf. Line 3). However, since these forecasts represent the centroid for an entire period, the cluster label must be repeated for every single observation in that period. For this purpose, the frequency of the time series is required, since each cluster label must be repeated as many times as there are observations within a period. The only exception might be the last centroid, as this period might not be complete. Then, the derived cluster labels for each observation are returned.

Algorithm 8 Forecast cluster labels

Input: cluster labels $clusters$, frequency f, observation values $\widetilde{x}$, number of repetitions for ANN $repeats$, length of the forecast h
Output: forecast of cluster labels $labels$

1: *### forecast cluster labels*
2: $ann =$ nnetar$(clusters, repeats)$
3: $forecast\_cluster =$ forecast(ann, h)
4: *### repeat each cluster label f-times*
5: $labels =$ repeatClusterCenterForEntirePeriod$(forecast_cluster, \widetilde{x}, f)$
6: **return** $labels$

4.4 Decomposition and Component Forecasting

Given that Telescope uses a kind of divide-and-conquer approach to forecast univariate, seasonal time series, the time series must be broken down into components that are easier to forecast. To this end, Section 4.4.1 describes the time series decomposition applied, while Section 4.4.2 explains the forecasting procedure applied to the derived time series components.

4.4.1 Time Series Decomposition

As mentioned above, Telescope applies STL decomposition, which separates the original time series into seasonal, trend, and remainder components [CCMT90] according to the STSM model (cf. the gray box *Decomposition* in Figure 4.1). However, since STL cannot handle multiplicative decomposition, the type of decomposition must be estimated beforehand. Therefore, a decision logic is implemented in Telescope that examines the time series to determine whether the time series has an additive or multiplicative composition type. This procedure is presented in Algorithm 9 and requires only the time series observations and frequency as input. In the first step, the procedure computes the STL decomposition on the original time series (cf. Line 2), which represents the additive composition type. Subsequently, the STL decomposition is also computed on the logarithmized values of the time series (cf. Line 3), which represents the multiplicative composition type. Subsequently, only the remainder components of the STL decompositions are used for decision-making (cf. Lines 5-6). For the multiplicative composition type, the exponential function is applied to scale the remainder component back to its original scale. As a first heuristic, the sum of squares is computed for both remainder components (cf. Lines 8-9). A second decision heuristic is obtained using the interquartile range, again computed on both remainder components (cf. Lines 11-12).

Furthermore, an optional Boolean parameter can be passed to the algorithm, which is set to false by default. If this parameter is enabled, the autocorrelation function is applied to the remainder components and the sum of squares are computed on them (cf. Lines 16-17). By applying the autocorrelation function to the remainders, the correlations of the residuals with a lagged version of themselves are determined for several lags. This involves examining how much information, such as repeating patterns, remains in the residuals. Finally, if at least two of the three multiplicative decomposition heuristics exceed the additive decomposition heuristics, the composition type is set to additive, otherwise it is set to multiplicative. However, if autocorrelation is disabled, both multiplicative decomposition heuristics must exceed the additive decomposition

Algorithm 9 Test for multiplicative decomposition

Input: observation values $\widetilde{x}$, frequency f, Boolean ACF (default: FALSE)
Output: TRUE if decomposition is of type multiplicative, FALSE if additive

perform additive and multiplicative STL decompositions
$decomp_{add}$ = stl(ts($\widetilde{x}$, f), s.window = "periodic", t.window = length($\widetilde{x}$)/2)
$decomp_{mult}$ = stl(ts(log($\widetilde{x}$), f), s.window = "periodic", t.window = length($\widetilde{x}$)/2)

determine remainder for the additive and multiplicative decompositions
$res_{add} = decomp_{add}$["remainder"]
$res_{mult} = \widetilde{x} - \exp(decomp_{mult}$["trend"]$) \times \exp(decomp_{mult}$["season"])

calculate the sum of squares of residuals for the additive and multiplicative decompositions
$ssres_{add}$ = sum(square(res_{add}))
$ssres_{mult}$ = sum(square(res_{mult}))

calculate the range from the 25% to the 75% quantiles of the residuals for the additive and multiplicative decompositions
$qrange_{add}$ = percentile(res_{add}, 75%) − percentile(res_{add}, 25%)
$qrange_{mult}$ = percentile(res_{mult}, 75%) − percentile(res_{mult}, 25%)

decide via majority decision which decomposition to choose
is_mult = TRUE
if (ACF) **then**
    ### calculate the sum of squares of the ACF of the residuals for the additive and multiplicative decompositions
    $ssacf_{add}$ = sum(square(acf(res_{add})))
    $ssacf_{mult}$ = sum(square(acf(res_{mult})))
    $bools$ = c(($ssres_{add} < ssres_{mult}$),($qrange_{add} < qrange_{mult}$), ($ssacf_{add} < ssacf_{mult}$))
    if (count($bools$, TRUE) ≥ length($bools$)/2) **then**
        is_mult = FALSE
    end if
else if ($ssres_{add} < ssres_{mult}$ && $qrange_{add} < qrange_{mult}$) **then**
    is_mult = FALSE
end if
return is_mult

heuristics to set the composition type to additive. In this case, a multiplicative composition type is returned if the additive decomposition heuristics exceed the multiplicative decomposition heuristics for at least one heuristic.

In case a multiplicative composition is detected, the logarithm must be applied to the time series before proceeding with the STL decomposition. In addition, as mentioned in Section 4.2.3, it is essential to set *s.window* to periodic and *t.window* to half the length of the history. However, when using multiplicative decomposition, the exponential function must be applied to the final forecast to re-transform the scale of the time series.

4.4.2 Season and Trend Forecasting

The forecasting of the trend and season components is shown in the gray box *Component Forecasting* in Figure 4.1. By definition (cf. Section 2.2.2), seasonality is an ever-recurring pattern, so the seasonal pattern extracted by the STL is continued for the entire forecasting horizon. In contrast, trend forecasting is a little more complex. The procedure for forecasting trend patterns within Telescope is shown in Algorithm 10. As input, the algorithm requires the trend component extracted by STL, the type of trend (i.e., linear or exponential), and the length of the forecasting horizon. If the time series exhibits an exponential

Algorithm 10 Forecast trend

Input: trend determined by stl $trend$, model $model$, length of the forecast h
Output: trend forecast $forecast\_trend$

```
### logarithmize the time series if the model is detected to be exponential
if (model == exp) then
    trend = log(trend)
    ### apply ARIMA without seasonality for fast trend forecasts
    arima = auto.arima(trend, stepwise = TRUE, seasonal = FALSE)
    forecast_trend = forecast(arima, h)
    ### apply the exponential function to the logarithmized forecast
    forecast_trend = exp(forecast_trend)
else
    arima = auto.arima(trend, stepwise = TRUE, seasonal = FALSE)
    forecast_trend = forecast(arima, h)
end if
return forecast_trend
```

trend, the time series must be logarithmized since ARIMA cannot handle exponential trends (cf. Line 3). Then, the trend is forecast using an ARIMA model (cf. Lines 5-6). As mentioned in Section 4.2.3, only standard ARIMA, especially without seasonality, is used to maintain a short runtime. Seasonality can be neglected, since the trend component no longer contains any seasonal patterns. Once ARIMA provides the forecast, the exponential function must be applied to the forecast to re-scale the values (cf. Line 8). If the time series instead exhibits a linear trend, the ARIMA modeling follows the same approach, but logarithmization and exponentialization are omitted (cf. Lines 10-11). Finally, the trend forecast is returned.

4.5 Remainder Learning and Component Combination

The key component of Telescope is the applied machine learning method, namely XGBoost. Within Telescope, XGBoost implicitly performs two main tasks: Regressing the remainder component and combining the other forecast time series with this remainder forecast. This last step of the Telescope workflow is illustrated by the lowermost gray box *Remainder Forecasting & Combination* in Figure 4.1. To this end, an XGBoost model is learned with the cluster labels, seasonal component, and trend component as covariates and the time series observations as targets. However, as explained in Section 4.1, the trend component is omitted from the covariates if the time series has an exponential trend. The workflow of Telescope's remainder forecasting and time series combination using XGBoost is illustrated in Algorithm 11. To regress the remainder and combine the individual components of the covariates, the historical covariates (i.e., the trend and season components extracted from STL and the cluster labels derived from the historical observations), the historical time series observations, the forecast covariates (i.e., the forecasts of trend, season, and cluster labels), and the correct boosting method determined by Algorithm 12 must be passed to the workflow. A major drawback of XGBoost is that it tends to overfit to the training data, so the historical data must be split into training and validation parts. Hence, 20% of the entries in the historical covariate matrix are randomly sampled for training, while the remaining 80% are held back to validate the learned model (cf. Lines 2-4). Regarding XGBoost, this set of training and validation data must be combined into a list called watchlist (cf. Line 5). In addition, numerous other parameters of XGBoost need to be set. To this end, the algorithm loads a pre-defined list of parameters, which is specified in

Algorithm 11 Train XGBoost model

Input: historical covariate matrix $hist\_covariates$, forecast covariate matrix $forecast\_covariates$, observation values $\widetilde{x}$, boosting method $booster$
Output: final forecasting result $forecast\_xgb$

1: *### split the time series in training and test parts to avoid overfitting*
2: $h =$ sample(nrow($hist\_covariates$),floor(0.2×nrow($hist\_covariates$))
3: $dtrain =$ xgb.DMatrix($hist\_covariates[h]$, $\widetilde{x}[h]$)
4: $dval =$ xgb.DMatrix($hist\_covariates[-h]$, $\widetilde{x}[-h]$)
5: $watchlist =$ list($dtrain$, $dval$)
6: *### get the parameters from Table 4.1*
7: $param\_list$ = getParameters(Table 4.1)
8: *### learn and apply the XGBoost model to return the final forecast*
9: xgb = xgb.train(data = $dtrain$, watchlist = $watchlist$, params = $param\_list$)
10: $forecast\_xgb =$ predict(xgb, $forecast\_covariates$)
11: **return** $forecast\_xgb$

Table 4.1[2] . The only parameter that is not fixed is the boosting method, as this is set to either gbtree or gblinear depending on the trend pattern. The decision logic for this choice is shown in Algorithm 12. Once the covariates are split into training and validation data and the XGBoost parameters are loaded, the XGBoost model is learned (cf. Line 9). Finally, this model is applied to the forecast covariates to regress the final forecast result (cf. Line 10).

As mentioned in the previous paragraph, the boosting method must be chosen before learning the XGBoost model. The booster's performance is highly dependent on the trend behavior of the time series, as gblinear can handle linear trends, while gbtree should not be used for time series with significant trend (cf. Section 4.2.3). Nevertheless, gbtree is superior for modeling time series without a significant trend pattern. Therefore, Telescope implements a decision logic for selecting the type of boosting method, which is shown in Algorithm 12. The decision logic requires the forecast of the trend and the historical observation values as input. In the first step, the algorithm computes the ranges of the trend forecast and the historical observation values (cf. Lines 2-3). The range is defined as the maximum difference between two observations and, therefore, the difference between the maximum value and the minimum

[2]For more details on the parameters of XGBoost, please refer to the documentation of XGBoost: `https://xgboost.readthedocs.io/en/latest/parameter.html`.

Table 4.1: Parameter settings of XGBoost.

Parameter	Value Assignment
objective	"reg:linear"
booster	$booster$ (gbtree or gblinear)
eta	0.1
max_depth	5
min_child_weight	1
num_parallel_tree	2
nthread	2
nrounds	500
early_stop_rounds	50
maximize	FALSE

value. If the range of the trend represents a large proportion compared with the total range of historical observations, the trend is considered significant. For this purpose, the ratio between these ranges is compared with a threshold, which is set to 0.05 in this thesis. Finally, if the ratio is below this threshold, the algorithm recommends the use of gbtree and otherwise gblinear.

Algorithm 12 Determine booster

Input: trend forecast $forecast\_trend$, observation values $\widetilde{x}$
Output: booster recommendation $booster$

determine the range of the trend in the history and in the forecast
$forecast\_trend\_range = \max(forecast\_trend) - \min(forecast\_trend)$
$hist\_range = \max(\widetilde{x}) - \min(\widetilde{x})$

compare the ratio of the trend range in the forecast with the trend range in the history
if $(forecast\_trend\_range/hist\_range < threshold)$ **then**
    $booster =$ gbtree
else
    $booster =$ gblinear
end if

return the booster according to the comparison
return $booster$

4.6 Summary and Discussion

We conclude this chapter by briefly summarizing the main contribution in answering the research question **RQ A.1** formulated in Section 1.3. Research question **RQ A.1** addresses the challenge of developing a novel hybrid forecasting method based on time series decomposition. To this end, we have introduced Telescope, which employs seasonal and trend decomposition with loess (STL) to split a time series into seasonal, trend, and remainder components. However, prior to time series decomposition, Telescope preprocesses the time series by means of frequency estimation, anomaly detection and removal, trend type determination, composition type examination, and a shift of the values, if necessary. Once the time series components are extracted, the trend and seasonal components are forecast separately using ARIMA and continuation, respectively. However, the remainder component is not forecast directly. Apart from univariate time series forecasting, categorical information are also extracted from the time series to further improve the forecast quality. To forecast the categorical information, NNetAR is deployed in Telescope. Finally, Telescope applies XGBoost to regress the remainder component using the forecasts of the other time series components and the categorical information as features. Thus, Telescope provides a fully automated end-to-end forecasting workflow that includes numerous sophisticated time series preprocessing tasks to facilitate the forecasting process for the operator.

As the "No Free Lunch Theorem" states, there is no single method that performs best for all types of data. This also applies to Telescope, so the following assumptions are made. First, Telescope is specifically designed for univariate time series. Second, due to the application of STL, Telescope is only applicable to seasonal time series with more than two full periods of observation data. Third, the length of the seasonal pattern exhibited by the time series must not vary over time. However, multiple overlapping seasonal patterns with different frequencies are allowed. Finally, Telescope is designed for multi-step-ahead forecasting of long time series with comparatively many observations per period. Nevertheless, Telescope can also be used for other time series and one-step-ahead forecasting.

Chapter 5

Evaluation of Telescope

In this chapter, we evaluate the forecasting performance of Telescope and compare it with several state-of-the-art time series forecasting methods with respect to average forecast accuracy, robustness in forecast accuracy, and required runtime for the entire forecasting task including model learning and forecasting. To this end, we first present the general evaluation design in Section 5.1 and then provide the results of the experiments performed in Section 5.2. Furthermore, we demonstrate the effectiveness and benefits of Telescope in an application scenario where virtual machines are automatically scaled according to workload demand forecasts in Section 5.3. Finally, we conclude this chapter with a concise discussion in Section 5.4.

5.1 Evaluation Design

As the threshold setting for critical event prediction depends on the particular use case and requires a priori domain knowledge, we did not evaluate explicit threshold exceedances, but the forecast quality in general. This is also representative for critical event prediction, since a more precise forecasting method in turn yields less forecast errors and is therefore more applicable to the prediction of critical events.

In order to assess the forecast quality of Telescope and compare it with several existing state-of-the-art forecasting methods, a data set consisting of 53 different seasonal time series was used. The names, sources, and lengths of these time series are summarized in Table 5.1. This summary shows that the time series originate from many different sources representing a variety of use cases. Furthermore, the time series exhibit highly varying lengths, with the shortest time series consisting of only 68 observations, while the longest time series contains 33,795 observations.

Table 5.1: List of all seasonal time series used for the evaluation of Telescope.

Name	Source	Length
Sales	Time Series Data Library [HY18]	2820
Gasoline	Hyndman [DLHS11]	745
TurkishElectricity	Hyndman [DLHS11]	3288
CarSalesQuebec	DataMarket [Bro17]	108
Taylor	`forecast` R package [HAB+18]	4032
Gas	`forecast` R package [HAB+18]	476
WineInd	`forecast` R package [HAB+18]	176
a10	`fpp2` R package [Hyn18]	204
Arrivals2	`fpp2` R package [Hyn18]	127
Arrivals3	`fpp2` R package [Hyn18]	127
AusBeer	`fpp2` R package [Hyn18]	218
AusCafe	`fpp2` R package [Hyn18]	416
AusTourists	`fpp2` R package [Hyn18]	68
Calls	`fpp2` R package [Hyn18]	27716
DebitCards	`fpp2` R package [Hyn18]	164
Departures1	`fpp2` R package [Hyn18]	491
Departures2	`fpp2` R package [Hyn18]	491
Departures3	`fpp2` R package [Hyn18]	491
Departures4	`fpp2` R package [Hyn18]	491
Departures5	`fpp2` R package [Hyn18]	491
ElecDemand1	`fpp2` R package [Hyn18]	17520
Hyndsight	`fpp2` R package [Hyn18]	365
QausElec	`fpp2` R package [Hyn18]	218
QCement	`fpp2` R package [Hyn18]	233
QGas	`fpp2` R package [Hyn18]	218
USMElec	`fpp2` R package [Hyn18]	486
VN2	`fpp2` R package [Hyn18]	72
VN4	`fpp2` R package [Hyn18]	72
AirPassengers	`datasets` R package [R C18]	144
ChickWeight$weight	`datasets` R package [R C18]	578
CO2	`datasets` R package [R C18]	468
EuroDist	`datasets` R package [R C18]	210
FDeaths	`datasets` R package [R C18]	72
LDeaths	`datasets` R package [R C18]	72
Loblolly$height	`datasets` R package [R C18]	84

Continued

Name	Source	Length
MDeaths	`datasets` R package [R C18]	72
Nottem	`datasets` R package [R C18]	240
Seatbelts1	`datasets` R package [R C18]	192
Seatbelts2	`datasets` R package [R C18]	192
Seatbelts3	`datasets` R package [R C18]	192
Seatbelts4	`datasets` R package [R C18]	192
Seatbelts5	`datasets` R package [R C18]	192
Treering	`datasets` R package [R C18]	7980
UKDriverDeaths	`datasets` R package [R C18]	192
UKGas	`datasets` R package [R C18]	108
USAccDeaths	`datasets` R package [R C18]	72
Wikipedia	Wikipedia Project-Counts[1]	712
IBM	N. Herbst *et al.* [HHKA14]	2670
NASA	Internet Traffic Archive[2]	5636
FIFA	Internet Traffic Archive[2]	3711
Calgary	Internet Traffic Archive[2]	33795
ClarkNet	Internet Traffic Archive[2]	1324
Saskatchewan	Internet Traffic Archive[2]	20524

As Telescope is designed to perform multi-step-ahead forecasting for time series with many observations per period, the first 80% of time series observations are used to train the forecasting model, while the last 20% of time series observations are all forecast at once. Note that such multi-step-ahead forecasting may result in a forecast of several hundreds or thousands of values applying the forecasting method only once.

In order to quantify the forecast error of the applied forecasting methods, three forecast error measures are calculated, namely mean relative absolute error based on a posteriori Naïve forecast, mean absolute scaled error, and mean absolute percentage error (cf. Section 2.6.1). Note that the baseline generated by the a posteriori Naïve forecast is only a theoretical calculation, since it requires the latest observation and forecasts exactly that value. It would be possible to forecast the last value of the training part of the time series for the entire forecast horizon, however, this would result in a constant forecast with inferior forecast insight. Therefore, we decided to use a posteriori knowledge

[1]Wikipedia Project-Counts: `http://dumps.wikimedia.org/other/pagecounts-raw/`
[2]Internet Traffic Archive: `http://ita.ee.lbl.gov/html/traces.html`

for baseline calculation. In addition to the forecast error measures, we also captured the time required for the entire forecasting task, which includes both model training as well as forecasting.

As state-of-the-art forecasting methods, we compare Telescope with the four statistical models ARIMA, ETS, sNaïve, and TBATS as well as with the four machine learning models NNetAR, Random Forest, Support Vector Machine, and XGBoost. For more details on the statistical and machine learning methods, please refer to Sections 2.3 and 2.4, respectively.

In the following, we report several aggregations of the achieved performances, namely the average over all time series, the standard deviation of the performance delivered across the data set, and the ranks per forecasting method. Regarding the rank, the forecasting method with the best performance, i.e., the lowest forecast error or the shortest runtime, receives rank 1, while the rank increases gradually according to the ordered performance of the forecasting methods. That is, the worst forecasting method receives rank 9, as there are nine forecasting methods in competition. Also, the rank is calculated per evaluation measure. Thereby, for all evaluation measures, the smaller the reported value, the better the performance provided.

5.2 Comparing Forecast Accuracy and Time-to-Result

First, we present detailed insights into the forecast accuracy of Telescope and the eight state-of-the-art methods for two well-known benchmark time series for forecasting seasonal time series, namely Taylor's Electricity Demand time series and the Airline Passengers trace. We then examine the average forecast performance as well as its standard deviation. Lastly, we present the average rank achieved by each forecasting method for every evaluation measure.

5.2.1 Detailed Forecasting Comparison

This section provides a detailed perspective on the forecasts produced by Telescope and the eight forecasting methods in competition for two representative seasonal time series that are commonly used for assessing forecast accuracy.

5.2.1.1 Taylor's Electricity Demand

The first time series analyzed is Taylor's Electricity Demand that represents the half-hourly electricity demand in England and Wales from June 5, 2000 to August 27, 2000. Thus, the time series consists of 4,032 observations and exhibits two seasonal patterns. The first seasonal pattern is a diurnal seasonality,

i.e., the electricity demand follows a regular pattern based on the time of day. In addition, the time series also shows a weekly seasonal pattern, where the electricity demand depends not only on the time of day, but also on the day of the week. More specifically, the daily maximum is noticeably smaller on weekends than on weekdays. In total, the recorded time series contains 84 periods of the daily seasonal pattern and 12 periods of the weekly seasonal pattern. Finally, the time series does not exhibit any trend pattern.

Figure 5.1 illustrates the forecasts of the nine forecasting methods. The time is depicted on the horizontal axis, while the half-hourly electricity demand is shown on the vertical axis. For visualization purposes, the first eight weekly periods are truncated. However, they were still used for model training, so the training time series consisted of the first 3,226 observations, while the later 806 observations had to be forecast all at once. The solid black line represents the latest observations of the training time series, while the dashed black line shows the actual observations of the testing part. The forecast of Telescope is colored in green, the forecasts of the four statistical methods ARIMA, ETS, sNaïve, and TBATS are colored in increasingly darker shades of red, and the forecasts of the four machine learning methods NNetAR, Random Forest (RF), SVM, and XGBoost are colored in increasingly darker shades of blue.

The first finding is that most machine learning-based forecasting methods, namely Random Forest, SVM, and XGBoost, cannot handle the time series at all, since they only forecast a constant value. This is due to the fact that they only receive the time information as input and no additional information for, e.g., autoregression. Moreover, ETS also does not forecast meaningful values, which is due to the property of ETS not being able to process seasonal patterns with more than 24 observations. In contrast, ARIMA, sNaïve, TBATS, and NNetAR provide good forecasts for weekdays. However, all four of these methods miss the second seasonal pattern, namely the lower peak values for weekends. Note that the forecasts of ARIMA and sNaïve are almost identical and, therefore, overlap most of the time. Only Telescope was able to model both seasonal patterns, which can be seen in the compressed periods on weekends. Therefore, Telescope's forecast is also very close to the actual observations.

In order to analyze the forecast accuracy numerically, Table 5.2 presents the achieved forecasting performance with respect to mean relative absolute error (MRAE), mean absolute scaled error (MASE), mean absolute percentage error (MAPE), and time-to-result. The numerical results provide the same findings as the visual assessment, namely that Telescope provides by far the smallest forecast errors with respect to all three forecast error measures. Telescope achieved an MRAE, MASE, and MAPE of only 1.644, 1.624, and 3.688%,

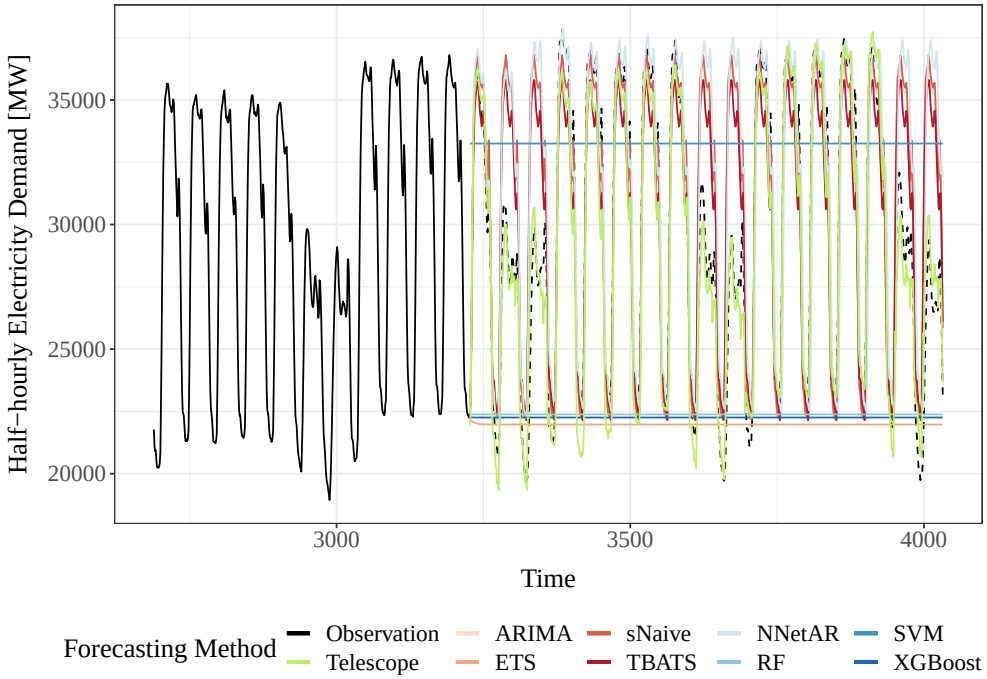


Figure 5.1: The detailed forecasts for Taylor's Electricity Demand time series using Telescope (green), the four statistical methods (reddish), and the four machine learning-based methods (bluish).

respectively. Moreover, the same two groups can be formed. However, the numerical results allow sorting the forecast quality, which was not possible on the basis of Figure 5.1. Although the difference between sNaïve, ARIMA, and TBATS is marginal, sNaïve performed second best, followed by ARIMA and then TBATS, while NNetAR also produced a comparatively small forecast error but still fell off a bit. Finally, the worst forecast accuracy is yielded by ETS, XGBoost, Random Forest, and SVM in descending order. This ranking is supported by all three forecast error measures. In terms of time-to-result, sNaïve delivered the forecast result the fastest, which is intuitive due to its simple forecasting procedure. In addition, the other four methods in the poor performing group also provided their results in less than two seconds. Among the well-performing group, besides sNaïve, only Telescope was able to produce the forecast in less than 2 seconds. In contrast, the other three forecasting methods required between 56 and 424 seconds.

Table 5.2: The achieved forecasting performance of the eight state-of-the-art forecasting methods and Telescope for Taylor's Electricity Demand time series. The best values for each evaluation measure are highlighted in bold.

Forecasting Method	MRAE	MASE	MAPE [%]	Time-to-Result [s]
Telescope	**1.644**	**1.624**	**3.688**	1.933
ARIMA	3.423	3.382	8.003	424.408
ETS	11.998	11.854	23.792	0.305
sNaïve	3.390	3.350	7.982	**0.002**
TBATS	3.565	3.523	8.115	121.210
NNetAR	4.266	4.215	9.956	56.974
Random Forest	11.497	11.360	22.762	0.890
SVM	8.407	8.306	21.323	1.828
XGBoost	11.650	11.511	23.077	0.018

5.2.1.2 Airline Passengers

The second time series used to analyze forecasting performance in more detail is the Airline Passengers trace, which contains the number of monthly international airline passengers from 1949 to 1960 in thousands. The Airline Passengers trace contains 144 values and shows a yearly seasonal pattern, resulting in a total of 12 periods in the time series. Yet, unlike Taylor's Electricity Demand, it does not contain a second seasonal pattern, but instead displays a significant trend. In addition, the Airline Passengers trace exhibits a multiplicative composition type, i.e., the amplitude of the seasonal pattern increases as the trend grows. Figure 5.2 shows the entire time series and the forecasts produced by the nine forecasting methods. The figure structure is similar to Figure 5.1, with the only difference being that the entire training observations are shown. Similar to Taylor's Electricity Demand, the forecasts from Random Forest, SVM, and XGBoost are only constant values. However, here ETS was able to deliver a forecast close to the actual observations, since the seasonal pattern has a length of only 12 values. The forecasts by Telescope, ARIMA, TBATS, and NNetAR also appear very promising, although TBATS tends to overestimate the actual observations. The sNaïve forecast captured the seasonal pattern very well, but failed to model the trend component due to its repeating nature.

Given that the forecasts by Telescope, ARIMA, ETS, and NNetAR are very close to the actual observations and, therefore, it is hard to tell which forecasting method performed best by considering only Figure 5.2, Table 5.3 reports the achieved forecasting performance with respect to MRAE, MASE, MAPE, and

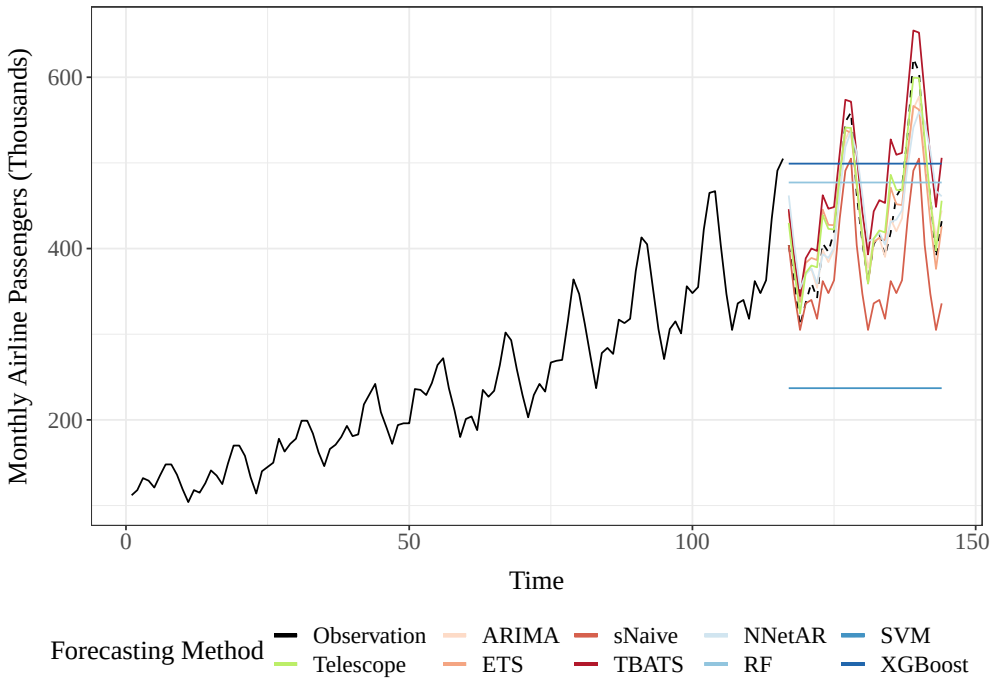


Figure 5.2: The detailed forecasts for the Airline Passengers time series using Telescope (green), the four statistical methods (reddish), and the four machine learning-based methods (bluish).

time-to-result. Again, Telescope achieved the lowest forecast error with respect to all three forecast error measures. Telescope's result showed an MRAE, MASE, and MAPE of 0.367, 0.801, and 4.097%, respectively. Yet, ARIMA performed almost as well as Telescope with an MRAE, MASE, and MAPE of 0.419, 0.914, and 4.422%, respectively. The third and fourth best forecasts were produced by ETS and NNetAR, respectively. TBATS and sNaïve also delivered mediocre forecast accuracy, while Random Forest, XGBoost, and SVM provided only poor forecasts. The ranking of best to worst forecasting methods applies to all three forecast error measures. When analyzing the time-to-result, sNaïve again delivered the fastest forecast. However, when analyzing only the top four forecasting methods, Telescope, ARIMA, ETS, and NNetAR, NNetAR provided the forecast with the shortest time-to-result of only 0.122 seconds, followed by Telescope with 0.520 seconds, ARIMA with 0.604 seconds, and ETS with 0.803 seconds. Compared to Taylor's Electricity Demand, the time-to-result for all methods is negligible due to the short length of the time series.

Table 5.3: The achieved forecasting performance of the eight state-of-the-art forecasting methods and Telescope for the Airline Passengers time series. The best values for each evaluation measure are highlighted in bold.

Forecasting Method	MRAE	MASE	MAPE [%]	Time-to-Result [s]
Telescope	**0.367**	**0.801**	**4.097**	0.520
ARIMA	0.419	0.914	4.422	0.604
ETS	0.505	1.102	5.461	0.803
sNaïve	1.375	3.000	13.804	**0.001**
TBATS	1.006	2.195	10.916	1.815
NNetAR	0.719	1.569	7.590	0.122
Random Forest	1.657	3.616	18.620	0.008
SVM	4.386	9.571	44.237	0.004
XGBoost	1.931	4.214	22.112	0.013

5.2.2 Average and Variation in Forecast Accuracy

In this section, the nine forecasting methods are analyzed across the entire data set rather than for individual time series. Therefore, Table 5.4 presents the average achieved forecasting performance of Telescope and the eight state-of-the-art forecasting methods in competition with respect to the three forecast error measures MRAE, MASE, and MAPE as well as the time-to-result.

Regarding average forecast accuracy, Telescope clearly achieved the least forecast errors, as it yielded the best mean value for all three forecast error measures. Speaking in numbers, Telescope reached an average MRAE, MASE, and MAPE of 1.501, 1.959, and 33.553%, respectively. The second best forecasting method was ARIMA, although its average forecast error measures were already 21% (MRAE), 22% (MASE), and 25% (MAPE) higher compared with Telescope. The other seven forecasting methods were even worse, with sNaïve, TBATS, and NNetAR still achieving useful forecasts, while the forecast error measures of ETS, Random Forest, SVM, and XGBoost indicate only inferior forecast accuracy. This fact clearly demonstrates the superiority of Telescope for seasonal time series. Regarding the time required for model creation and forest delivery, it can be observed that especially the poorly performing methods achieved the shortest time-to-result. The only exception is sNaïve, which achieved the shortest time-to-result due to its simplicity. However, sNaïve cannot handle trend patterns, which means that in practice, it should only be used when it can be guaranteed that the time series does not exhibit long-term movements. However, considering only the other well-performing forecasting methods

Table 5.4: The average achieved forecasting performance of the eight state-of-the-art forecasting methods and Telescope. The best values for each evaluation measure are highlighted in bold.

Forecasting Method	MRAE	MASE	MAPE [%]	Time-to-Result [s]
Telescope	**1.501**	**1.959**	**33.553**	3.105
ARIMA	1.819	2.394	42.052	52.096
ETS	2.555	3.090	94.077	0.719
sNaïve	2.079	2.787	35.671	**0.009**
TBATS	2.006	2.792	49.588	15.588
NNetAR	2.077	2.879	54.048	32.697
Random Forest	2.851	3.603	93.989	1.866
SVM	3.832	5.711	59.641	6.489
XGBoost	2.895	3.651	91.961	0.022

that are also able to handle trend patterns, i.e., Telescope, ARIMA, TBATS, and NNetAR, Telescope clearly required the shortest time-to-result with an average of only 3.105 seconds. In contrast, ARIMA, TBATS, and NNetAR delivered the forecast after an average duration of 52.096 seconds, 15.599 seconds, and 32.697 seconds, respectively. Thus, the speed-up achieved by Telescope reaches values of about 1578% compared with ARIMA, 402% compared with TBATS, and 953% compared with NNetAR. Consequently, Telescope not only achieved considerably higher forecast accuracy, but also significantly shorter time-to-result compared with the other well-performing forecasting methods.

In addition to achieving high average forecast accuracy and short average time-to-result, another design goal of Telescope is to provide robust results. Therefore, Table 5.5 shows the standard deviations of the three forecast error measurements and the time-to-result for Telescope and the eight state-of-the-art forecasting methods in competition over the entire set of time series.

With respect to MRAE and MASE, Table 5.5 shows that Telescope also achieved the smallest standard deviation, indicating robust and, hence, reliable forecast quality. For these measures, ARIMA again scored the second lowest, although far behind. Only with respect to MAPE, sNaïve provided a smaller standard deviation compared to Telescope. Nevertheless, Telescope closely follows sNaïve, while all other forecasting methods had a standard deviation of MAPE of at least about 70% more than that achieved by sNaïve. Similar to the average time-to-result, most poorly performing forecasting methods, namely ETS, Random Forest, and XGBoost, showed little variation in the time-to-result. Only SVM exhibited larger variations than Telescope. Based on

Table 5.5: The standard deviation of forecasting performance across the 53 time series for the eight state-of-the-art forecasting methods and Telescope. The best values for each evaluation measure are highlighted in bold.

Forecasting Method	MRAE	MASE	MAPE [%]	Time-to-Result [s]
Telescope	**1.536**	**2.563**	69.522	11.756
ARIMA	2.386	3.747	111.410	178.301
ETS	4.202	4.860	388.705	0.653
sNaïve	2.497	3.863	**65.441**	0.051
TBATS	2.595	4.425	126.322	37.041
NNetAR	2.719	4.783	152.103	109.268
Random Forest	3.568	4.370	362.623	6.623
SVM	4.261	7.004	116.731	24.678
XGBoost	3.422	4.166	348.516	**0.027**

the simplicity of sNaïve's forecasting procedure, its standard deviation of time-to-result is very small, although XGBoost yielded an even smaller standard deviation. Comparing Telescope with the three main competitors, ARIMA, TBATS, and NNetAR, Telescope again achieved the least variation in time-to-result. In particular, ARIMA and NNetAR show a drastically higher standard deviation in time-to-result, which makes both of them hardly applicable for applications with real-time requirements, such as virtual machine auto-scaling. Concluding, Telescope provides more robust and reliable forecast accuracy as well as time-to-result compared to the state-of-the-art forecasting methods.

Even more details on the robustness of the delivered forecast accuracy as well as the time-to-result of the nine different forecasting methods can be derived from Figure 5.3, Figure 5.4, Figure 5.5, and Figure 5.6, which show violin plots of the achieved MRAE, MASE, MAPE, and time-to-result, respectively. To this end, the different forecasting methods are shown on the horizontal axes, while the achieved values are shown on the vertical axes. Due to high maximum values, the vertical axes are scaled logarithmically.

Figure 5.3 shows that Random Forest, Support Vector Machine, and XGBoost exhibit only inferior forecast accuracy with respect to MRAE, as their peak values are comparatively high, their minimum values are very high, and their box plots are also at a much higher level. ETS also shows a long tail toward high MRAEs. Again, this can be explained by the fact that ETS cannot handle seasonal patterns with more than 24 observations per period. A similar violin is shown by sNaïve, although the extreme values are less low respectively high. In addition, the box plot of sNaïve is at a considerably higher level, especially

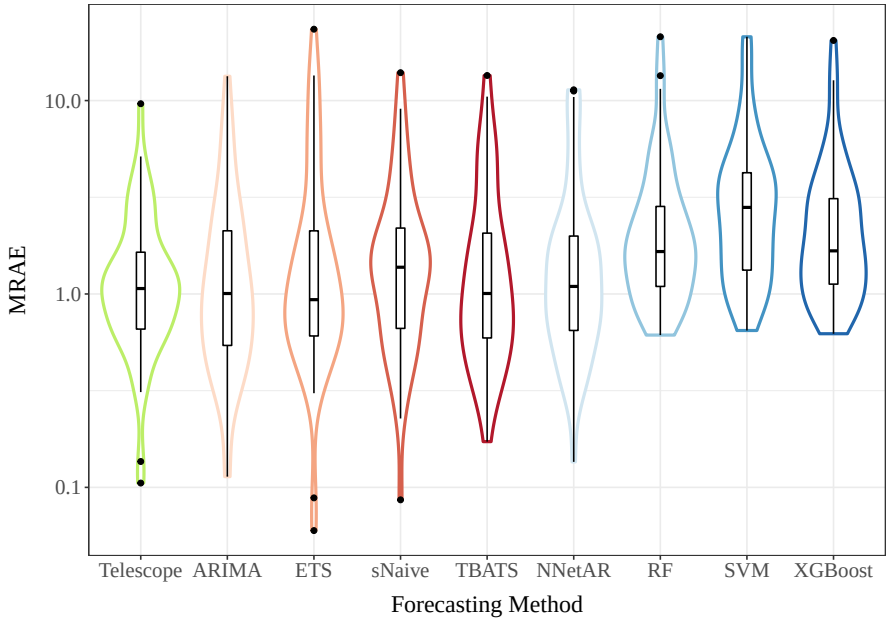


Figure 5.3: Violin plot of the achieved MRAE for all forecasting methods in competition.

the median. The violins of Telescope, ARIMA, TBATS, and NNetAR are similar in many aspects, such as the range of their peak and minimum values, their boxplots, and their medians. However, Telescope exhibits a much wider bulge around the median, which is also evident in the shorter interquartile range of the box plot. This indicates a more robust, high forecast accuracy of Telescope.

Similar results can be drawn from the analysis of the achieved MASE values presented in Figure 5.4. In particular, the bulge around the median in combination with the small interquartile range of Telescope's forecasts are also evident for MASE, while Random Forest, Support Vector Machine, and XGBoost cannot keep up with the other forecasting methods at all.

The poor forecast quality of Random Forest, Support Vector Machine, and XGBoost can also be observed in Figure 5.5 with respect to MAPE. However, ETS also shows shortcomings here, as the peak values are extraordinarily high. Telescope, ARIMA, sNaïve, TBATS, and NNetAR, by contrast, exhibit a distinct bulge around their median. Although the bulge of sNaïve appears to be the widest, its level is above the others. In contrast, the bulges of Telescope and TBATS are at similar levels, but Telescope provides the smallest minimum value as well as lower peak values than TBATS. The bulge of NNetAR is slightly lower compared to Telescope and TBATS, yet NNetAR also exhibits high outliers.

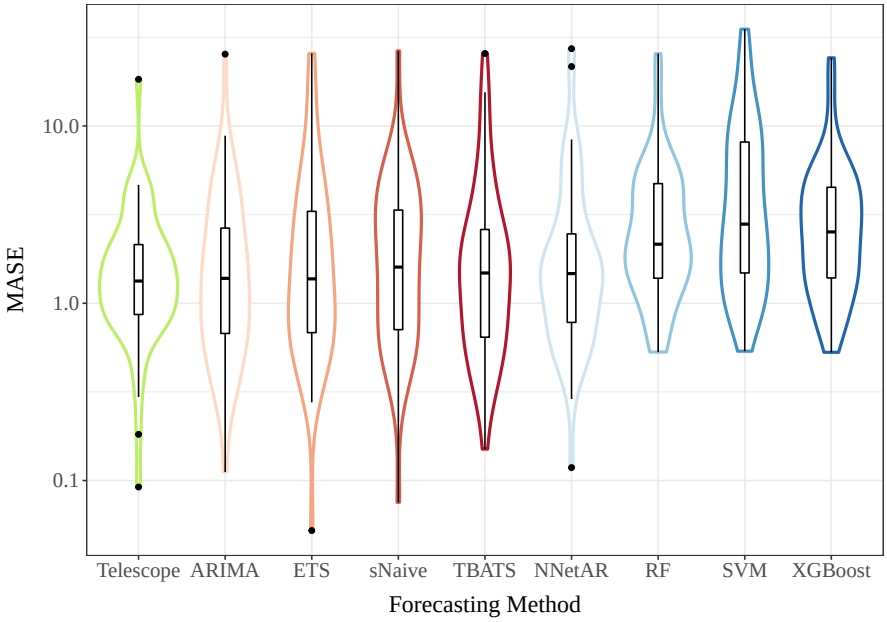


Figure 5.4: Violin plot of the achieved MASE for all forecasting methods in competition.

Finally, the bulge of ARIMA is not as pronounced as that of the other three methods, but is at a similar level as NNetAR, while showing less high outliers.

Figure 5.6 illustrates the violin plots for the time-to-result. Considering only the four best forecasting methods, namely Telescope, ARIMA, TBATS, and NNetAR, it can be observed that Telescope shows the least variation with a pronounced bulge around the box plot. In contrast, the violin plots of ARIMA and NNetAR rather resemble lines, revealing that their time-to-result scatters widely and, thus, is hardly reliable. Although the bulge is also visible for TBATS, the time-to-result is at a substantially higher level. Moreover, Telescope does not exhibit any outliers above 100 seconds, with a maximum time-to-result of 85 seconds. In contrast, ARIMA, TBATS, and NNetAR exceed this value significantly with a maximum time-to-result of 1025 seconds, 186 seconds, and 558 seconds, respectively.

To conclude, the violin plots as well as the average and variation in forecasting performance demonstrate Telescope's superiority in terms of both average and robustness of the achieved forecast accuracy and time-to-result. This makes Telescope suitable for applications with real-time requirements where conventional state-of-the-art forecasting methods, such as ARIMA and TBATS, should not be used due to their unreliable time-to-result.

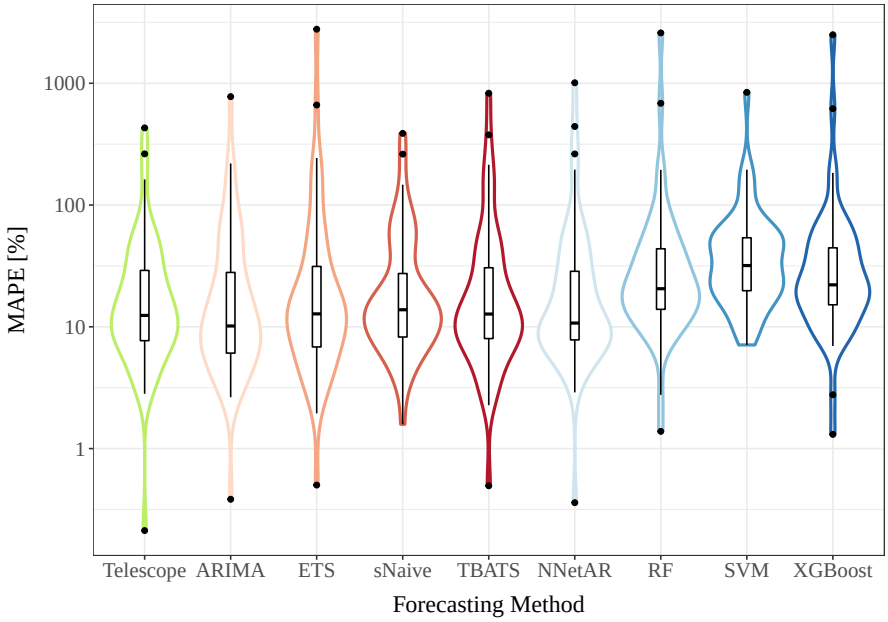


Figure 5.5: Violin plot of the achieved MAPE for all forecasting methods in competition.

5.2.3 Achieved Ranks per Forecasting Method

As described in Section 5.1, we also calculated the rank that each forecasting method achieved for each evaluation measure on each time series. By rank, we refer to the position of the forecasting method when ordering all nine forecasting methods by their delivered results from best to worst. Thus, a rank of one denotes best performance, while a rank of nine denotes worst result.

Table 5.6 reports the average ranks of the forecasting methods for each evaluation measure. The average ranks were derived as the arithmetic mean over all time series in the data set. Similar to the average and variation of forecast accuracy, Telescope achieved the smallest average rank for the forecast error measures MRAE and MAE. More specifically, when examining MRAE and MAE more closely, it can be seen that the average ranks of these measures are the same for each forecasting method, indicating that the order of best to worst forecasting method was always the same for both measures. Apart from ARIMA, which also achieved comparatively low average ranks for MRAE and MASE that are close to Telescope, the other seven forecasting methods were considerably inferior. In contrast, with respect to MAPE, ARIMA achieved an even smaller average rank than Telescope. However, when this result is combined

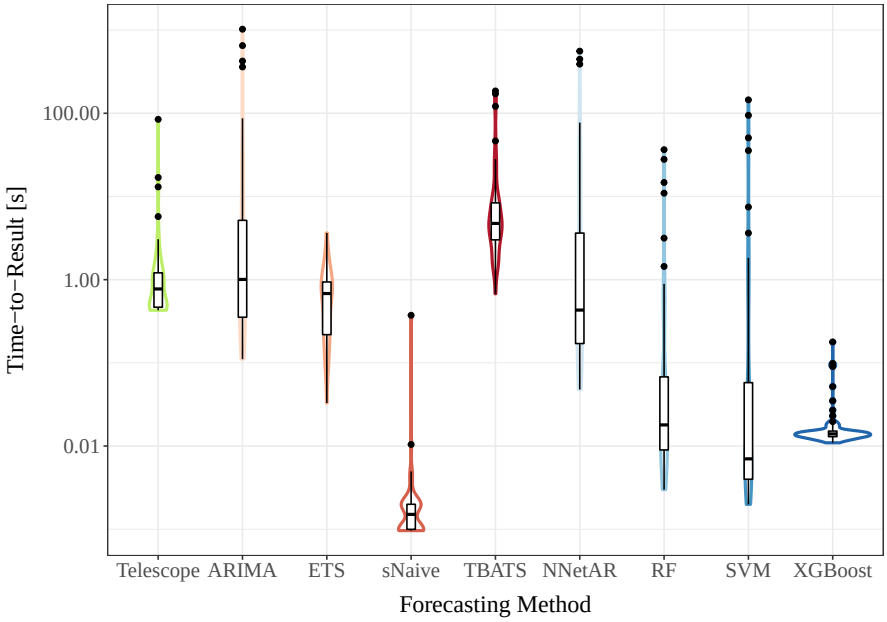


Figure 5.6: Violin plot of the time-to-result for all forecasting methods in competition.

with the findings obtained from Table 5.4, which shows the average forecasting performance, it can be concluded that ARIMA reached a smaller average rank with respect to MAPE, but if Telescope performed better, it must have performed substantially better than ARIMA. Otherwise, the average MAPE of Telescope would not have been so much smaller compared to ARIMA, while the average rank is higher. Considering the time-to-result, it is not surprising that sNaïve achieved by far the lowest rank of just above one. Yet, when comparing the four more sophisticated and powerful forecasting methods, Telescope, ARIMA, TBATS, and NNetAR, NNetAR reached the smallest average rank, closely followed by Telescope, while ARIMA and TBATS were clearly outperformed. With an average rank of 8.509, TBATS actually achieved almost the worst theoretical result of nine. Comparing Telescope with NNetAR regarding time-to-result, a similar conclusion can be drawn as for the comparison of Telescope with ARIMA regarding MAPE. Table 5.4 clearly shows that Telescope was on average much faster than NNetAR, although NNetAR achieved a smaller average rank. This amplifies the findings obtained from Table 5.5, which showed that the time-to-result of Telescope is much more robust compared to ARIMA, TBATS, and NNetAR. That is, Telescope does not exhibit as large outliers as the other methods, which can also be seen in Figure 5.6.

Table 5.6: The average yielded rank for the eight state-of-the-art forecasting methods and Telescope with respect to each evaluation measure. The best values for each evaluation measure are highlighted in bold.

Forecasting Method	MRAE [Rank]	MASE [Rank]	MAPE [Rank]	Time-to-Result [Rank]
Telescope	**3.509**	**3.509**	3.849	6.528
ARIMA	3.585	3.585	**3.415**	7.132
ETS	4.491	4.491	4.509	5.717
sNaïve	4.642	4.642	4.660	**1.019**
TBATS	4.057	4.057	4.113	8.509
NNetAR	4.358	4.358	4.264	6.415
Random Forest	6.557	6.557	6.547	3.642
SVM	7.283	7.283	7.019	3.094
XGBoost	6.519	6.519	6.623	2.943

5.3 Application of Telescope for Critical Event Prediction of Virtual Machine Scaling

A typical use case for critical event prediction using time series forecasting methods is the area of auto-scaling, where virtual machines are added or removed according to current and estimated future demand. In addition, common auto-scaling workloads show daily peaks and troughs due to the Internet usage behavior of humans. Hence, time series in auto-scaling usually exhibit seasonal patterns. Moreover, demand is measured with a high frequency, resulting in a large number of observations within a single period. To scale the number of virtual machines properly, not only the current demand is used, but also an estimate of the future demand. First, the future demand must be forecast as virtual machines require a certain amount of time to start up. Second, the virtual machines should be up and running just before the demand arrives. As a consequence, time series forecasting is the natural method of choice. Yet, time series forecasting methods in the field of auto-scaling must meet four important requirements: (I) The forecast accuracy should be as high as possible, (II) the forecast accuracy should be stable, especially for seasonal time series with many observations per period, (III) the forecasts are required within a fixed time window, as otherwise, the result cannot be used, and (IV) the forecasting method must be able to maintain the previous aspects while forecasting multi-step-ahead, since there are many observations within a period. Note that the

fourth requirement often involves several hundreds of values. To summarize, the runtime of the forecasting method should be as small and reliable as possible while keeping the accuracy of the forecasts high. In practice, however, most common forecasting methods, such as ARIMA and ETS, cannot provide a stable runtime or cannot handle time series with high frequencies very well. Given that auto-scaling deals with time series exhibiting a strong seasonal pattern and that TBATS was developed exactly for this type of time series, only TBATS and Telescope are compared for this use case. Furthermore, we apply auto-scaling to a multi-tier application. Multi-tier auto-scaling requires forecasting the demand of numerous tiers to properly scale the number of virtual machines at each tier simultaneously. The multi-tier auto-scaler employed in this use case is called Chameleon and was developed as a single-tier auto-scaler by A. Bauer [Bau16] and extended to a multi-tier auto-scaler by V. Lesch [Les17]. More specifically, this use case implements three tiers, namely presentation, business, and database. The service level objective (SLO) is that 95% of all requests have a response time of less than 2 seconds. The time series used as workload for this use case describes the number of requests sent to the social bookmark and publication sharing system BibSonomy[3].

Figure 5.7 illustrates the auto-scaling performance for all three tiers and the requests per second when applying TBATS as forecasting method. To this end, the top graph shows the presentation tier, the second graph depicts the business tier, and the third graph presents the database tier. For these top three diagrams, the black line represents the actual demand, while the red line illustrates the supply. Finally, the bottom graph shows the requests per second, where the black line represents the requests sent, the green line shows the number of requests that confirm the SLO, and the red line depicts the SLO violations. The top three graphs show that the supply does not match the demand very well. In particular, from minute 30 to 80 and from minute 250 to 310, it can be seen that the supply collapses while the demand remains high, indicating that the forecasts of TBATS either do not provide high accuracy or require too much runtime, so the results were discarded. This finding can also be derived by taking a closer look at the bottom plot, because the SLO violations increase considerably for the time periods mentioned. More precisely, almost all requests in these periods violate the SLO. Besides these two long periods, most requests are served within less than 2 seconds, thus confirming the SLO.

The auto-scaling performance for all three tiers and the requests per second when applying Telescope instead of TBATS is shown in Figure 5.8. The structure of Figure 5.8 is the same as in Figure 5.7. In contrast to Figure 5.7, the top three

[3]BibSonomy: `https://www.bibsonomy.org/`

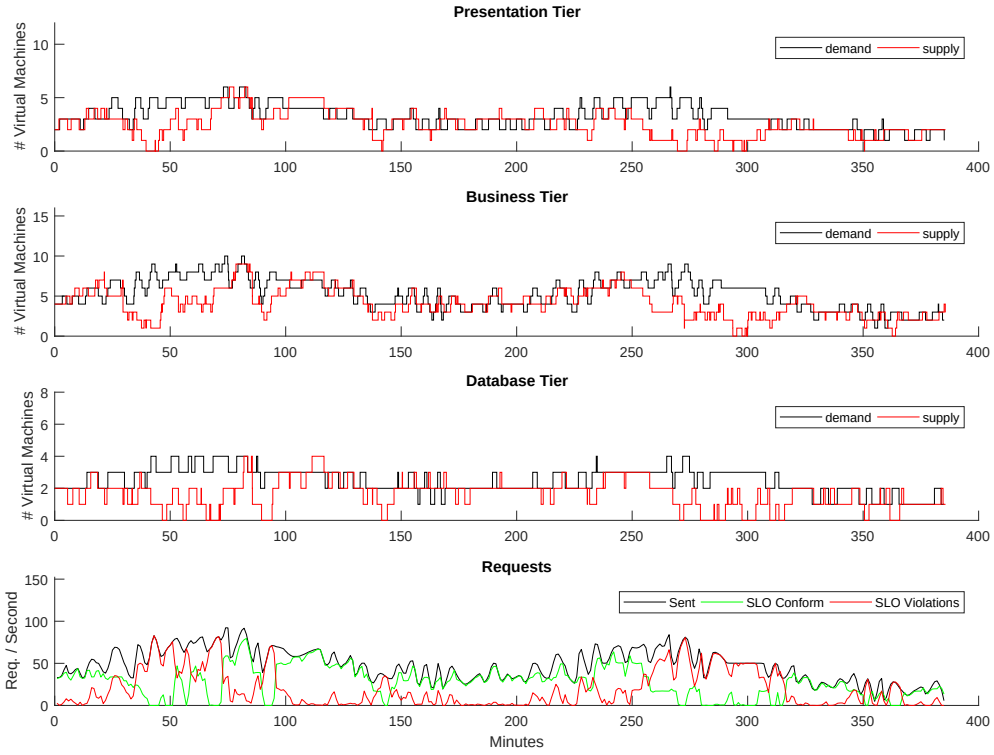


Figure 5.7: Demand, supply, and requests evaluation of Chameleon for the BibSonomy trace on multi-tier application with TBATS.

plots of Figure 5.8 reveal that the supply matches the demand very well. In most cases, the number of virtual machines is scaled just before a change in demand occurs, demonstrating the benefit of critical event prediction by means of forecasting rather than reactive auto-scaling. The bottom diagram implies the same conclusion, as there are hardly any requests violating the SLO. Only for a few short periods of time (e.g., around minutes 90 and 160), a small number of requests exceed the SLO. In general, however, the green line is almost equal to the black line, so most requests confirm the SLO. Compared to TBATS, it can be seen that Telescope considerably improves the auto-scaling performance.

To compare the auto-scaling performance achieved by TBATS and Telescope as forecasting methods numerically, we used three measures and report the values achieved by both forecasting methods in Table 5.7. The first measure is the average response time of the scaled service in milliseconds. For the calculation of this measure, the response time of requests that do not confirm

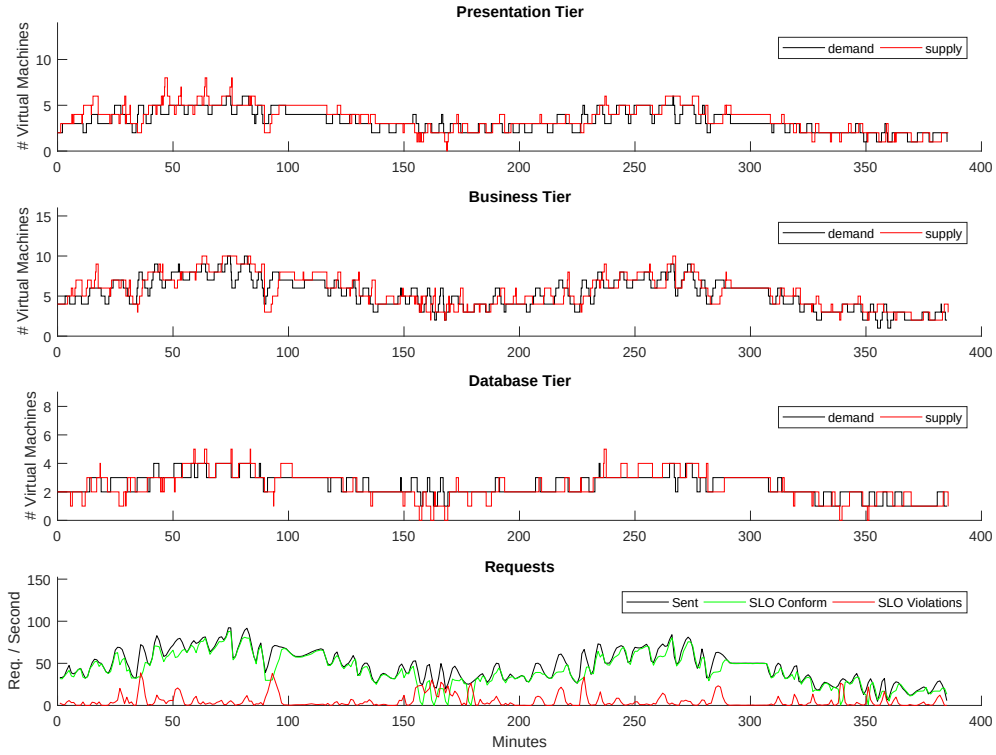


Figure 5.8: Demand, supply, and requests evaluation of Chameleon for the BibSonomy trace on multi-tier application with Telescope.

the SLO of 2 seconds is set to this value. Thus, a maximum response time of 2000 milliseconds cannot be exceeded. The second measure of auto-scaling performance is the percentage of requests that do not meet the SLO. The last evaluation measure is called distance, as it describes the deviation of the auto-scaler from the optimal auto-scaler, i.e., the auto-scaler that perfectly matches the demand. For all three measures, a lower value indicates better auto-scaling performance. Regarding the average response time, Telescope achieved a much

Table 5.7: Results of the multi-tier auto-scaling evaluation of Telescope and TBATS.

Forecasting Method	Response Time	SLO Violations	Distance
Telescope	1071.08 ms	9%	0.57
TBATS	1339.94 ms	36%	0.68

lower value of about 1071 milliseconds, while the average response time of TBATS was about 1340 milliseconds. This increase of about 25% in the average response time when using TBATS is caused by the large percentage of SLO violations. When using Telescope, only 9% of all requests exceeded the SLO, while 36% of all requests violated the SLO when using TBATS. Finally, the distance measure confirms that auto-scaling with the use of Telescope is closer to the optimal auto-scaler than auto-scaling with TBATS. Considering all these evaluation results, Telescope significantly improves the critical event prediction of virtual machine auto-scaling compared to the state-of-the-art method TBATS.

5.4 Concluding Remarks

To summarize the main contribution of this chapter, we answer research question **RQ A.2**, which focuses on the comparison of state-of-the-art forecasting methods with our proposed hybrid, component-based forecasting method Telescope. The results of our experiments show the superiority of Telescope compared with eight state-of-the-art forecasting methods. Compared with the state-of-the-art method that yielded the second best forecast accuracy, namely ARIMA, Telescope achieved on average 21%, 22%, and 25% less forecast errors with respect to MRAE, MASE, and MAPE, respectively. A second criterion was the time-to-result, since many applications require forecasts within fixed time windows. Here, we have shown that Telescope achieved average speedups of approximately 1578%, 402%, and 953% compared to the three most accurate competitors ARIMA, TBATS, and NNetAR, respectively. Apart from this improvement in average forecast accuracy and time-to-result, we also discussed Telescope's enhanced robustness. To this end, we reported the standard deviations of forecast accuracy as well as time-to-result and provided violin plots of the achieved forecast accuracy as well as time-to-result. Both investigations revealed the superior robustness of Telescope compared with the other forecasting methods. Finally, we applied Telescope to the use case of critical event prediction of virtual machine auto-scaling and compared the achieved auto-scaling performance with that obtained when using TBATS instead of Telescope. Here, we demonstrated that the average response time of requests was reduced by about 20% by using Telescope instead of TBATS. Furthermore, Telescope resulted in only 9% SLO violations, while TBATS produced 36% SLO violations.

Although the results are very promising, one assumption must be mentioned. The experiments included only seasonal time series, since Telescope cannot handle non-seasonal time series. Thus, for applications where the seasonality of time series cannot be ensured, a fallback method must be established.

Chapter 6

Meta-Learning for Time Series Forecasting Method Recommendation

In order to react to changes in the system or the environment through adaptation, Self-Aware Computing Systems integrate an analysis mechanism to monitor data on system resources and the environment. Traditional approaches make use of models or thresholds to track the current state of the system. However, since this is a status quo analysis of the system, it can only be used to perform reactive system adaptations. Due to the increasing ubiquity of computing power, for instance through the incorporation of cloud resources, the integration of proactive adaptation by analyzing predicted system states is becoming a viable alternative [KRV+15, Wey17]. Using proactive rather than reactive adaptations prevents delays in the adaptation process by allowing the system to anticipate adaptation requirements in advance.

Regarding proactive adaptation, systems can incorporate forecasting methods to predict impending critical events that will lead to changes in the system state. However, based on the "No Free Lunch Theorem" [WM97], there is no forecasting method that is best suited for all scenarios. Hence, the choice of forecasting method depends on the time series under consideration. Typically, selecting the best forecasting method relies on expert knowledge, which makes it costly, potentially subjectively biased, and often requires a significant amount of time to produce results. In addition, Self-Aware Computing Systems must deal with complex types of uncertainties [Wey17], making it impossible to a priori model all situations the system might encounter at runtime. Moreover, this affects the selection of the appropriate forecasting method, since the characteristics of the time series data might be unknown at design time. Therefore, integrating expert knowledge at design time is not suitable for Self-Aware Computing Systems. Consequently, the automatic selection of forecasting methods at runtime must handle unforeseen situations and context changes.

In an effort to automate the process of using forecasting methods while avoiding the need to rely on expert knowledge, several hybrid approaches have

been developed in the literature. As described in Section 3.1.1, one class of approaches is ensemble forecasting, which applies multiple forecasting methods to the same time series and returns a weighted average of their forecasts as final result [BG69, Cle89, NG74, DMBT00, KKZ+00, SSH16]. Another class of approaches processes time series component-wise, by applying different forecasting methods to the individual components [LTZ+14, XCCP06, CPEM05, SD10]. Then, the results of the component forecasts are aggregated to derive a forecast for the overall time series. We have also contributed to this area of hybrid, component-based time series forecasting techniques with the introduction of Telescope. More information on the design and an evaluation of Telescope is presented in the Chapters 4 and 5, respectively. Finally, another class of hybrid forecasting approaches is based on forecasting method recommendation [CA92, AKA97, WSMH09, LG10, NAL17]. Thus, a set of rules is derived to select an appropriate forecasting method based on certain characteristics of the time series under consideration.

In this chapter, we present two novel approaches for meta-learning the dependencies between time series and the performance of forecasting methods on them. The first meta-learning approach is based on a database of numerous diverse time series, on which several time series characteristics are computed and the forecast accuracy of all potential forecasting methods is evaluated. Using this information, a machine learning model is trained to learn the relationship between the time series characteristics and the performance of each forecasting method. The second approach to time series forecasting method recommendation is based on the assumption that newly arriving data of a time series do not differ much from the last known observations. That is, the forecast accuracy of the potential forecasting methods is computed using the last known observations as the out-of-sample horizon, while the previous observations are used to train the forecasting methods. Therefore, the first approach requires a large database with many particularly diverse time series. The second approach, by contrast, does not require such a database, but it demands comparatively long time series, so that a part of these can be used for validation.

The remainder of this chapter is organized as follows: Section 6.1 introduces the meta-learning approach to learning the relationship between time series characteristics and forecasting method performance based on a large and diverse database. The content of this section is based on our previous work published as a full paper as part of the 16th IEEE International Conference on Autonomic Computing (ICAC) [ZBL+19]. A. Bauer *et al.* adopted the approach to create a forecasting method recommendation system specifically for seasonal time series only. This work has been published as a short pa-

per as part of the 11th ACM/SPEC International Conference on Performance Engineering [BZG+20]. In Section 6.2, we present the second approach to recommending time series forecasting methods, which considers only the historical observations of the respective time series. The content of this section is based on another previous work published as a short paper at the 15th International Symposium on Advanced Artificial Intelligence in Applications (AAIA) as part of the 15th Conference on Computer Science and Intelligence Systems (FedCSIS) [ZK20]. Finally, Section 6.3 summarizes the chapter and answers the corresponding research question. For evaluation results on both time series forecasting method recommendation systems, we refer to Chapter 7.

6.1 Data-based Time Series Forecasting Method Recommendation

The most popular approach to time series forecasting method recommendation is by X. Wang *et al.* [WSMH09] and postulates universally applicable rules for the selection of forecasting methods. Due to its detailed introduction to the subject of forecasting method recommendation, the approach is quite popular. Curiously, although the approach is highly cited, the validity of these rules has never been comprehensively evaluated. To fill this gap, we thoroughly evaluate the rules proposed by X. Wang *et al.* (cf. Chapter 7) and present a novel approach to dynamically (re-)learn recommendation rules based on a growing collection of data sets. Thus, we investigate the following aspects:

- How do the rules proposed by X. Wang *et al.* perform on the original training data set from their publication?
- Can the recommendation rules by X. Wang *et al.* be directly applied to other data sets?
- How can we improve the quality of time series forecasting method recommendations?

To this end, we present the general basics on rule learning for time series forecasting method recommendation in Section 6.1.1, followed by more details on the approach of X. Wang *et al.* in Section 6.1.2. We then propose two novel approaches to recommending forecasting methods. The first approach (cf. Section 6.1.3) generates rules by applying binary classification with oversampling, while the second approach combines forecasting method recommendation and weighted ensemble forecasting (cf. Section 6.1.4).

6.1.1 Basics on Rule Learning for Forecasting Method Recommendation

In general, the common approach to recommending forecasting methods is to learn rules that map time series characteristics to method recommendations. For this purpose, numerous time series characteristics need to be determined to cover the most relevant time series attributes. Subsequently, these features are used to learn dependencies between the performance of forecasting methods and the time series characteristics themselves.

X. Wang *et al.* [WSMH09] adapted an architecture for meta-learning from Vilalta *et al.* called *knowledge acquisition mode* [VGCBS04], which was originally intended for data mining tasks. Figure 6.1 illustrates the basic concept of this approach. The essential part of this concept is the database of time series examples, since these are used for rule generation. If the time series in the database are not representative enough or do not cover a broad range of time series properties, the generated rules may not generalize well at runtime. However, once the time series database is sufficiently large and covers all relevant time series properties, the tripartite rule generation approach can be applied.

The left side of Figure 6.1 presents the forecasting component. Here, all time series in the example database are forecast using the base-level methods, i.e., the potential forecasting methods implemented in the recommendation framework. For this purpose, all time series are split into a history, which is used to train the forecasting model, and a validation set. The forecast values are then compared to the original values in the validation set. Subsequently, the forecast results are stored in the prediction results database. On the right side, the computation of the time series characteristics is shown. During this step, the characteristics (for instance those presented in Section 2.2.1) are determined, normalized, and stored in the meta-level attributes database. The third part combines the database of prediction results and the database of meta-level attributes into a meta-level data set that maps time series characteristics directly to forecast accuracy. Based on these data, rule generation algorithms derive application rules and store them in the knowledge base. As this approach is quite intuitive and has proven successful in the data mining domain, we also adopt this architecture for our forecasting method recommendation approach.

6.1.2 General Approach of X. Wang et al.

This section is intended to provide a brief overview of the approach by X. Wang *et al.* to better understand its shortcomings and serves as a foundation for our approach presented in Section 6.1.3. For more details on the approach by X. Wang *et al.*, refer to their original paper [WSMH09].

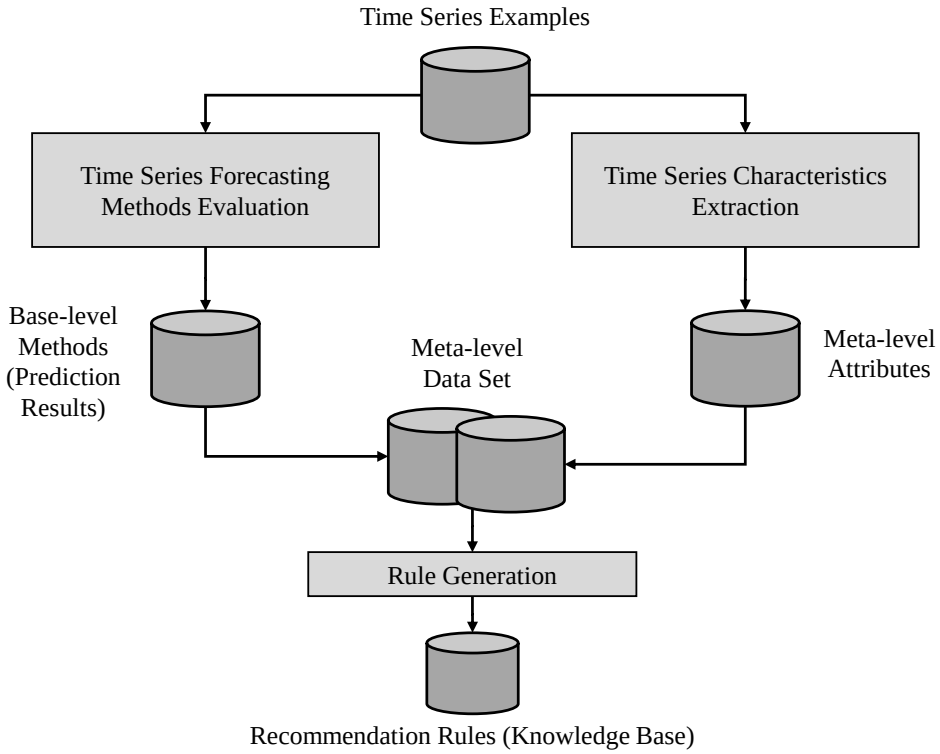


Figure 6.1: Knowledge acquisition mode according to Vilalta *et al.* [VGCBS04]. X. Wang *et al.* applied an adapted version in [WSMH09].

X. Wang *et al.* proposed clustering and rule induction algorithms to generate categorical and quantitative rules based on a variety of time series characteristics [WSMH09]. Therefore, the authors presented nine time series characteristics that are assumed to have a correlation with the performance of the four forecasting methods ARIMA, ETS, NNetAR, and Random Walk. The nine time series characteristics used by X. Wang *et al.* were trend, seasonality, periodicity, skewness, kurtosis, serial correlation, non-linearity, self-similarity, and chaos. In addition, X. Wang *et al.* also calculated the characteristics serial correlation, non-linearity, skewness, and kurtosis for the de-trended and de-seasonalized time series. Consequently, 13 features were determined for each time series.

Subsequently, hierarchical clustering and Self-Organizing Maps were presented in the field of time series forecasting. That is, X. Wang *et al.* described how to use such clustering methods to group similar time series together and, consequently, generate judgmental and conceptual rules. Moreover, they ap-

plied a decision tree technique, namely the C4.5 algorithm, to automatically generate quantitative rules. Therefore, the forecasting methods were ranked for each time series according to their achieved accuracy. Given that the recommendation system only tried to predict which forecast method to choose, X. Wang *et al.* set the class label to `1` for the best forecasting method for each time series. All other forecasting methods were given class label `0` for the respective time series. The resulting class labels were used as prediction targets, while the time series characteristics were used as meta-level features. Based on these data, the C4.5 algorithm was applied to each forecasting method to generate quantitative recommendation rules. This resulted in four sets of rules, one for each forecasting method. Finally, the derived rules indicated whether or not the particular forecasting method should be used.

To reconstruct the results, X. Wang *et al.* provided all necessary parameter settings for the application of the C4.5 algorithm and presented the generated rules. Unfortunately, however, neither the conceptual nor the quantitative rules were evaluated. Furthermore, the data set used to learn the models was not partitioned, so there was no distinction between training and validation data.

6.1.3 Binary Classification with Oversampling

To improve the forecast accuracy achieved by recommending forecasting methods, we developed a novel approach on the basis of the approach of X. Wang *et al.* Our rule learning approach, which is schematically presented in Figure 6.2, directly addresses the limitations of the approach of X. Wang *et al.*:

1. The single decision tree created by the C4.5 algorithm tends to overfit to the training data.

2. The training data are highly imbalanced.

3. It is possible that no forecasting method is recommended by the rules.

Although any forecasting method can be incorporated into this approach, we only include the four forecasting methods also used by X. Wang *et al.*, i.e., ARIMA, ETS, NNetAR, and Random Walk, to maintain comparability. In the following, this section describes the approach in more detail.

The first step is to compute the time series characteristics for all time series in the training set. Here, we compute the same characteristics as X. Wang *et al.* and also apply the same normalization to these characteristics. The normalization is necessary because the different time series characteristics vary greatly in range. Since this could disturb the learning process of decision tree algorithms,

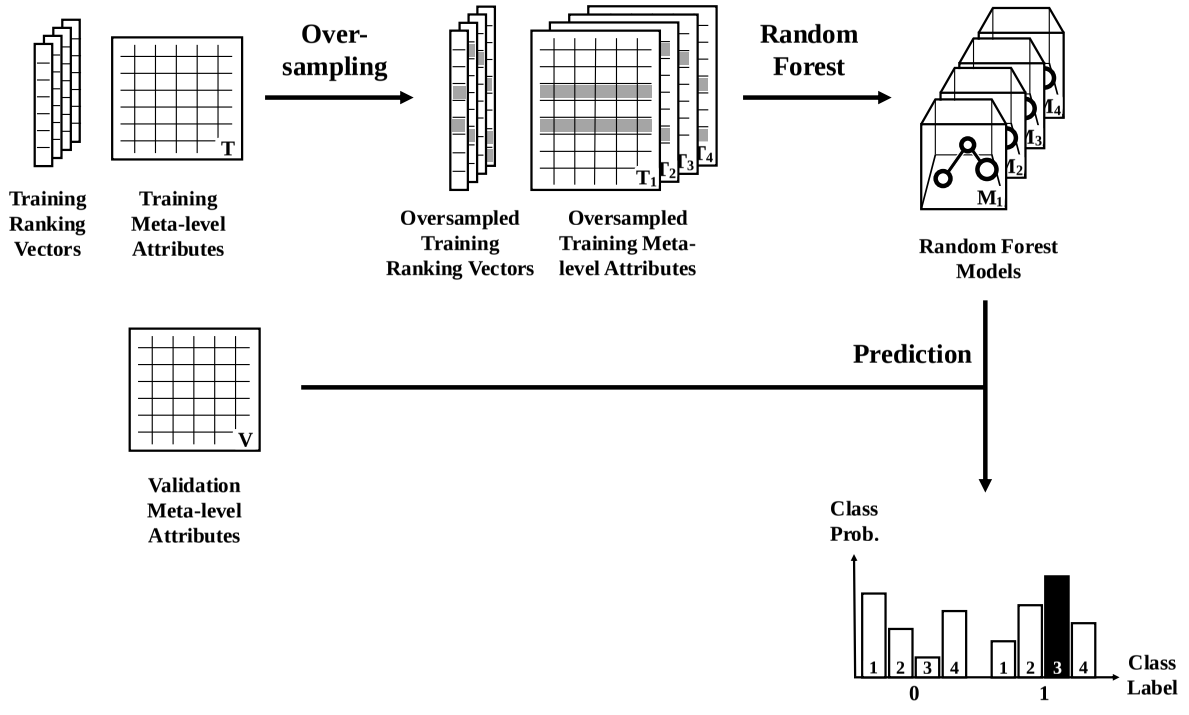


Figure 6.2: Schematic illustration of the rule generation approach. First, oversampling is applied to the training data, i.e., the class label vectors and the matrix of characteristics T. The oversampled class label vectors and matrices of characteristics T_1 to T_4 are then used to learn a binary classification Random Forest model M_1 to M_4 for each forecasting method in the recommendation system. The characteristics of the validation time series V are passed to these models, resulting in probabilities for the class labels `0` and `1` for each forecasting method. Finally, the forecasting method with the highest probability of class label `1` is returned.

the ranges of the different time series characteristics must be unified. Once the characteristics are computed and normalized, they are stored in a matrix of characteristics. Here, each row represents the time series characteristics for one time series. Figure 6.2 represents this feature matrix, i.e., the so-called meta-level attributes, for the training set as T on the left-hand side.

Second, to assess the forecast accuracy and determine the best forecasting method, the first 80% of time series observations are used to train the four forecasting methods. The remaining last 20% of time series observations are defined as the forecast horizon, which is used to compute the out-of-sample accuracy. Once all four forecasting methods are trained, forecasts are provided, and the accuracy is computed, the approach determines for each time series which forecasting method provides the most accurate forecast. The result of this step is a vector of zeros and ones for each forecasting method. Here, each entry in a vector represents a time series. If the particular forecasting method performs best for the time series under consideration, the value of that particular cell in the vector is set to `1`, and to `0` otherwise. Note that exactly one forecasting method can receive the label `1` for a given time series, while all other forecasting methods receive label `0`. These vectors, the so-called ranking vectors, can be seen in Figure 6.2 on the far left.

However, since there are four forecasting methods in competition, each forecasting method can be expected to receive more zeros as class labels than ones. This leads to a highly imbalanced training set and, therefore, degrades the quality of the generated rules. Several approaches have been developed to deal with such imbalanced data. One option is to adjust the performance measures (e.g., precision, recall, and F1-score) to correctly represent the imbalanced data set. Another approach is to utilize oversampling or undersampling to balance the number of instances for the minority and majority classes. Using oversampling, new instances of the minority class are created either by simply duplicating existing instances or by artificially creating new instances based on existing instances. In contrast, undersampling discards instances of the majority class until the number of instances for the minority and majority classes are equalized. In the approach presented in this section, we apply oversampling to avoid shrinking our training base. More specifically, we use the duplication approach for oversampling, as it provided better performance for this scenario in a preliminary study than the approach that synthetically generates new instances. However, since the entire training set has to be oversampled, not only the entries of the ranking vectors are duplicated, but also the corresponding rows of the matrix of meta-level attributes. Figure 6.2 shows this oversampling step and depicts the oversampled meta-level attribute matrices as T_1 to T_4

along with the respective ranking vectors. The oversampled entries of the vectors and matrices are highlighted in gray. Note that the training sets of the four forecasting methods may have different numbers of instances after the oversampling procedure.

In the next step, a rule-learning algorithm derives rules using the oversampled meta-level matrix as features and the oversampled ranking vectors as labels. For this task, X. Wang *et al.* used the C4.5 algorithm, which constructs a decision tree based on the time series characteristics of the training set. However, such a single decision tree might not cover all relevant aspects for decision-making and tends to overfit to the training data. To address this shortcoming, we employ Random Forest, a decision tree-based ensemble learning method, to dynamically learn the rule set. Unlike the single decision tree of the C4.5 algorithm, Random Forest constructs multiple decision trees by considering only a randomly sampled subset of characteristics at each candidate split of the decision tree and returns the majority of their predictions. For more information on Random Forest, refer to Section 2.4.4. Although Random Forest can be used for regression and classification, we perform binary classification with labels `0` and `1`, where `0` denotes that the particular forecasting method is not recommended and `1` indicates that the Random Forest model recommends the use of that particular forecasting method for the time series under consideration. For this purpose, we set the Random Forest parameters to the settings shown in Table 6.1[1]. Finally, this step yields four Random Forest models, M_1 through M_4, one for each forecasting method (cf. Figure 6.2).

Table 6.1: Parameter settings of Random Forest for binary classification.

Parameter	Description	Setting
`ntree`	Number of trees generated	500
`mtry`	Number of randomly selected features as candidates at each split	2
`classwt`	Priorities of the classes	false

In the last step, the best performing forecasting method for a particular time series is predicted. For this purpose, the same time series characteristics as for the training time series are computed for all time series in the validation set. This matrix of meta-level attributes of the validation part V is then passed

[1]For more information on the parameters of Random Forest, we refer to its documentation: `https://cran.r-project.org/web/packages/randomForest/index.html`.

to the Random Forest models. Note that this validation set consists of only new time series that were not known during training. As each Random Forest model predicts whether the particular forecasting method is appropriate, each model provides two class probabilities per time series. The probability of class `1` indicates how likely that forecasting method is to be suitable for the particular time series. Note that the class probabilities for a forecasting method add up to one for each time series. In contrast, when summing the probabilities for a particular class, i.e., `0` or `1`, across all forecasting methods, the result does not necessarily equal one because the class probabilities are derived from independent models. Once all Random Forest models provide the class probabilities, the forecasting method with the highest probability for class `1` is recommended to ensure that a forecasting method is recommended in any case.

To summarize our approach and highlight its advantages over the approach of X. Wang *et al.*, we optimized the following aspects:

1. Due to the high imbalance in the class labels of the training data, which in turn degrades the quality of the generated rules, we employ oversampling on the ranking vectors as well as on the meta-level attribute matrices.

2. The rule learning algorithm applied by X. Wang *et al.*, namely the C4.5 algorithm, constructs only a single decision tree, which tends to overfit to the training data. In contrast, we apply Random Forest, an ensemble learner based on constructing multiple trees using bagging and, consequently, prevents overfitting.

3. The rules generated by X. Wang *et al.* recommend for each forecasting method whether or not to apply the forecasting method. However, in many cases, the recommendation system does not suggest the application for any of the forecasting methods. Although this might be an interesting insight into the data, it does not help autonomous systems in decision-making. Therefore, our approach always recommends the forecasting method with the highest probability of being suitable for the time series under consideration.

6.1.4 Recommendation-based Ensemble Forecasting

The main issue of typical forecasting method recommendations is that by selecting only one forecasting method for each time series, the forecast accuracy over a set of time series typically exhibits large variance. To remedy this drawback, we additionally propose a combination of ensemble forecasting and forecasting method recommendation. On the one hand, we employ linear regression to

automatically adjust the weights for a linear combination of forecasts from different forecasting methods. On the other hand, we utilize an activation function that filters out forecasting methods that are assumed to perform poorly on the time series under consideration.

To this end, we derive a linear combination of each forecasting method Y^i (ARIMA, ETS, NNetAR, and Random Walk). Each weight ϑ^i is estimated by a linear regression of the meta-level attributes of the previously examined time series. Having only zero and one as input, the regression yields values within this range. In the following, we interpret this output of the linear regression as the degree to which the forecasting method is suitable for the particular time series. However, in contrast to common weights in ensemble forecasting, these weights must be activated. More precisely, if a weight satisfies the activation function Γ, the corresponding forecasting method is used in the ensemble, otherwise its weight is set to zero and, thus, the result of the forecasting method is discarded. In addition, the linear combination is normalized by ϑ, which is the sum of the activated weights. In terms of activation, different functions can be used. The activation function used in this thesis is two-fold, which means that a weight is activated only if it meets both criteria. The weight ϑ^i must be (I) greater than or equal to the mean of the weights $\overline{\vartheta}$ and (II) greater than or equal to the share α of the maximum weight $\hat{\vartheta}$. The first condition has the advantage that by using the mean value of all weights, the presumed most appropriate forecasting method is always included, while the presumed least appropriate method is always omitted. However, in the unlikely case that all weights have the same value, all forecasting methods are considered. As the mean is biased toward outliers, the second condition allows having a higher threshold than the mean if the outlier is close to zero. Finally, the steps of the recommendation-based ensemble forecasting approach can be formally expressed as follows, where Y denotes the final forecast:

$$Y = \frac{1}{\vartheta} \sum_{i \in \{A,E,N,R\}} Y^i \cdot \Gamma(\vartheta^i) \quad \text{with} \tag{6.1}$$

$$\vartheta = \sum_{i \in \{A,E,N,R\}} \Gamma(\vartheta^i) \ , \tag{6.2}$$

$$\Gamma(\vartheta^i) = \vartheta^i \cdot \Big(1 - \max(\text{sign}(\max(\overline{\vartheta}, \alpha \cdot \hat{\vartheta}) - \vartheta^i), 0)\Big) \, , \tag{6.3}$$

$$\overline{\vartheta} = \underset{i \in \{A,E,N,R\}}{\text{mean}}(\vartheta^i) \, , \quad \text{and} \tag{6.4}$$

$$\hat{\vartheta} = \max_{i \in \{A,E,N,R\}}(\vartheta^i) \tag{6.5}$$

Both the binary classification with oversampling approach as well as the recommendation-based ensemble forecasting approach can be applied to runtime performance decisions in Self-Aware Computing Systems. In an offline phase, the initial recommendation rules, respectively the linear regression models, must be learned. During runtime, these rules and weights can be used for new and unseen time series, since the application of the recommendation can be done in real-time. Subsequently, the new time series can be used to dynamically re-learn the recommendation rules and weights. As the binary classification approach with oversampling is based on Random Forest, which is typically very fast in both learning and prediction[2], the dynamic re-learning of rules is feasible with only low overhead at runtime.

The evaluation of the proposed approaches along with the comparison to the approach by X. Wang *et al.* are presented in Section 7.1.

6.2 History-based Time Series Forecasting Method Recommendation

There are already several existing approaches to recommending forecasting methods in the literature. While early approaches use manually created expert systems, such as F. Collopy and J. Armstrong [CA92], more recent approaches to recommending forecasting methods derive rules automatically, such as X. Wang *et al.* [WSMH09] and our approach presented in Section 6.1. However, these automatic rule-learning approaches compute time series characteristics of a large and diverse training database and evaluate the forecast accuracy of the available methods using these meta-level attributes. Then, a rule-learning technique is applied to map the time series characteristics to the best forecasting method. Therefore, these approaches are highly dependent on the size and diversity of the training database. However, such databases are not always available, which is the reason why we present a different approach that does not require such a training database. Instead, the performance of the different forecasting methods is estimated under the assumption that the performance of the forecasting methods on the last known part of the time series to be forecast, which we refer to as the in-sample validation part, is representative for the out-of-sample forecast. This also does not require a rule-learning approach, since our approach selects the forecasting method with the highest R^2-score on the in-sample validation part of the time series under consideration. Figure 6.3 presents the simplified overall workflow of our history-based forecasting

[2] On our data sets, rule generation was done within minutes and the recommendation for a single time series was completed within milliseconds.

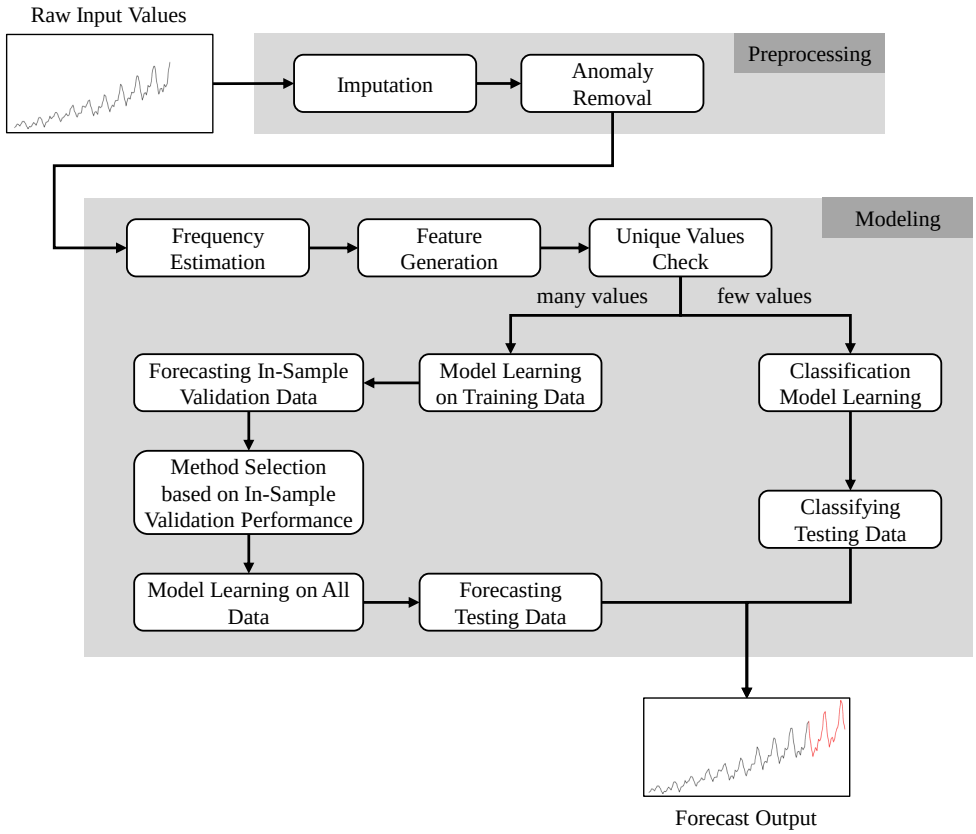


Figure 6.3: The overall workflow of the history-based forecasting method recommendation approach including preprocessing and modeling.

method recommendation approach, which consists of two main parts, namely time series preprocessing (cf. Section 6.2.1) and modeling (cf. Section 6.2.2).

6.2.1 Preprocessing

In most real-world applications, the acquisition of time series is an error-prone process due to recording errors or transmission faults. Therefore, time series often contain missing values or anomalous entries. For this reason, the workflow preprocesses the time series by imputing missing data and removing anomalous values as the first step.

6.2.1.1 Missing Data Imputation

Missing data imputation is the first preprocessing step, as shown in the topmost gray box of Figure 6.3. Such missing values of a time series can be located either at the beginning of the time series or somewhere within the time series. In this contribution, it is assumed that in case of missing values within the time series, only a few consecutive data points are missing. For missing values at the beginning of a time series, we do not reconstruct the missing values because these gaps can be arbitrarily long and reconstructing long time series is typically highly error-prone and would therefore tend to worsen the forecasting model. Furthermore, missing data at the beginning of a time series are not too critical, as they merely shorten the time series.

Instead, the algorithm for imputing missing values within the time series is shown in Algorithm 13. First, a daily pattern is assumed. This is a common assumption for human-influenced time series, since the human behavior depends on the time of day. Thus, the algorithm requires the time series observations and the number of observations per day as input. The algorithm starts by identifying the indices of missing values (cf. Line 2). Next, it loops through all missing values and imputes the missing value based on the daily pattern along with an estimated trend. For this purpose, the algorithm analyzes whether there is a trend between the day of the missing data and the next or previous day. If the missing value occurs before the end of the first season (cf. Line 4), the algorithm computes the trend as the ratio between the value one season after the first available measurement and the first available measurement itself (cf. Line 8). However, if the value one season ahead is also missing, the trend is set to non-existent (i.e., the trend factor equals 1) (cf. Line 5). Subsequently, the algorithm imputes the missing value as the value one season later divided by the trend factor. In contrast, if the index of the missing value is after one season, but at maximum at the length of two seasons (cf. Line 11), the algorithm computes the trend as the ratio of the value before the missing value to the value one season before (cf. Line 15). Thus, the algorithm always computes the ratio of two values with a distance of one daily pattern. Finally, if the index of the missing value is greater than the length of two periods plus one, the algorithm computes the ratio of the entire daily season before the missing value to the corresponding previous values one daily season apart (cf. Lines 21-23). The final trend factor is determined as the median of this set of trend factors (cf. Line 24). In the latter two scenarios, the missing value is computed by multiplying the known value one season prior to the missing value by the computed trend factor (cf. Lines 17 and 25). We apply this procedure in chronological order so that imputed values can be used to impute subsequent gaps. If there are still

Algorithm 13 Impute missing values within a time series

Input: observation values x, observations per day n_day
Output: imputed time series ts

```
1: ts = x
2: indices = getNaNs(ts)
3: for all (index in indices) do
4:   if (index < n_day) then
5:     if (ts[getNextValue(ts, index) + n_day] in indices) then
6:       trend = 1
7:     else
8:       trend = ts[getNextValue(ts, index) + n_day] / ts[getNextValue(ts,
         index)]
9:     end if
10:    ts[index]= ts[index + n_day] / trend
11:  else if (index > n_day & index ≤ (2 × n_day)) then
12:    if (ts[getPreviousValue(ts, index) − n_day] in indices) then
13:      trend = 1
14:    else
15:      trend = ts[getPreviousValue(ts, index) − n_day] /
         ts[getPreviousValue(ts, index)]
16:    end if
17:    ts[index]= ts[index − n_day] × trend
18:  else
19:    trends = createEmptyList()
20:    prev_value = getPreviousValue(ts, index)
21:    for all (index in [(prev_value − n_day):prev_value]) do
22:      trends = append(trends, ts[index] / ts[index − n_day])
23:    end for
24:    trend = median(trends, ignoreNaNs = TRUE)
25:    ts[index]= ts[index − n_day] × trend
26:  end if
27: end for
28: return ts
```

few missing values after applying this algorithm (i.e., the values one season before or after the missing value are also missing), the value is imputed by linear interpolation between the last known value before the missing value and the first known value after the missing value.

6.2.1.2 Anomaly Removal

The second step of the preprocessing pipeline is the removal of anomalies, as depicted in the topmost gray box of Figure 6.3. Removing anomalies from time series before model learning is an essential task, as such outliers can severely degrade the quality of forecasting models. Therefore, we apply a method to identify and remove anomalies. For this approach, the workflow implements a modified version of the well-known *three-sigma rule*. However, unlike the typical three-sigma rule, the workflow employs median instead of mean as baseline, since the distributions of the time series are not necessarily symmetric. In addition, the standard deviation is only computed between the 1st and 99th percentile of the data, since possible outliers would already affect the standard deviation if it was computed over the entire time series. Also, we set the tolerance multiplier to a more conservative value (i.e., 10-sigma rule), as we do not want to remove normal peaks in a daily seasonal pattern. After detecting outliers, i.e., observations less than the median minus ten times the standard deviation or greater than the median plus ten times the standard deviation, the algorithm overwrites these values with a linear interpolation between the non-anomalous precursor and the non-anomalous successor.

6.2.2 Modeling

After imputing missing values and removing outliers, the workflow performs the main part, namely modeling. This part is shown as the bottom gray box in Figure 6.3. It takes the preprocessed time series as input and eventually provides the forecast.

6.2.2.1 Frequency Estimation

In the modeling component, the approach first estimates the seasonal frequency of the time series. Although the imputation algorithm assumes a daily seasonal pattern, this step tests whether there is further seasonality present after imputing gaps and removing anomalies. Telescope already provides such a frequency estimation method that uses a periodogram to extract the most dominant frequencies and, subsequently, searches for meaningful human-based frequencies nearby. For more information on Telescope's frequency estimation method, please see Section 4.2.1. Such human-based frequencies include 60 (minutely measurements with hourly seasonality), 24 (hourly measurements with daily seasonality), or 365 (daily measurements with yearly seasonality).

6.2.2.2 Feature Generation

After estimating the time series frequency, lags of the univariate time series are generated. Here, the approach uses lags one to six for all time series to provide the most recent data as features. If the time series exhibits a seasonal pattern, the approach also incorporates the seasonal lag to provide not only the most recent data as features for the machine learning models, but also those from a season ago. For such seasonal time series, these longer lags contain even more important information than the smaller lags.

Apart from the lagged time series, the approach also provides the hour of the day, the day of the week, and whether the day is a holiday as features for the machine learning models. Since this approach is specifically designed for human-based time series, this temporal information can add relevant information in terms of additional seasonality. With respect to the holiday feature, this information can also explain deviations from normal behavior.

6.2.2.3 Classification

In the subsequent step, the algorithm determines the number of unique values within the time series. We have found that for time series with only few different values and, most importantly, no trend pattern, classification can be advantageous over regression models. Thus, if the algorithm observes that a time series consists of less than six different values, it learns a Random Forest multi-class classification model (see the right-hand side of the gray *Modeling* box in Figure 6.3). In this case, the classification model contains as many classes as the time series has different values, with each class representing the corresponding value. Moreover, in the special case where the time series consists of only one unique value, the algorithm forecasts exactly this value, since the available training data do not contain any further information.

6.2.2.4 Regression

In contrast, the workflow applies twelve different forecasting and regression methods if it encounters six or more different values in a time series, since the "No Free Lunch Theorem" states that there cannot be a single best method for all scenarios [WM97]. This procedure is illustrated on the left-hand side of the gray *Modeling* box in Figure 6.3. To this end, the algorithm again splits the available training data into a training set and an in-sample validation set. The in-sample validation set contains the last observation values of the time series and is as long as the demanded out-of-sample forecasting horizon, while the training set contains all previous values. In the remainder of this section, we

use the term training data to refer to this subset of the original training data and in-sample validation data to refer to this forecasting horizon within the original training data. In terms of test data, we continue to refer to the out-of-sample data of the time series.

The twelve forecasting methods implemented in the history-based forecasting method recommendation approach are median, mode, ARIMA, ETS, NNetAR, Random Walk, sNaïve, TBATS, Telescope with frequency estimation enabled, Telescope with frequency estimation disabled, Random Forest, and XGBoost. The first two methods forecast only a constant value of the training part, i.e., the median and the mode, respectively, for the entire forecasting horizon. ARIMA, ETS, NNetAR, Random Walk, sNaïve, TBATS, and the two Telescope versions also interpret the data as univariate time series. While the first six methods simply forecast the univariate time series, both versions of Telescope learn internal features but do not use the features described in Section 6.2.2.2. In contrast, Random Forest and XGBoost receive the lagged time series, the hour of the day, the day of the week, and the holiday as features. For both machine learning methods, namely Random Forest and XGBoost, we have carried out a hyperparameter tuning. The best parameter settings for Random Forest and XGBoost are shown in Table 6.2.

Table 6.2: Parametrization of Random Forest and XGBoost.

Method	**Parameters**
Random Forest	`ntree` = 500, `mtry` = 5
XGBoost	`nrounds` = 2000, `booster` = gbtree, `eta` = 0.2, `gamma` = 0, `subsample` = 0.9, `max_depth` = 10, `objective` = reg:linear, watchlist = list(train, val)

Due to this approach being designed for multi-step-ahead forecasting, the lag features must be continuously created during execution time. Accordingly, the original values given by the training data are initially used and gradually extended by the forecasts made. More precisely, the approach forecasts each value in the horizon as a one-step-ahead forecast and after each of these one-step-ahead forecasts, the feature set for the next value is generated. Note that the forecast model still remains the same, with only the feature set being recreated for each value in the forecasting horizon.

In order to estimate which forecasting method performs best for a given time series, the approach uses the in-sample validation data to compute the R^2-score of each method. As described in Section 2.6.1.5, the R^2-score is defined

as one minus the ratio between the sum of squares of forecast errors and the sum of squares of forecast errors when forecasting the actual mean of the forecasting horizon. Thus, the R^2-score has an upper bound of 1 but no lower bound, while a higher R^2-score implies a better forecast. Note that the normalization baseline of the R^2-score is only a theoretical concept, since it already contains information on the actual values in the forecasting horizon. Once the algorithm computed all forecasts and assessed their R^2-scores, the approach selects the forecasting method that achieved the highest R^2-score and learns a new model using the entire time series (i.e., the training and in-sample validation data). Finally, the approach forecasts the entire forecasting horizon using the presumably best forecasting method. In addition, the algorithm adjusts the forecasts under the assumption that human-based data typically do not contain negative values. Therefore, it sets negative forecasts to zero, but only if all values in the training data are non-negative.

The evaluation of the proposed history-based forecasting method recommendation framework is presented in Section 7.2.

6.3 Summary and Discussion

To conclude this chapter, we provide an answer to research question **RQ A.3**, while also summarizing the core contribution of this chapter. Research question **RQ A.3** deals with the challenge of designing forecasting method recommendation approaches to improve forecast accuracy. To this end, we have presented two approaches. While the first approach is data-based and, hence, requires a large and, in particular, diverse database, the second approach does not require such a database, but is purely based on the assumption that newly arriving data of a time series are strongly related to the last known data. The data-based approach builds upon the approach of X. Wang *et al.* to derive rules linking time series characteristics and the accuracy of forecasting methods. In this context, shortcomings of the approach of X. Wang *et al.* were pointed out and measures were proposed to solve them. More specifically, our approach addresses the problem of imbalanced data by oversampling, overfitting of the single decision tree by applying an ensemble learner, and missing recommendations by learning models with probabilities as output, which allows recommending the forecasting method with the highest probability of being suitable for the particular time series. The second approach, by contrast, considers only the history of a time series and splits it into a training part and an in-sample validation part. While all forecasting methods are learned individually on the training part, the in-sample validation part is used to determine the forecast accuracy.

Based on the assumption that future values are similar to the last known values, the forecasting method that performed best on the in-sample validation part is recommended for the forecasting task.

Due to the design of the data-based forecasting method recommendation approach, a very large and, above all, diverse database is needed, otherwise the learned rules will not generalize well. If such a database is not available, the history-based forecasting method recommendation approach should rather be used. This in turn requires time series of a certain length, since splitting the historical data into internal training and in-sample validation parts might otherwise result in too short time series. Moreover, the assumption that future values will be similar to the last known values is not necessarily given.

Chapter 7

Evaluation of the Meta-Learning Approaches for Time Series Forecasting Method Recommendation

In this chapter, we evaluate the forecasting performance of the two forecasting method recommendation approaches. Thus, Section 7.1 presents the results of the data-based approach. Here, we first thoroughly examine the approach of X. Wang *et al.* and, subsequently, compare it with our approach. In addition, we also compare the forecasting method recommendation approaches with the individual forecasting methods included in them. The contents of this section are based on our previous work published as a full paper as part of the 16th IEEE International Conference on Autonomic Computing (ICAC) [ZBL+19]. The history-based forecasting method recommendation approach is evaluated in Section 7.2. Similar to the data-based forecasting method recommendation approach, it is compared with the individual forecasting methods included in the recommendation framework, among other studies. The contents of this section are based on another previous work published as a short paper at the 15th International Symposium on Advanced Artificial Intelligence in Applications (AAIA) as part of the 15th Conference on Computer Science and Intelligence Systems (FedCSIS) [ZK20]. Finally, we conclude this chapter with a brief discussion in Section 7.3.

7.1 Evaluation of the Data-based Forecasting Method Recommendation

This section evaluates the forecast accuracy achieved by the approach of X. Wang *et al.* and our data-based forecasting method recommendation approach. First, Section 7.1.1 describes the experimental setup. Next, Section 7.1.2 examines the quality of the rules proposed by X. Wang *et al.*, i.e., it addresses the first two aspects raised in Section 6.1. In Section 7.1.3, we compare our own

approaches with a dynamic version of the approach proposed by X. Wang *et al.* Subsequently, we discuss threats to validity in Section 7.1.4 and summarize our results in Section 7.1.5.

7.1.1 Experimental Setup

The entire analysis is implemented in *R* version 3.3.2. For the calculation and normalization of time series characteristics, the original script, which was already used by X. Wang *et al.*, is used. The *R* package `forecast` is integrated in version 7.3 to perform the forecasting methods [HAB+18].

As we aim to evaluate the forecast accuracy achieved by the state-of-the-art recommendation approach of X. Wang *et al.* and compare our novel approaches with it, we first use the data set and the same forecasting methods mentioned in the original paper. Nevertheless, it would be an interesting aspect for future work to include other forecasting methods, such as TBATS, Theta, SVM, and Telescope. Moreover, as some of the data sources are no longer available, we focus on the time series from the UCR Time Series Classification Archive [CKH+15], which constitute the main part of the original data set. This data set consists of 377 time series and will be referred to as UCR hereafter. We use this data set to evaluate the approach of X. Wang *et al.* on the data set the rules were learned on. In addition, we use another data set consisting of 1005 time series from the M3 competition [MH00] to investigate whether the rules postulated by X. Wang *et al.* are also transferable to other data sets. Finally, we use both data sets to compare our approaches with the rules of X. Wang *et al.* and the individual forecasting methods.

In order to assess the recommendation quality, we calculate three evaluation measures. The first measure is the average rank achieved by the recommendations. By rank, we refer to the same concept as in the evaluation of Telescope in Chapter 5. Thus, the rank reflects the position at which the method is placed according to the forecast accuracy. The second evaluation measure compares the forecast error of each recommended forecasting method with the actual best performing forecasting method in the competition. We refer to this second evaluation measure as accuracy degradation, which indicates how much worse the corresponding forecasting method performs compared to the actual best performing method. Note that the ground truth of which forecasting method would have been the best is only known ex post. For both measures, we chose MAPE as forecast error measure, since it is independent of the value range of the input and can therefore be aggregated over several time series with different value ranges of the inputs. The third and final evaluation measure is the percentage of missing recommendations.

7.1.2 Evaluation of the Approach by X. Wang et al.

First, we evaluate the postulated rules of X. Wang *et al.* [WSMH09] using the average rank achieved by the recommendation system. To evaluate the rules from X. Wang *et al.* on both their own training data as well as on another data set, Table 7.1 presents the average ranks when applying the rules from X. Wang *et al.* on the UCR and M3 data sets. For this purpose, we apply only the rules reported by X. Wang *et al.* On both data sets, we examine the accuracy of both one-step-ahead and the multi-step-ahead forecasting. In addition to the average ranks achieved by the rules of X. Wang *et al.*, we also applied the individual forecasting methods. Table 7.1 also reports the average ranks of these four individual forecasting methods. The average rank of the recommendation rules varies from 2.62 to 2.80. Since there are only four forecasting methods in competition, the theoretical expected value of a random guess would be 2.5 if all forecasting methods ranked best with equal frequency. In addition, it can be seen that the average ranks of ARIMA, ETS, and NNetAR are better than the average ranks of the recommendation rules for both the one-step-ahead and multi-step-ahead scenarios of the UCR data set. Only Random Walk shows a worse average rank for the UCR data set. Note that X. Wang *et al.* used the UCR data set to learn their rule set, which makes these results even more surprising. With respect to the M3 data set, ARIMA and ETS again outperformed the recommendation rules for both scenarios. Even Random Walk shows a smaller average rank than the rule-based selection for one-step-ahead forecasting. Only NNetAR provided slightly worse average ranks than the rules of X. Wang *et al.* for both scenarios on the M3 data set.

Table 7.1: Average ranks for the rules by X. Wang *et al.* and all individual forecasting methods. The best values for each scenario are highlighted in bold.

Data set	X. Wang *et al.*	ARIMA	ETS	NNetAR	RW
UCR one-step	2.62	**2.33**	2.42	2.34	2.92
UCR multi-step	2.80	**1.95**	2.64	2.59	2.83
M3 one-step	2.63	2.38	**2.37**	2.67	2.58
M3 multi-step	2.75	2.28	**2.17**	2.79	2.76

The distribution of achieved ranks when applying the recommendation rules can be seen in Figure 7.1. Here, the horizontal axis represents the four ranks and the vertical axis shows the respective probability density. For each rank, four bars are displayed, where each bar represents one scenario of one data set, i.e., from left to right: one-step-ahead forecasting on UCR, multi-step-ahead

forecasting on UCR, one-step-ahead forecasting on M3, and multi-step-ahead forecasting on M3. The figure shows that the most frequent rank yielded is rank 4, representing the worst forecasting method in the system. This means that for all four scenarios, the decision rules recommend the actual worst method most often. In figures, the probability density of rank 4 is more than 30% for each scenario.

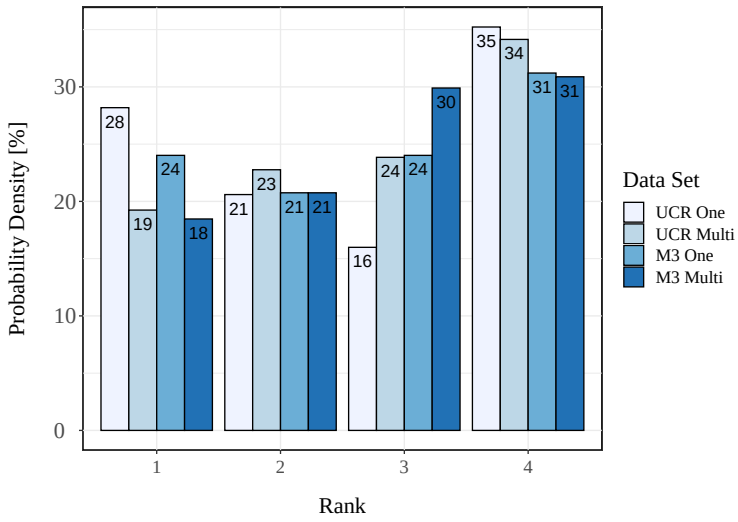


Figure 7.1: Histogram of the distribution of achieved ranks for the rules by X. Wang *et al.* on the UCR and M3 data sets.

Given that the rules generated by X. Wang *et al.* provide a separate recommendation for each forecasting method as to whether or not it should be applied, it is possible that the rules do not recommend any of the forecasting methods. However, this does not help autonomous systems in predicting critical events or decision-making in general. Forecasts are always required to identify impending critical events or to make decisions proactively. Table 7.2 shows the number and percentage of missing recommendations for the data sets UCR and M3. For each data set, the values for one-step-ahead and multi-step-ahead forecasting are the same, since the recommendation depends only on the characteristics of the training data. It can be seen that the recommendation system fails to suggest a forecasting method for almost 15% of all time series in the UCR data set and even 44% in the M3 data set.

Apart from the ranking and the investigation of missing recommendations, the accuracy degradation is also examined. Thus, for each time series and forecasting method, the achieved MAPE is compared with the lowest MAPE

Table 7.2: Missing recommendations for the rules of X. Wang *et al.*

Data Set	Number of Missing Recommendations	Percentage Share
UCR	55	14.6%
M3	442	44.0%

value for that particular time series. Table 7.3 reports the average accuracy degradation. For each data set, the first row shows the mean value. It can be seen that the mean is typically very large due to the presence of many outliers. Moreover, for one-step-ahead forecasting on the M3 data set, the mean values of X. Wang *et al.*, ARIMA, ETS, and NNetAR are infinite, since Random Walk reached a MAPE of 0% for a couple of time series. For this reason, the median is also provided in the second row for each data set. Again, ARIMA and ETS provide better median results for all data sets and scenarios compared to the rule-based selection by X. Wang *et al.* NNetAR also provides a lower median accuracy degradation than the recommendation rules in most cases. Only the median of the multi-step-ahead forecast on M3 is slightly worse than applying the rules of X. Wang *et al.* When comparing the recommendation rules with Random Walk, neither the rules of X. Wang *et al.* nor Random Walk have a significant advantage in the presented scenarios.

To conclude our goal of evaluating the recommendation quality of the rules of X. Wang *et al.* on their own training data, we come to the finding that the

Table 7.3: Accuracy degradation for all forecasting methods and the rules by X. Wang *et al.* For each data set, the first row shows the mean value and the second row the median, with the best values being highlighted in bold.

Data set	X. Wang *et al.*	ARIMA	ETS	NNetAR	RW
UCR	992.6%	2096.9%	**312.7%**	404.0%	1053.2%
one-step	96.8%	**46.1%**	55.8%	48.6%	130.1%
UCR	282.3%	**92.5%**	328.5%	391.4%	284.6%
multi-step	108.5%	**3.1%**	100.1%	74.1%	89.8%
M3	Inf	Inf	Inf	Inf	**1150.8%**
one-step	106.4%	**40.4%**	42.5%	68.7%	62.3%
M3	69.4%	53.7%	**42.6%**	92.0%	81.6%
multi-step	33.4%	10.0%	**7.9%**	35.1%	34.0%

postulated rules of X. Wang *et al.* on the UCR data set achieve average ranks and accuracy degradations worse than those of ARIMA and ETS. Thus, the mere use of one of these methods would provide a better forecast accuracy. Therefore, the recommendation quality is insufficient. ARIMA and ETS also outperform the rules of X. Wang *et al.* on the M3 data set. Moreover, the percentage of missing recommendations increases dramatically compared with the UCR data set. Thus, regarding the second evaluation criterion for the rules of X. Wang *et al.*, namely testing the transferability of their postulated rules, we conclude that the rules are not transferable, as they provided only inferior forecast quality also on the M3 data set.

7.1.3 Evaluation of Alternative Approaches

To evaluate our proposed alternative approaches, we use randomly sampled 80% of the time series from both data sets to learn the dependency between time series characteristics and the performance of the forecasting methods. Subsequently, the remaining 20% of the time series from both data sets are used to evaluate the recommendation systems. Given that such a split might be arbitrary, the data sets are randomly split 100 times and the results of all permutations are averaged. Due to the fact that proactive planning in autonomous systems, such as Self-Aware Computing Systems, usually requires forecasts of the variable under observation for several time steps at once, we only examine multi-step-ahead forecasts in the next sections.

7.1.3.1 Binary Classification with Oversampling

In general, it is not possible to cover all aspects of time series in such small training databases like the UCR or M3 dataset (cf. Section 7.1.1). Thus, either a huge and diverse time series database is needed or the learned recommendation system targets only a specific domain. As our evaluation is based on two data sets, we consequently create the rules twice, i.e., once for each data set. Thus, unlike X. Wang *et al.*, we do not provide general rules for selecting forecasting methods. Instead, we learn the rules dynamically on each data set under study, which leads to a more realistic setting of a recommendation system optimized for a particular domain.

Tables 7.4, 7.5, and 7.6 show the comparison of (I) the rules by X. Wang *et al.*, using the C4.5 algorithm for rule generation, with (II) the dynamic generation of rules following the procedure of X. Wang *et al.*, but using the C5.0 algorithm, an extension of the C4.5 algorithm, and with (III) our approach based on binary classification with oversampling. Replacing the C4.5 algorithm with the C5.0

algorithm, which generally produces more accurate results, updates the X. Wang *et al.* approach to the current version. Dynamically generating the rules means learning the rules on the training portion of each data set rather than learning a global, static set of recommendation rules for all data sets. Apart from this adaptation of dynamic rule learning and the use of C5.0 instead of C4.5, approach (II) applies the rule induction process of X. Wang *et al.* To compare the three approaches, we use average ranks, missing recommendations, and average accuracy degradation as evaluation measures.

With respect to the average ranks (cf. Table 7.4), both approach (II) as well as our approach show substantial improvements. However, for the M3 data set, approach (II) improved the average rank even more than our approach. However, the difference between the average ranks of these two approaches is rather small, i.e., 0.07. In contrast, our approach outperformed approach (II) on the UCR data set with respect to the average rank by 0.03. Again, the difference between the two approaches is quite small.

Table 7.4: Average ranks for the rules by X. Wang *et al.* (I), learned rules using C5.0 without oversampling (II), and our binary classification with oversampling approach (III). The best values for each data set are highlighted in bold.

Approach	UCR	M3
(I) Original Rules from X. Wang *et al.*	2.80	2.75
(II) Learned Rules - no Oversampling	1.89	**2.07**
(III) Binary Classification with Oversampling	**1.86**	2.14

Concerning the average degradation of accuracy (cf. Table 7.5), our approach and approach (II) again yielded very similar results on both data sets. The measure achieved by our approach is similar to that of approach (II) on the M3 data set and is only 0.2 percentage points worse than approach (II) on the UCR data set. Yet, both the dynamic rule learning modification of X. Wang *et al.*, i.e., approach (II), and our approach considerably reduced the average accuracy degradation compared with the static rules postulated by X. Wang *et al.*

Nevertheless, when considering the missing recommendations (cf. Table 7.6), it is evident that approach (II) does not provide a recommendation for a very large fraction of the time series. In fact, the percentage of missing recommendations increased dramatically compared with the rules of X. Wang *et al.*, i.e., from 14.5% to 31.6% and from 44.0% to 79.6%. Thus, approach (II) performs a kind of "cherry-picking", since it provides recommendations only for time series for which it is relatively confident. In contrast, our approach guarantees to recommend a forecasting method for all time series. Thus, our approach rec-

Table 7.5: Average degradation in accuracy for the rules by X. Wang *et al.* (I), learned rules using C5.0 without oversampling (II), and our binary classification with oversampling approach (III). The best values for each data set are highlighted in bold.

Approach	UCR	M3
(I) Original Rules from X. Wang *et al.*	282.3%	69.4%
(II) Learned Rules - no Oversampling	**79.5%**	**39.5%**
(III) Binary Classification with Oversampling	79.7%	**39.5%**

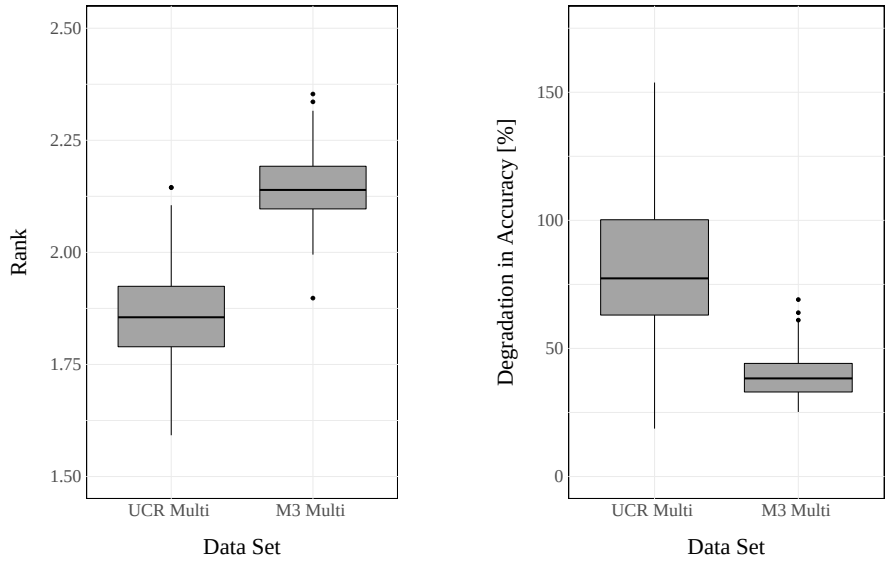
ommends a forecasting method even if none of them is perfectly fitting for the time series. For such scenarios, it is difficult to estimate which of the forecasting methods would perform best, as the models try to learn which forecasting method performs well for certain time series characteristics. However, in this scenario, there is no forecasting method that performs well, but still the one with the smallest forecast error should be recommended. Thus, it is reasonable that the average rank and accuracy degradation is slightly larger than for approach (II). Beyond that, in practice, it is impractical for most autonomous systems not to recommend a forecasting method, since they need this input for their proactive decision-making process.

Table 7.6: Share of missing recommendations for the rules by X. Wang *et al.* (I), learned rules using C5.0 without oversampling (II), and our binary classification with oversampling approach (III). The best values for each data set are highlighted in bold.

Approach	UCR	M3
(I) Original Rules from X. Wang *et al.*	14.5%	44.0%
(II) Learned Rules - no Oversampling	31.6%	79.6%
(III) Binary Classification with Oversampling	**0.0%**	**0.0%**

As the results of the binary classification with oversampling approach presented in Tables 7.4, 7.5 and 7.6 were averaged over 100 random splits for training and testing, the results of all splits are also provided in the form of box plots. To this end, Figure 7.2a presents the box plots for the ranks and Figure 7.2b displays the distributions of accuracy degradation. For both figures, the data sets, i.e., UCR and M3, both for the multi-step-ahead forecasting, are shown on the horizontal axes.

Figure 7.2a shows that the interquartile range, indicated by the top and bottom of the box plot, is very small for the M3 data set, as the first quartile has a value of 2.10 and the third quartile is at 2.19. In addition, there are only a few



(**a**) Ranks. (**b**) Degradation of accuracy.

Figure 7.2: Box plots of ranks and accuracy degradation for 100 splits using the binary classification with oversampling approach on the M3 and UCR data sets.

outliers. Thus, this shows that the average rank for the oversampling approach is barely dependent on the choice of the split between training and validation data. For the UCR data set, the variation is a bit larger. The interquartile range here is from 1.79 to 1.92. However, this range is still acceptable.

With respect to the degradation of accuracy, Figure 7.2b reveals that the deviation for the M3 data set is very small, i.e., the first quartile is at 33.0% and the third quartile at 44.2%. This underlines the previous statement that the performance of this approach does not depend much on the split of the data. In contrast, the degradation in accuracy varies greatly on the UCR data set. Here, the interquartile range is much larger compared to the M3 data set, ranging from 63.0% to 100.2%. Moreover, the long whiskers indicate that some of the splits strongly affect the deterioration of accuracy on the UCR data set, presumably because this data set is smaller than the M3 data set and is also intended for time series classification rather than forecasting.

Figure 7.3 provides a histogram of the obtained ranks for the binary classification with oversampling approach for the data sets UCR and M3. The horizontal axis shows the ranks, while the vertical axis depicts the probability density. Two bars are shown for each rank, with the left bar representing the UCR data set and the right bar representing the M3 data set. Unlike the histogram of

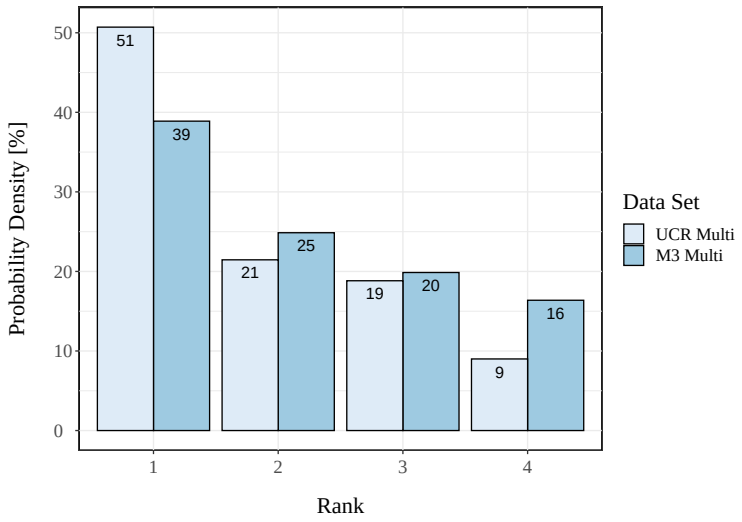


Figure 7.3: Histogram of the rank distribution for the approach using binary classification and oversampling on the UCR and M3 data sets.

achieved ranks for the original rules by X. Wang *et al.* (cf. Figure 7.1), the probability densities of the ranks are strictly decreasing. Thus, rank one is by far the most frequent for both data sets with 51% and 39% for the UCR and the M3 data set, respectively. In addition, the worst forecasting method for each time series, i.e., rank four, is the least recommended (9% for the UCR data set and 16% for the M3 data set), with a distinct distance to the other ranks.

Considering the average rank, accuracy degradation, distribution of ranks, and share of missing recommendations our binary classification with oversampling approach demonstrates considerable improvements in the recommendation quality. Moreover, when comparing the recommendation quality with each individual forecasting method, our approach also outperformed them. On the UCR data set, ARIMA performed best with an average rank of 1.95 (cf. Table 7.1) and an average accuracy degradation of 92.5% (cf. Table 7.3). Accordingly, the average rank of our approach is 0.09 lower and the average accuracy degradation is 12.8 percentage points less (cf. Table 7.4). On the M3 data set, ETS achieved the best results of the individual prediction methods. However, our approach still obtained better results than ETS with a reduction of 0.03 in the average rank (cf. Tables 7.1 & 7.4). Moreover, our approach reduced the accuracy degradation by 3.1 percentage points (cf. Tables 7.3 & 7.5).

7.1.3.2 Recommendation-based Ensemble Forecasting

In order to evaluate the recommendation quality of the linear ensemble model with activation function, Table 7.7 reports the accuracy degradation for all forecasting approaches when using multi-step-ahead forecasting, averaged over all 100 splits. As this linear model approach generates a completely new forecast, it can be considered a fifth forecasting method in addition to ARIMA, ETS, NNetAR, and Random Walk. Thus, a comparison of the ranks would yield completely different results than those shown above. Therefore, only the accuracy degradation is shown for this approach. However, note that the values of accuracy degradation here are slightly larger than in the previous sections, since the optimal forecast may now be the recommendation-based ensemble forecasting approach. Table 7.7 demonstrates that the recommendation-based ensemble forecasting approach achieved the lowest degradation in accuracy for the M3 data set, followed by the binary classification with oversampling approach and ARIMA. All other approaches were clearly outperformed. With respect to the UCR data set, the binary classification with oversampling approach achieved the best result ahead of ARIMA and the recommendation-based ensemble forecasting approach. Again, the other approaches were all far behind. On the M3 data set, the rules by X. Wang *et al.* outperformed only NNetAR and Random Walk, while they performed worst on the UCR data set.

Table 7.7: Degradation in accuracy for all approaches with multi-step-ahead forecasting, averaged over all 100 random splits. The best values for each data set are highlighted in bold.

Method	M3	UCR
Original Rules from X. Wang *et al.*	69.8%	404.9%
ARIMA	59.9%	98.0%
ETS	46.2%	345.6%
NNetAR	91.0%	386.3%
Random Walk	83.6%	300.9%
Binary Classification with Oversampling	42.3%	**82.1%**
Recommendation-based Ensemble Forecasting	**40.6%**	104.9%

7.1.4 Threats to Validity

We have tried to reconstruct the original data set used by X. Wang *et al.*, which consists mainly of the UCR data set. Unfortunately, the exact time series have

not been reported, but only the counts. However, by analyzing the year of publication of X. Wang *et al.*, we were able to identify the data sets used from the UCR archive. Still, we could not use time series from the other sources, as some of them were no longer available or could not be clearly identified. In total, we used 46 of 62 time series data sets. As these are almost 75% of the original data set, the results for the rules proposed by X. Wang *et al.* might be slightly better when using the entire original data set, but the missing time series would not cause the rules to perform best compared with the individual methods and our proposed approaches. Furthermore, X. Wang *et al.* postulate universally applicable rules, so they should be independent of the data set.

Moreover, the time series from the UCR data set originally aim at classification, not forecasting. The predictability of these time series is rather poor, since their entropy is very low, e.g., some time series exhibit only Dirac impulse-like peaks. For this reason, the forecasts are generally relatively inaccurate. This results in a largely random ranking of the best forecasting methods for such time series. Although Random Walk is typically used as a baseline forecasting method, this approach performs comparatively well on these time

Given that the rules of X. Wang *et al.* did not recommend any forecasting method for several time series, we omitted these time series from the evaluation in Section 7.1.2. If missing recommendations were penalized, for instance by setting their rank to the worst possible, the approach of X. Wang *et al.* would have performed even worse.

As we tried to reproduce the setup of X. Wang *et al.*, we only considered time series with more than 100 and less than 1000 values. Therefore, the obtained results might be valid only for time series with the given length. However, for a real-time system that operates in a dynamic environment, this short time frame might be valid to adapt to the rapidly and frequently changing environmental conditions that cause context drift in the data.

To ensure comparability, we have considered only the forecasting methods also used by X. Wang *et al.*, namely ARIMA, ETS, NNetAR, and Random Walk. Thus, the results are valid only when using these methods. However, since there are only four methods in the recommendation system, a significant improvement in the average rank is hard to validate. In addition, some of the forecasting methods are very similar, for instance, ETS and ARIMA often provide very similar results for short time series with little information content and short seasonal patterns. However, the data sets contain many time series of this type. To validate the recommendation system in more detail and independently of the approach of X. Wang *et al.*, more and especially more diverse forecasting methods should be integrated into the recommendation system.

7.1.5 Summary of Evaluation Findings

The approach proposed by X. Wang *et al.* has not yet been evaluated, so we examined the recommendation quality on the original and an additional, well-known time series data set. Furthermore, we introduced a novel forecasting method recommendation system that overcomes the shortcomings of the approach of X. Wang *et al.* as well as a recommendation-based ensemble forecasting approach. The key findings of our evaluation are:

(I) The experimental results reveal that the rules proposed by X. Wang *et al.* are outperformed by all forecasting methods except Random Walk on both data sets. Therefore, the static rules should not be implemented by autonomous systems to select time series forecasting methods.

(II) Compared with the rules of X. Wang *et al.* and the individual forecasting methods, our binary classification with oversampling approach yields the best results on both data sets for both average rank and average degradation in accuracy. Thus, we also show that the recommendation of forecasting methods can certainly provide better and more robust results than the individual methods on their own. Moreover, our recommendation approach resolves the "cherry-picking" issue, i.e., the lack of recommendations.

(III) The approach of using a linear regression model with activation function for the recommendation and combination of forecasting methods achieves the best results on the M3 data set and the third best results on the UCR data set. In contrast, the binary classification with oversampling approach achieves the second best results on the M3 data set and the best results on the UCR data set. Therefore, it cannot be definitively concluded which of the approaches is superior in general.

7.2 Evaluation of the History-based Forecasting Method Recommendation

This section presents the experimental results of the proposed history-based forecasting method recommendation framework based on the FedCSIS 2020 Challenge data set [JPBŚ20]. First, the necessity of time series preprocessing is demonstrated by means of example time series in Section 7.2.1. In Section 7.2.2, the forecast accuracy achieved by the history-based recommendation framework is compared with the individual forecasting methods integrated within it. Subsequently, Section 7.2.3 provides further details on the distribution of recommended forecasting methods. Section 7.2.4 discusses threats to validity, followed by a brief summary of the main evaluation findings in Section 7.2.5.

7.2.1 Time Series Preprocessing Steps

As the time series in the FedCSIS 2020 Challenge data set represent real-world monitoring data of network device workloads, they also show missing values. However, in order to apply forecasting methods, these must first be imputed. Figure 7.4 depicts an exemplary time series of the FedCSIS 2020 Challenge data set with a gap of seven missing values. Here, the original time series is displayed in black, while the green color indicates the imputation generated by our algorithm. It can be seen that the imputation produces reasonable reconstructions based on the existing values of the preceding and succeeding data points. In particular, the imputation algorithm even reconstructs a first spike for the double-spiked seasonal pattern, similar to the other seasonal highs, since the algorithm considers precursors and successors with a distance equal to the frequency of the seasonal pattern.

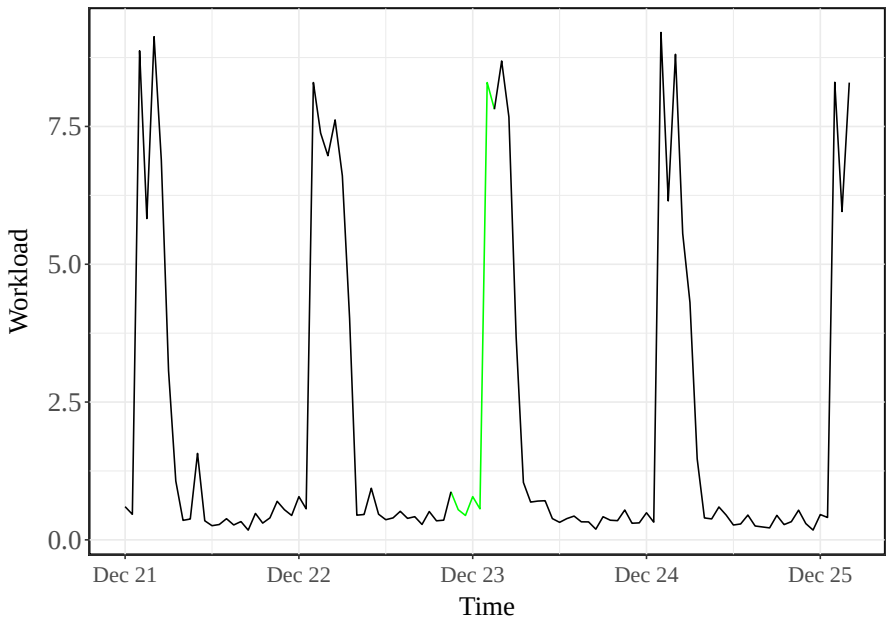


Figure 7.4: An example time series of the FedCSIS 2020 Challenge data set with imputed values highlighted in green.

While Figure 7.4 exemplarily demonstrates the imputation of missing values, Figure 7.5 illustrates an example of the second preprocessing step, namely the removal of anomalies in one of the 10,000 relevant time series of the FedCSIS 2020 Challenge. The black line indicates the corrected time series, while the red line displays the anomalous values in the original time series. It can be

seen that the peak value of the daily pattern significantly exceeds the normal range and, therefore, the anomaly detection algorithm overwrites these values by interpolating between the first non-anomalous precursor and the next non-anomalous successor. If such anomalies were not removed before modeling, the forecasting methods could learn a completely different, incorrect behavior. In particular, if the anomaly is at the end of a time series, as seen in Figure 7.5, the trend component may be biased such that the approximation would erroneously detect an exponential trend.

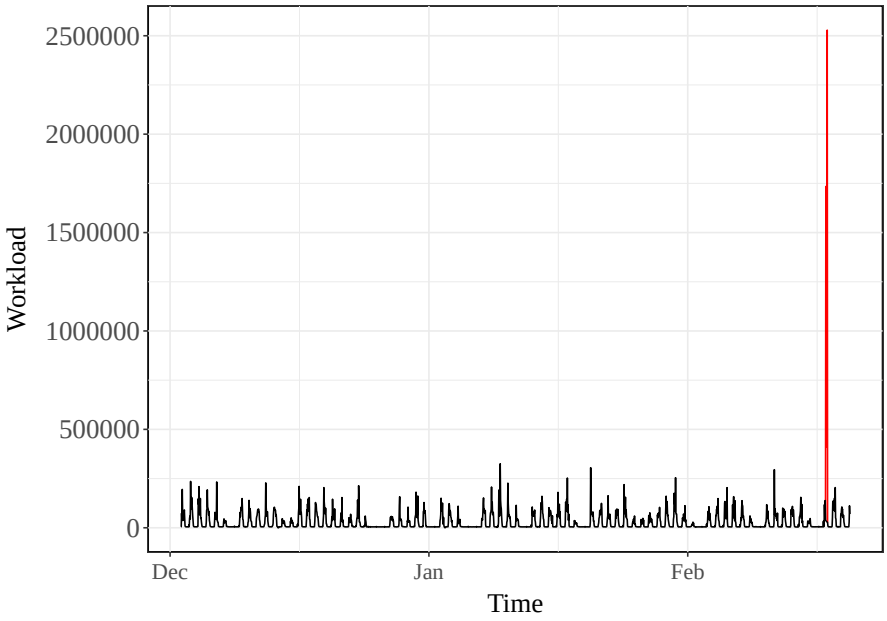


Figure 7.5: An example time series of the FedCSIS 2020 Challenge data set, with the removed anomaly highlighted in red.

7.2.2 Forecasting Accuracy for the FedCSIS 2020 Challenge

In total, the FedCSIS 2020 Challenge data set consists of 10,000 time series to be forecast. The time series represent the network device workloads in hourly resolution. Although the forecasting horizon of all time series is one week, i.e., 168 observations, the lengths of the training parts vary. As the evaluation measure of the FedCSIS 2020 Challenge was the R^2-score, we employed this measure as decision measure for the history-based forecasting method recommendation framework. Recall that unlike typical forecast error measures, a higher value of

the R^2-score indicates better forecast accuracy. However, to assess the forecast accuracy of the proposed recommendation framework and compare it with the individual state-of-the-art methods, we also computed the forecast error measures MASE and MAPE. As competing individual forecasting and regression methods, we selected all methods included in the recommendation pool of the history-based recommendation framework.

Given that we do not only consider the forecast accuracy per se, Table 7.8 illustrates the average rank achieved when ordering the forecasting methods from best to worst accuracy, the median forecast accuracy, and the median degradation in forecast accuracy with respect to MASE, MAPE, and R^2-score. The accuracy deterioration is calculated ex post by comparing the achieved forecast accuracy of the considered forecasting method with the forecast accuracy of the method that actually performed best. In terms of forecast accuracy and degradation in forecast accuracy, we considered the median to be a more meaningful measure, because the values of the measures on the 10,000 time series exhibited significant outliers biasing the aggregation of forecast accuracy.

First, the bold highlighted values clearly show that the proposed history-based forecasting method recommendation framework (HbRF) provided the smallest forecast error, respectively the highest forecast accuracy. Regarding the average rank, this can be observed as it achieved the lowest average rank for all three evaluation measures. Only Telescope II, i.e., Telescope using the internal frequency estimation algorithm, was able to keep up with the recommendation framework. The other individual methods were considerably outperformed. With respect to the median forecast accuracy, Random Forest was also able to keep up with Telescope II, as it achieved smaller median MASE and MAPE values. However, the median R^2-score of Telescope II was still better compared to Random Forest. More specifically, even Telescope without internal frequency estimation, i.e., Telescope I, achieved the same median R^2-score as Telescope with internal frequency estimation. Yet, the use of internal frequency estimation greatly improved the median MASE and MAPE. The advantage of using internal frequency estimation is also evident in the median accuracy degradation. Here, it improved the forecast accuracy with respect to all three evaluation measures. Random Forest, though, produced an even smaller median accuracy degradation than Telescope II. Nevertheless, the median accuracy degradation achieved by the history-based forecasting method recommendation framework is still substantially lower, with reductions of about 40%, 31%, and 36% with respect to MASE, MAPE, and R^2-score, respectively.

Overall, an interesting finding is that most conventional forecasting methods for univariate time series performed poorly. In fact, ARIMA, ETS, NNetAR,

Table 7.8: The average rank, median forecast accuracy, and median degradation in forecast accuracy, which is calculated as ex-post comparison between the respective forecast accuracy and the actual best forecast accuracy, obtained by the history-based recommendation framework (HbRF) and the twelve individual forecasting and regression methods with respect to MASE, MAPE, and R^2-score. The best values for each evaluation measure are highlighted in bold.

Forecasting Method	Avg. Rank			Md. Accuracy			Md. Accuracy Degradation		
	MASE	MAPE	R^2	MASE	MAPE	R^2	MASE	MAPE	R^2
HbRF	**4.487**	**4.751**	**4.033**	**1.085**	**17.492%**	**-0.003**	**5.688%**	**13.259%**	**4.363%**
Median	5.891	5.686	7.758	1.427	22.881%	-0.122	23.068%	26.416%	46.784%
Mode	7.228	6.232	9.131	1.534	25.415%	-0.282	34.275%	27.786%	78.653%
ARIMA	6.842	6.928	5.794	1.449	21.832%	-0.018	29.882%	42.726%	26.662%
ETS	7.665	7.676	6.899	1.606	26.835%	-0.063	39.824%	56.768%	40.776%
NNetAR	7.415	7.399	6.967	1.649	24.248%	-0.084	40.825%	54.993%	44.618%
Random Walk	8.019	8.168	8.556	1.800	27.153%	-0.267	51.100%	67.307%	74.169%
sNaïve	7.462	7.774	8.207	1.548	23.877%	-0.176	39.875%	56.376%	66.961%
TBATS	6.024	5.944	5.211	1.338	19.595%	-0.015	24.184%	35.003%	20.301%
Telescope I	5.208	5.417	4.642	1.222	19.040%	-0.004	14.141%	25.819%	11.567%
Telescope II	4.687	5.068	4.516	1.193	18.776%	-0.004	10.033%	21.826%	9.558%
Random Forest	5.364	5.667	4.698	1.186	18.607%	-0.006	9.502%	19.157%	6.836%
XGBoost	6.196	6.041	5.620	1.244	18.662%	-0.028	17.371%	25.960%	17.955%

Random Walk, and sNaïve provided worse forecasts than using a simple baseline, namely forecasting only the median for the entire forecasting horizon. Among the set of conventional forecasting methods for univariate time series, only TBATS was able to compete with the median forecasts. Considering the set of more advanced forecasting approaches, namely both Telescope versions as well as Random Forest regression and XGBoost regression, all four methods performed superior compared with the median baseline, although XGBoost performed considerably worse than both Telescope versions and Random Forest. In general, the best individual methods were Telescope II and Random Forest. However, since their ranking varied from evaluation scenario and measure to evaluation scenario and measure, it cannot be determined unambiguously which of these two methods performed better. Nevertheless, both were clearly outperformed by the proposed history-based forecasting method recommendation framework. However, it should be noted that the history-based forecasting method recommendation framework naturally results in a much larger runtime, since a model must be trained for each forecasting method in the recommendation pool to decide which method is most suitable.

7.2.3 Share of Forecasting Methods Recommended

In order to provide further insight into the results of the history-based forecasting method recommendation framework, Figure 7.6 depicts the distribution of forecasting methods recommended by the framework for the FedCSIS 2020 Challenge data set. Due to the lack of variation in 124 time series, the single unique value that forms the time series is forecast for the entire forecasting horizon. Further, 104 other time series have more than one but less than six individual values within the time series. For these time series, Random Forest classification (RF Class.) is applied. For the remaining 9772 time series, forecasting or regression is used. Median and mode are used 431 and 325 times, respectively. In agreement with the poor average and median forecasting performance (cf. Section 7.2.2), the conventional statistical forecasting methods were chosen rather infrequently. In terms of numbers, ARIMA, ETS, NNetAR, Random Walk, sNaïve, and TBATS were selected 611, 540, 786, 341, 88, and 696 times, respectively. Thus, such statistical forecasting methods were used a total of 3062 times. On the contrary, the more sophisticated approaches, namely Telescope I and II as well as Random Forest regression (RF Reg.) and XGBoost regression, collectively were used a total of 5954 times. Thus, these approaches were used almost twice as often as the statistical approaches, even though this set of approaches involves one-third fewer algorithms. Interestingly, although Telescope II and Random Forest regression achieved the best

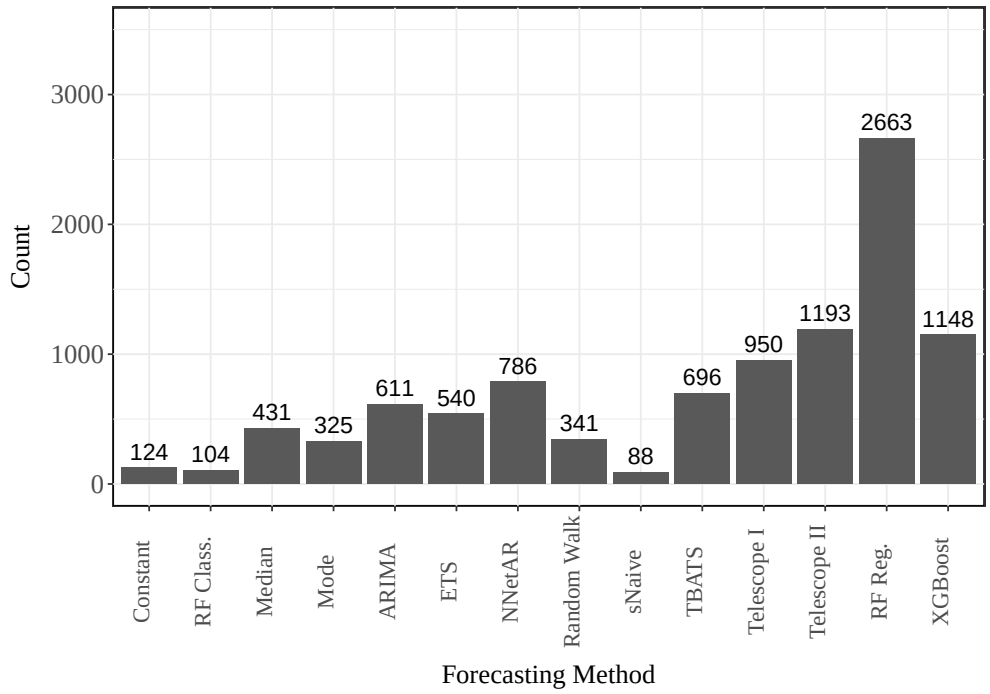


Figure 7.6: The distribution of methods used by the history-based recommendation framework over the 10,000 time series in the FedCSIS 2020 Challenge data set.

individual performances (cf. Section 7.2.2), the recommendation framework chose Random Forest regression about 123% more often. More specifically, with 2663 applications, the history-based forecasting method recommendation framework applied Random Forest regression even more frequently than both versions of Telescope combined.

7.2.4 Threats to Validity

The history-based forecasting method recommendation framework is based on the assumption that the accuracy of a forecasting method on the last part of the training set, i.e., the in-sample validation part, is related to the actual out-of-sample accuracy. However, this may not be the case for all types of time series. In particular, if a time series exhibits breakpoints at the end of the in-sample validation part or even later in the out-of-sample testing part, this assumption may not be fulfilled.

Furthermore, the history-based time series recommendation framework is only applicable to long time series, where separating the training time series into a smaller training part and an in-sample validation part is feasible. Otherwise, other aggregation or recommendation approaches should be employed. A possible approach to handle shorter time series is shown in Section 6.1, where characteristics are derived from the training part of the time series and classification models are learned to estimate the best forecasting method given the particular set of time series characteristics.

Finally, the time series of the FedCSIS Challenge data set represent only network device workloads. The statistical forecasting methods might perform better for other types of time series. As a reminder, when only the univariate time series were used as input to the forecasting methods, without deriving additional features, these methods achieved sufficient forecast accuracy that clearly exceeded that of the machine learning-based methods (cf. Chapter 5).

7.2.5 Summary of Evaluation Findings

The experimental results using the FedCSIS 2020 Challenge data set revealed the three main findings:

(I) Typical forecasting methods for univariate time series do not provide a sufficient forecast accuracy for network device workload forecasting. Instead, features should be derived to provide more accurate forecasts. This fact is demonstrated by the improved forecast accuracy obtained by both Telescope versions and the iterative regression methods Random Forest and XGBoost.

(II) The "No Free Lunch Theorem" also applies here, as the history-based forecasting method recommendation framework significantly outperformed all individual forecasting and regression methods. Considering only the individual forecasting methods, there is no unambiguously best method, although Telescope with internal frequency estimation and Random Forest regression appear to be the most accurate.

(III) The history-based forecasting method recommendation framework applied each method in the recommendation pool several times, resulting in a broad variety of applied forecasting methods. Nevertheless, the approaches using additional features (i.e., Random Forest regression and XGBoost regression) or deriving features themselves (Telescope with and without internal frequency estimation) were selected most frequently.

7.3 Concluding Remarks

The main contribution of this chapter is the evaluation of the proposed time series forecasting method recommendation approaches. Thus, this chapter provides an answer to research question **RQ A.4**. To highlight the most important outcomes regarding the comparison of the proposed time series forecasting method recommendation approaches with the state-of-the-art approach in the field as well as state-of-the-art individual forecasting methods, this section provides a brief summary.

First, using two time series data sets, we have shown that the state-of-the-art forecasting method recommendation system by X. Wang *et al.* does not provide satisfactory forecasting quality. Moreover, we have shown that the design of their recommendation system leads to a large proportion of missing recommendations. In contrast, our proposed data-based forecasting method recommendation system not only surpassed all individual forecasting methods but also the rules of X. Wang *et al.* Moreover, it ensures to suggest a forecasting method for every time series, rather than "cherry-picking" time series where the estimation of the best forecasting method is fairly distinct. A second data-based approach, which combines ensemble forecasting with forecast method recommendation, also exhibited high forecast accuracy. While our first approach performed best for one data set, the second approach was superior for the other data set.

Second, the history-based forecasting method recommendation framework, which does not rely on a large and diverse database of time series, but only on the historical observations of the time series under consideration, was evaluated on the FedCSIS 2020 Challenge data set. Here, we compared the recommendation framework with all individual forecasting methods included in the recommendation pool and demonstrated its superiority over the individual methods. In addition, we demonstrated the versatility of the forecasting methods chosen by the recommendation system.

Part III

Modeling, Detecting, and Predicting Machine Failures

Chapter 8

End-to-End Workflow for Automated Anomaly Detection of Industrial Machine Components

Rapid developments in the Internet-of-Things (IoT) and the continuous miniaturization of sensors are leading to new applications in areas such as Industry 4.0 and Industrial IoT. In Industry 4.0, the already existing automation of static work steps is extended by the automation of flexible work steps enabled by more intelligent self-adjusting machines [VHH16]. To this end, the machines are equipped with more sensors allowing extensive monitoring of the machines as well as data collection. However, the mere monitoring and collection of data is only a first step, providing no benefit by itself. Instead, the main goal is the intelligent analysis of monitoring data in combination with machine learning and data mining algorithms.

Early-stage machine anomaly detection is one such application that benefits from the intelligent use of monitoring data. A prominent example showcasing the immense impact of proactively identifying production problems is the case of Volkswagen in 2016, where production problems led to financial losses of up to 400 million Euros per week [Zei16]. Although this case highlights the relevance of automated and early mechanisms to detect machine failures, companies often implement periodic or threshold-based maintenance plans due to insufficient understanding of machine learning approaches that can be used to detect anomalies proactively. In contrast, machine failure prediction is a hot topic in academia, with many researchers contributing to the field. To this end, we have presented several approaches to machine failure prediction in Chapter 3, among others. The approaches referenced there use different methods to determine a machine health index and derive the remaining useful life. However, the applicability of the aforementioned approaches is highly dependent on the specific machines and the available data. Therefore, many companies struggle to identify an approach suitable for their problem and

to adapt the methods proposed in the literature to their individual machine. Therefore, in this chapter, we present a novel generalizable end-to-end workflow for automatic anomaly detection for machines that is tailored to the specific requirements of manufacturing machines. More specifically, the workflow targets multi-purpose machines with different tools. Automatic anomaly detection is a necessary step to predict machine deterioration and avoid equipment downtime. Specifically, the contributions of this approach are the following:

- We propose a clustering-based approach to segment raw machine data into different work steps and, thus, multiple machine tools (i.e., phases).
- We present a generic workflow for data preprocessing and oversampling that involves applying basic machine learning methods to learn the distinction between normal and degraded behavior in order to predict the current deterioration state of production machines.
- We evaluate the learned model with real-world data from a general-purpose machine to demonstrate its applicability to machine anomaly detection (cf. Chapter 9). The experimental results show that by using only basic machine learning classifiers (i.e., in contrast to deep neural networks), the workflow is able to learn the distinction between normal and anomalous states with only very little training data. This is an essential criterion for the practical application in Industry 4.0 that deep neural networks typically cannot fulfill, as they require large training data sets.

The content of this chapter is based on our earlier work, which has been accepted and is currently under publication in the journal ISA (International Society of Automation) Transactions [ZML+21]. The remainder of this chapter is organized as follows: Section 8.1 outlines the general data acquisition approach for multi-purpose machines. Then, Section 8.2 describes the overall design of the end-to-end workflow for machine anomaly detection. Finally, Section 8.3 concludes this chapter with a brief summary answering the corresponding research question. For details on the real-world case study and experimental results, please refer to Chapter 9.

8.1 Data Acquisition

In any data mining application, one of the most fundamental steps is data acquisition. In the field of industrial machinery, obtaining relevant data is often challenging. Although machines typically rely on sensor data for their

operation, extracting them for further analysis is difficult. However, with the increasing interest in data analytics in recent years, the interface for extracting sensor data has been enabled so that some of the limited computing power of industrial machines can now be used to leverage available data sources.

A subgroup of industrial machines is that of CNC (computerized numerical control) machines. The overall components of such a CNC machine with relevant sensors and the resulting data flow are illustrated in Figure 8.1. The core of the machine is its numerical control (NC), which controls motion by executing programmed instructions and is usually complemented by a programmable logic controller (PLC). Together, they form the central point for all data in the machine. The PLC is connected to the individual drive controllers via a real-time bus (RT Bus) system. Each drive is then connected to one or two motors. The drive controllers supply current to the motors according to the axis position required by the controller. They are also responsible for other motion-related tasks, such as acceleration and deceleration ramps, and can also interpolate motion. The motor is mechanically connected to the moving parts of the machine. Internal sensors provide the drive with measured values necessary to control the motion. These data can also be transmitted to the control system (NC/PLC) via the bus system. However, the ability to extract such high-frequency data from the control system is highly manufacturer-specific, if it exists at all.

We had access to a five-axis CNC milling machine for the approach presented in this chapter. It allows X, Y, and Z motions, among other capabilities, and is equipped with Bosch Rexroth components, such as controller, drive, and motors. A tool magazine with numerous slots enables automatic tool changes. In addition, a software tool for data acquisition with the NC controllers allows

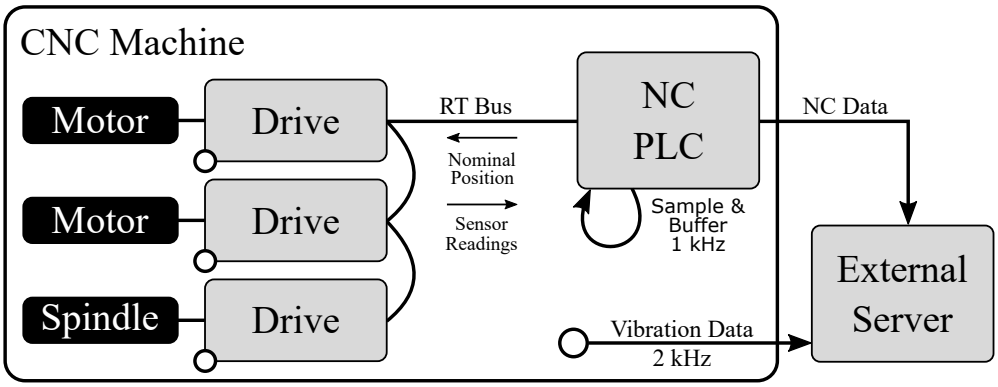


Figure 8.1: Data flow inside and outside of a CNC machine.

us to collect drive-internal data. We chose the position and velocity of the axes along with torque and power draw from the DC intermediate circuit and temperature of the motors, as these are common standard parameters that should be captured by most of today's machines. The collected data are buffered by the controller and transmitted once the measurement is complete. Triggering can be done from the NC program, which is also recorded synchronously. This means that the program instructions can be directly linked to measurement values of physical signals. Recording synchronous NC commands simplifies phase detection, but it is difficult to apply this to different domains or applications where such data are not accessible. Here, we tackle the phase detection problem without relying on a recorded NC program (cf. Section 8.2.1). The NC program is only used as gold standard data to validate the results (cf. Section 9.2).

Besides the machine-internal sensors, an external vibration sensor is also used. Since vibrations inevitably occur in any rotating machine, this appears to be a natural choice. A position close to the main spindle of the machine was chosen to mount the sensor. Therefore, Fig 8.2a shows the machine with the sensor

(**a**) CNC milling machine. Circle: Position of the additionally mounted vibration sensor on top of the machine.

(**b**) Closer view on the mounting position of the vibration sensor. Compared to Figure 8.2a, the z-axis is rotated by 90 degrees.

Figure 8.2: CNC milling machine with highlighted sensor mounting position.

position circled, while Fig 8.2b provides a closer view of the sensor mounting position. The part on which the sensor is mounted moves with the spindle in X, Y, and Z directions. The sensor used (called CISS) is a multi-sensor specialized for industrial applications. Although it features 3-axis inertial sensors, such as accelerometer, gyroscope, and magnetometer, as well as several environmental sensors with a maximum sampling rate of 1 kHz, it also allows a sampling rate of 2 kHz if only the accelerometer is active. For our use case, a rotational motion of the main spindle with less than 10000 min^{-1} (about 167 Hz) is expected, so the sampling rate of 2 kHz is sufficient to capture the expected vibrations and harmonics. For this reason, we use the CISS only as a vibration sensor. We have chosen this sensor over more sophisticated measurement equipment, as it offers sufficient resolution and accuracy at a low price and ease of use. These advantages also make it suitable for retrofitting existing machines. Besides, a detailed acoustic analysis is out of the scope of this thesis.

8.2 Design of the End-to-End Machine Part Anomaly Detection Workflow

This section presents a novel, end-to-end workflow to automatically identify anomalous tools in CNC milling machines. For this purpose, Figure 8.3 provides a simplified overview of the approach. First, data acquisition takes place (cf. Section 8.1), followed by a data conversion step. During data conversion, the raw data extracted from the machine (typically in XML format) is converted into a readable format for further analysis. The remaining procedure of this workflow consists of two steps: (I) dividing the raw machine signals into different phases (cf. Section 8.2.1), each representing a particular production step, and (II) identifying anomaly effects in each phase (cf. Section 8.2.2). Here, the first part itself involves two steps: (I) a distinction between *on* and *off* phases

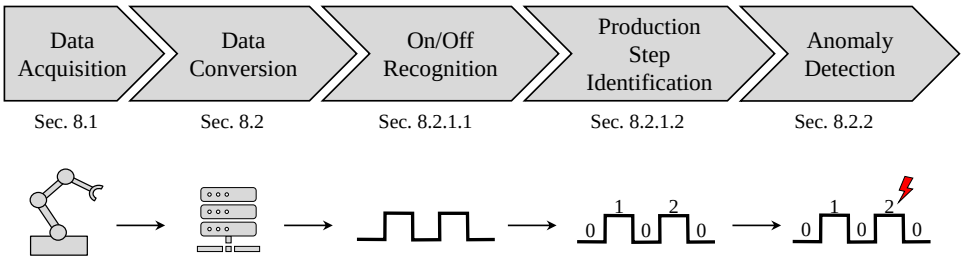


Figure 8.3: The overall end-to-end workflow for machine tool anomaly detection.

(cf. Section 8.2.1.1) and (II) a production step identification for the *on* phases (cf. Section 8.2.1.2). Here, an *on* phase represents the use of a tool, while an *off* phase corresponds to a tool change. Given that the proposed approach is an automated end-to-end workflow, the output of each step is directly used as input for the next step. The operator only needs to configure one parameter, which is the number of production steps executed by the CNC milling machine.

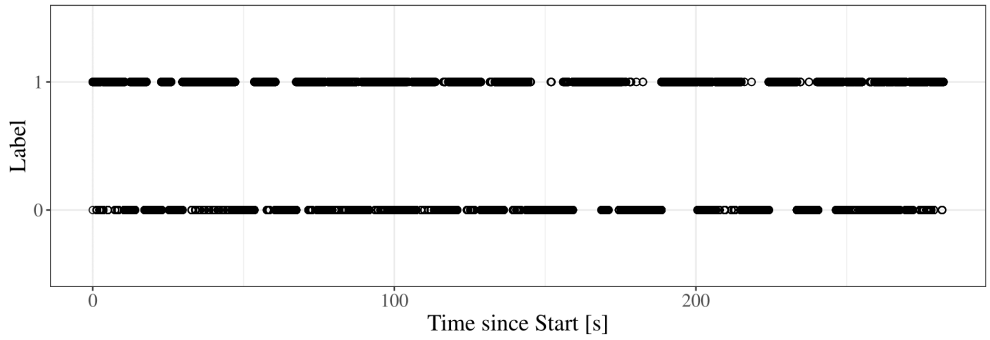
8.2.1 Phase Detection

This procedure aims at separating the raw data time series into individual signals that can be unambiguously assigned to specific work steps in the manufacturing process. Performing this step is necessary to achieve comparability of the individual work steps across several manufacturing processes. Given the overall goal of detecting anomaly effects for individual tools in the CNC milling machine, comparing time series of the entire manufacturing process does not provide viable results, as the variations within the time series are too subtle and are lost in the data set. The process of dividing the raw time series into phases again requires two main steps: (I) dividing the raw data into *on* and *off* phases and (II) assigning each *on* phase to a specific identifier. Such mechanisms are essential due to the fact that anomalies can only be detected for individual tools with their respective manufacturing processes. As mentioned earlier, it is not always feasible to extract the phases from the NC program, as this would limit the approach's applicability. In other words, this would require domain knowledge and, consequently, make it impossible to transfer the approach to other domains or use cases where these data are not accessible.

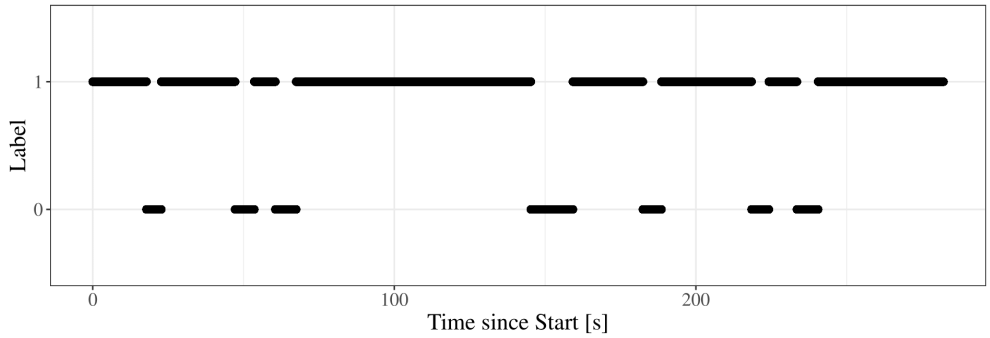
8.2.1.1 On/Off Recognition of Tool Change Phases

Due to the fact that anomaly effects of individual tools do not cause variations in the machine signals during the tool change phases, these must be filtered out. For this purpose, the raw machine signals are passed to this stage of the algorithm. Here, k-Means clustering [Llo82] is performed on the data, where k is set to two, since this component of the workflow should distinguish only between *on* and *off* phases. Accordingly, each measurement timestamp is assigned to either cluster label `1` (*on* phase) or cluster label `0` (*off* phase), depending on the signals from the machine. However, the use of k-Means clustering on noisy raw data results in many mislabelings. Figure 8.4a illustrates this behavior, with the horizontal axis showing the time since the start of the measurement and the vertical axis depicting the cluster label for each timestamp, i.e., either `1` or `0`, representing *on* and *off*, respectively. Each circle

in the figure indicates whether the particular set of features was clustered as an *on* or *off* phase. In the optimal case, we would want to have long sequences with the same cluster labels with no interruptions. These long sequences would represent a single phase. However, as evident in Figure 8.4a, there are many interruptions (i.e., rapid changes between cluster labels) within the long sequences that indicate mislabelings. Therefore, the approach implements a systematic threshold-based smoothing of the provided cluster labels.



(**a**) Clustering with many mislabelings (interrupted sequences of the same cluster label).



(**b**) Clustering without mislabelings (clear boundaries between sequences of different labels).

Figure 8.4: One manufacturing process divided into *on* (label `1`) and *off* (label `0`) phases according to the cluster labels.

To remove mislabelings, the derivative x' of the labels is computed first:

$$x' = \frac{dx}{dt} \tag{8.1}$$

As the signal is a discrete time series of length n, the derivative x' is equal to the first order difference of the time series x:

$$x'_i = x_{i+1} - x_i, \text{ for } i \text{ in } 1, ..., n-1 \tag{8.2}$$

Subsequently, the indices of the timestamps with a non-zero derivative are extracted. This ordered set of indices p indicates possible label changes:

$$p = \{0\} \cup \{i | x'_i \neq 0, \text{ for } i \text{ in } 1, ..., n-1\} \cup \{n\} \tag{8.3}$$

However, since p contains all potential cluster label changes, they may occur due to a new phase or due to a mislabeling within an actual phase. To this end, the distance between indices along the time axis is computed. If the distance is greater than a certain threshold λ, a new phase is assumed, otherwise the label changes are treated as mislabelings. The ordered set of final cluster label changes c is computed as

$$c = \{p_j | \, (p_j - p_{j-1}) > \lambda, \text{ for } j \text{ in } 2, ..., |p|\}. \tag{8.4}$$

Although the threshold λ has a default value, it can also be adjusted by the operator in case the tools are changed or the data are recorded with a different sampling frequency. In the case of mislabelings, i.e., p_j that are not included in c, the label of the next accepted cluster is assigned. Figure 8.4b shows the resulting *on*/*off* recognition after applying this procedure. The scattered cluster labels are now attached to longer sequences of identical cluster labels.

8.2.1.2 Production Step Identification and Mapping

After identifying the tool changes (i.e., *off* phases) and production steps (i.e., *on* phases) in the raw data, the workflow must map the individual production steps to each other. To accomplish this, each production step is assigned a specific tool with its particular production process. Typically, an anomaly in one machine tool does not affect the production steps of other tools, so the workflow only identifies anomalies for such individual phases. Based on this mapping, the specific phases can be grouped across all manufacturing processes. Figure 8.5 schematically visualizes the workflow of the production step identification and mapping method.

As input, the production step identification and mapping algorithm receives the raw machine signals and the breakpoints between *on* and *off* phases to separate the machine data into segments phase by phase (cf. step (1) in Figure 8.5). Subsequently, the algorithm computes features for each phase (cf. step (2) in Figure 8.5). Preliminary experiments have indicated that the best results were obtained for the measures mean and standard deviation. Finally, the workflow applies hierarchical clustering [RM05] to the feature space in order to group

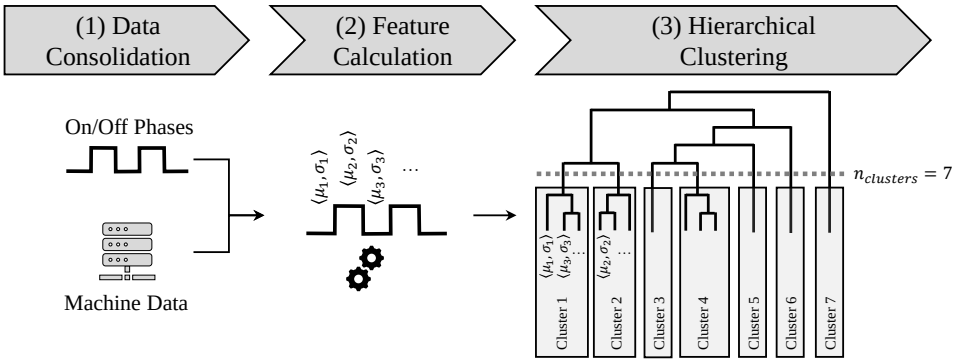


Figure 8.5: The simplified production step identification and mapping using hierarchical clustering.

similar phases to a common identifier (cf. step (3) in Figure 8.5). Hierarchical clustering constructs a tree structure, also referred to as a *dendrogram*. Such a dendrogram either starts with all instances in a cluster and, subsequently, splits each cluster into two clusters until all clusters consist of only one instance (divisive, top-down), or starts with a separate cluster for each instance and gradually merges two clusters into one until only one cluster remains (agglomerative, bottom-up). Here, the workflow applies the bottom-up approach with centroid linking as the agglomeration method. Accordingly, the centroid of each cluster is determined and the distance between centroids a and b of two clusters A and B is computed as $D_{\text{centroid\_linkage}}$:

$$D_{\text{centroid\_linkage}}(A, B) = d(a, b) \tag{8.5}$$

Therefore, the workflow employs the Euclidean distance as distance metric d, where l is the dimension of the feature space:

$$d(a, b) = ||a - b||_2 = \sqrt{\sum_{i=1}^{l} (a_i - b_i)^2} \tag{8.6}$$

However, a dendrogram cutting point is required, which specifies the intended number of clusters. This parameter is the only setting to be made by the operator. However, for a given manufacturing process, the production steps are known beforehand and, therefore, the number of clusters is equal to the number of production steps plus one for the tool change phase.

8.2.2 Machine Learning-based Anomaly Detection Approach

Based on the preprocessed signals, the workflow learns several machine learning models to detect the effects of machine anomalies and compare their detection performance. As an alternative, we have also implemented a typical order analysis method, i.e., resampling the signal into the order domain before computing the Fourier transform [UW99]. However, this commonly used frequency analysis method did not provide satisfactory results (cf. Section 9.3.1). Therefore, our end-to-end workflow integrates machine learning methods by applying these algorithms to the individual phases identified by the previously presented workflow steps. This step targets to learn the dependency between anomaly effects and intrinsic properties of the machine signals.

Nevertheless, a common problem in machine learning and artificial intelligence applications is the amount of training data. Training such methods depends heavily on the amount and variety of data, which are the basis for good generalizability and predictive power of the resulting model. Therefore, the workflow includes a method for enlarging the training data set and, subsequently, applying the machine learning methods. In the following paragraphs, these steps are described in more detail.

The workflow uses vibration signals to detect machine anomalies. Exemplary monitored signals of a three-axis vibration sensor are presented in Figure 8.6. As mentioned in Section 8.1, the implemented vibration sensor provides a comparatively high sampling rate of 2 kHz. This high frequency of the vibration signals allows resampling of the measurements. Accordingly, the original vibration time series of each phase are split into r resampled time series. Each new time series ts_i starts at the i-th position, $i = 1, \ldots, r$, of the original vibration

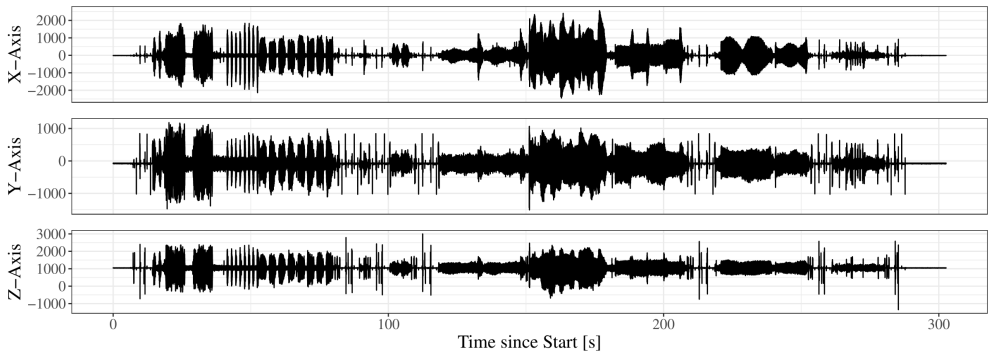


Figure 8.6: An example of the vibration signals provided by the vibration sensor.

time series. After adding a value v_i of the original vibration time series to the resampled vibration time series, $r-1$ values are skipped until the next value is added. This resampling strategy can be described as follows:

$$ts_i = (v_i, v_{i+r}, v_{i+2r}, \dots) \quad , \quad i = 1, \dots, r \tag{8.7}$$

Subsequently, these resampled vibration time series of individual phases are used to compute several statistical characteristics that describe the intrinsic behavior of the time series. For this purpose, the following characteristics are selected, where x denotes the time series and n the length of the time series:

- Mean: Represents a base level around which the vibration signal varies.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{8.8}$$

- Median: Similar to the mean, but less sensitive to outliers. A large difference between the mean and the median indicates high outliers.

$$\tilde{x} = \frac{1}{2} \left(x_{\lfloor (n+1)/2 \rfloor} + x_{\lceil (n+1)/2 \rceil} \right) \tag{8.9}$$

- Standard deviation: Describes the extent of variation within the time series.

$$\tilde{s} = \sqrt{\frac{1}{(n-1)} \sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{8.10}$$

- Skewness: Measures the asymmetry of a probability distribution function around its mean value.

$$\tilde{\mu}_3 = \frac{m_3}{m_2^{3/2}} \quad \text{with} \quad m_k = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^k \tag{8.11}$$

- Kurtosis: Similar to skewness, kurtosis describes the shape of the probability distribution function. Instead of asymmetry, the kurtosis quantifies the steepness of the probability distribution function.

$$\tilde{\mu}_4 = \frac{m_4}{m_2^{4/2}} \tag{8.12}$$

- Root mean square (RMS): The square root of the mean of the squared values of the measured signal is directly related to the energy content of the vibration.

$$\text{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{8.13}$$

- Crest factor: Measures the extremeness of peaks in a waveform. A large value indicates high peaks compared with the RMS of the waveform.

$$C = \frac{\max\left(|x|\right)}{\text{RMS}} \tag{8.14}$$

- Gradient: β_1 represents the slope of the measured signals. For this purpose, a linear model $f\left(t_i; \beta_0; \beta_1\right)$ of the measured signals is fitted to the measurement time t_i.

$$\beta_1 \quad \text{with} \quad f\left(t_i; \beta_0; \beta_1\right) = \beta_0 + \beta_1 t_i \tag{8.15}$$

- Peak to peak (PTP): Specifies the total range between the minimum and maximum values.

$$\text{PTP} = \max\left(x\right) - \min\left(x\right) \tag{8.16}$$

This characteristics computation step provides a feature vector for each resampled time series of each phase. Coupled with a class label indicating whether or not an anomaly is present, such a feature vector represents a single instance used to train the machine learning models. As the machine learning methods employed are supervised, the entire workflow requires labeled training data. For the evaluation of the approach, the class label is supposed to be predicted by the machine learning models. Therefore, only the feature vectors without labels are passed to the method. The predicted class labels are then compared with the actual class labels to assess the anomaly detection quality. As machine learning methods, the proposed approach utilizes Random Forest [Bre01], Support Vector Machine [CV95], and XGBoost [CG16], while a simple logistic regression model serves as the basis of comparison.

8.3 Summary and Discussion

The main contribution of this chapter is the introduction of an automated end-to-end workflow for anomaly detection in industrial machine tools, addressing

research question **RQ B.1**. The key components of the approach are data segmentation, mapping, and classification. First, the raw machine signals are divided into manufacturing and tool change phases using k-Means clustering. Then, similar manufacturing phases are mapped to common identifiers, while different manufacturing phases are mapped to other identifiers via hierarchical clustering. After these preprocessing steps, each manufacturing phase is analyzed individually. Thereby, the original signals are resampled to expand the size of the training set. Subsequently, statistical characteristics are computed for each time series, which are then used to learn a classification model that distinguishes between normal and anomalous machine tool states. Due to resampling, these models can be learned with a small number of measurement runs for training. Based on the automated end-to-end design, the operator only needs to specify the number of manufacturing steps performed by the machine.

Unlike many existing approaches that involve expensive sensors and sophisticated expert knowledge, the presented approach relies only on standard industrial monitoring data, namely position and velocity of the axes, torque and power draw from the DC intermediate circuit, temperature of the motors, and machine vibration. Therefore, the workflow is generalizable so that it can be easily adopted for other machines. However, due to the analysis of vibration data, the approach is specifically tailored to rotating machines.

Chapter 9

Evaluation of the End-to-End Workflow for Automated Anomaly Detection of Industrial Machine Components

In order to assess the performance of the proposed automated end-to-end anomaly detection workflow, we carried out a real-world case study. First, Section 9.1 describes the experimental setup. Then, Section 9.2 shows the results obtained from the phase detection component of the workflow, followed by Section 9.3, which provides the evaluation results of the anomaly detection component. Subsequently, Section 9.4 summarizes the main evaluation findings and discusses potential threats to validity. Finally, Section 9.5 draws a brief conclusion of this chapter. The contents of this chapter are based on our previous work, which has been accepted and is currently under publication in the journal ISA (International Society of Automation) Transactions [ZML+21].

9.1 Experimental Setup

For emulating a faulty tool, we have attached a small weight to a drill in radial direction. It does not interfere with the drilling process, but as an unbalance in the rotation of the tool, it causes additional vibrations in the entire machine. At higher rotational speeds, this caused an audible oscillation, which differs from the usual operating noise of the machine. Unbalance effects cause increased wear and defects, so they should be avoided. While sophisticated systems for detecting tool wear and breakage are available on the market, the expense and effort required to deploy them are usually high. The systems must be integrated into the machine and require detailed process information. For this reason and for the simple nature of the emulation, an unbalance is a suitable example defect that allows us to investigate a loosely coupled detection system with little involved process knowledge.

As for the manufacturing process, an NC program was prepared to mill a 5 mm deep shape into the surface of an aluminum block. This process was chosen because it can be easily repeated without the need for large amounts of material by simply lowering the zero reference by 5 mm in between runs. The prepared faulty tool is used in two drilling operations (i.e., phases P3 and P5) with two different rotational speeds. In total, 5 different tools are used in 6 manufacturing steps. During these phases, which are referred to as *on* phases, the main spindle rotates. In between these *on* phases, automatic tool changes are performed. These are referred to as *off* phases. The *on* phases in which the tool with the prepared unbalance is active are referred to as unbalance phases. Table 9.1 contains an overview of the phases in the recorded NC program.

Table 9.1: Overview over the phases in the recorded NC program. The tool with unbalance is used in phases 3 and 5.

Phase	Description
-	Initial tool pickup: Cutter head ∅ 63 mm
P1	Surface milling, circular deepening, rectangular deepening (3200 min^{-1})
-	Tool change: Insert drill ∅ 24 mm
P2	Drill start bore for milling circular pocket (3000 min^{-1})
-	Tool change: Slot mill ∅ 20 mm
P3	**Widening start bore (2800 min^{-1})**
-	Tool change: Roughing cutter ∅ 32 mm
P4	Pre-milling circular pocket, roughing rectangular deepening (2800 min^{-1})
-	Tool change: Slot mill ∅ 20 mm
P5	**Roughing circular pocket (4500 min^{-1})**
-	Tool change: HSC end mill ∅ 20 mm
P6	Contour edges (17000 min^{-1})
-	Return last tool to magazine

Two aluminum blocks were used for the experiments carried out here, allowing a total of 30 runs of the described machining process to be recorded. Of these, 13 were with an attached unbalance, as described above.

9.2 Accuracy of the Phase Detection Component

Here, the accuracy of the phase segmentation (i.e., the *on*/*off* phase recognition) and the production step identification and mapping are evaluated. The *on*/*off* phase recognition algorithm is evaluated by comparing the timestamps of the phase edges provided by our algorithm with the timestamps captured

in the NC data. We record the NC data only as a gold standard to evaluate the accuracy of the *on*/*off* recognition algorithm, as narrowing the proposed workflow to scenarios where NC data are available would limit the applicability of the approach (cf. Section 8.1). The parameter for threshold-based smoothing was set to 100 ms. This parameter was determined empirically to provide a meaningful distinction between *on* and *off* phases.

Table 9.2 presents the results in terms of time differences. Each column represents a change between two phases, with a phase change ID with an asterisk symbolizing that the phase has changed from an *on* phase to an *off* phase. In contrast, phase change IDs without an asterisk denote phase changes from *off* phases to *on* phases. Table 9.2 reports the mean time difference, the standard deviation of the difference, and the interquartile range of the difference for each of these phase change IDs. All units are presented in milliseconds. Here, a positive mean time difference means that the time computed by the algorithm is later than the time from the NC data, while a negative value indicates the opposite. For presentation reasons, the table has been split into two smaller tables, which are displayed one below the other.

Regarding the mean difference, a clear distinction between phase changes from *on* to *off* phases and phase changes from *off* to *on* phases is evident. Phase changes from *off* to *on* phases range from about 300 ms to 360 ms. In contrast, phase changes from *on* to *off* phases show a mean difference of only about 65 ms to 140 ms, except for the last phase change, which exhibits an average time difference of almost 550 ms. These deviations are not sufficient for a clear

Table 9.2: Mean μ [ms], standard deviation σ [ms], and interquartile range (IQR) [ms] of the deviation between the timestamps of the predicted phase change and recorded phase change. A * indicates that the phase changed from a manufacturing step to a tool change phase.

ID	CH1	CH2*	CH3	CH4*	CH5	CH6*
μ	356.957	98.957	313.913	78.087	301.348	66.870
σ	18.190	16.339	1.041	13.588	0.982	8.081
IQR	35.5	32.5	2.0	26.0	1.0	14.5

ID	CH7	CH8*	CH9	CH10*	CH11	CH12*
μ	301.000	112.522	301.217	137.174	305.696	-548.826
σ	1.243	14.235	1.043	10.836	0.974	11.472
IQR	2.0	27.5	1.0	16.5	1.0	22.0

distinction between different phases. However, both the standard deviation and the interquartile range show that the variation between runs is very small. In contrast to the mean values, the standard deviation and interquartile range show smaller values for phase changes from *off* to *on* phases. The interquartile range is of particular interest because it represents that across all phase changes, half of all values fall within a range of at most 35.5 ms. For 5 of the 12 phase changes, this limit is even significantly smaller, reaching a maximum of 2 ms. This demonstrates that the proposed *on*/*off* recognition algorithm identifies the timestamps very well. Nevertheless, the identified timestamps differ by a relatively fixed offset compared with the timestamps of the NC data. However, this fixed offset can be determined during a calibration phase and, therefore, does not affect the quality of the *on*/*off* recognition. Furthermore, the offset can be explained by the fact that the NC data may use a different point in time as a phase change than the algorithm.

The hierarchical clustering-based production step identification and mapping algorithm, which is executed on the detected *on* and *off* phases, even obtained a perfect matching with an accuracy of 100%.

9.3 Prediction Quality of the Anomaly Detection Component

For identifying the degradation effects caused by the attached unbalance, we employed two approaches. First, we used an acoustic analysis method to investigate the vibrations. However, since the results were not convincing, we applied a second approach using machine learning methods. In the following, we first discuss the results of the order analysis in Section 9.3.1, followed by the results of the machine learning methods in Section 9.3.2.

9.3.1 Acoustic Analysis

Unbalance, as an example of a machine tool anomaly, often produces audible effects, which we also experienced during data collection for our experiments. Therefore, the use of acoustical methods seems natural. Thus, our first attempt was to apply a basic order spectrum analysis to find a discrimination between good and anomalous measurements.

The occurring frequencies in the signal strongly depend on the rotational speed of the main spindle of the machine under investigation. A more general measure for varying rotational speeds than the frequency spectrum is the order analysis [UW99]. Here, the energy of the signal is displayed over the orders and not the frequency. The orders are multiples of the rotational frequency and

provide a normalization for varying rotational speeds. While order analysis is often applied to varying rotational speeds in order to obtain a spectrogram over an acceleration ramp, it can still be helpful to normalize various constant rotational speeds. The transformation can be simplified in such constant cases. In our case study, the spindle is accelerated only once in each phase, i.e., after picking up the tool, and then maintains the programmed speed. The speed variation is negligible, with a deviation of only about ± 5 min^{-1}, so the assumption of a stationary case is valid.

To this end, we calculate the fast Fourier transform (FFT) and apply Welch's method [Wel67] to cancel out noise effects. We employ a window size of 1000 samples, which corresponds to 0.5 s, with an overlap of 50% and apply the Hanning window function. The frequency values are transformed to orders via division by the fundamental order f_0, which is the average rotational frequency of the spindle for that segment.

In the case of an unbalance, we expect the effect to be visible in the spectrum. In particular, we expect the energy of the first order to be significantly higher [Wir98]. Figure 9.1 shows an overlay of all spectra for the measurements with and without unbalance for the second unbalance phase (i.e., phase P5). The comparison shows that the shapes of the curves with and without unbalance are similar. While there is no pronounced difference in the first order, it is noticeable that the overall level is slightly increased. In contrast, the first unbalance phase (i.e., phase P3) exhibits no difference at all and is therefore not presented here.

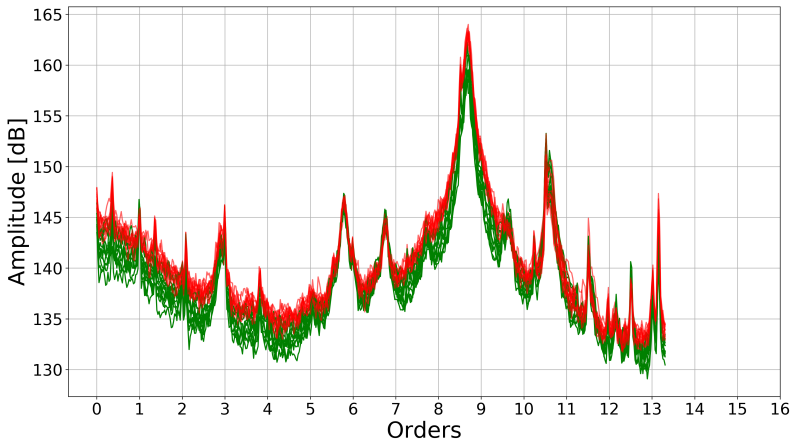


Figure 9.1: Order spectra of all measurements in X direction for the second unbalance phase P5. Measurements of the experiments with attached unbalance are plotted in red, measurements of the normal experiments are plotted in green.

Differences between the two unbalance phases could be attributed to the higher rotational speed in the second unbalance phase, as we generally expect higher excitation levels and, therefore, a more significant effect of the unbalance. The lack of an increased first order might indicate that the unbalance, although having an audible effect, does not affect the structure-borne sound as much. While a slight effect is visible in the plot, it is difficult to derive criteria that clearly separate the two classes.

Based on this result, it can be seen that order analysis as a stand-alone classifier or as a preprocessing step is not as sensitive to the anomaly as expected. For various anomaly effects, it might be even less effective. We conclude that the acoustic approach is not suitable for our simplified means of data collection and that a more generic approach would improve the applicability.

9.3.2 Comparison of Machine Learning Methods

In order to augment the data set, we resample the original vibration signals to a lower sampling rate (cf. Section 8.2.2). For our experiments, we set the resampling factor r to 4. Therefore, the actual sampling rate of the vibration sensor is reduced from 2 kHz to 500 Hz. Given that we focus on automatic anomaly detection, the machine learning task is a binary classification. Thus, we chose accuracy, precision, recall, F1-score, and Matthew's correlation coefficient (MCC) as evaluation measures. For more details on these evaluation measures, please refer to Chapter 2.6.2.1.

We split the data into a training and a testing set to investigate the predictive power of the learned machine learning models. For this purpose, we used 70% of the data for model learning and the remaining 30% of the data for validating the respective model performance. We are aware that splitting the data into training and testing sets may result in an arbitrary ranking of the machine learning models. Therefore, we conducted the experiment 100 times with random splits between the training and testing data sets. However, we ensured that resampled signals originating from the same base signal were used for either training or testing. Otherwise, the results would be biased since the resampled vibration signals are very similar to the original signal. As introduced in Section 8.2.2, we applied logistic regression (LogReg), Random Forest (RF), Support Vector Machine (SVM), and XGBoost (XGB) to the data in order to detect the unbalance. The parametrizations used for the algorithms as well as the `R` libraries are given in Table 9.3.

The results for the first unbalance phase (i.e., phase P3) are presented in Figure 9.2. As we conducted 100 experiments per machine learning method, the figure depicts box plots of the distributions of the obtained scores. The

Table 9.3: Parametrization and used R libraries for the machine learning methods.

Method	Library	Parameters
LogReg	`stats::glm`	family = binomial(link = "logit")
RF	`randomForest` [BCLW18]	ntree = 500, mtry = 10
SVM	`e1071` [MDH+19]	kernel = "polynomial", degree = 3, coef0 = 0, cost = 5, scale = FALSE, epsilon = 0.1, shrinking = TRUE
XGB	`xgboost` [CH19]	objective = "binary:logistic", booster = "gbtree", gamma = 0.5, eta = 0.1, max_depth = 10, nrounds = 1000, subsample = 0.75, colsample_bytree = 1, eval_metric = "error", eval_metric = "logloss", watchlist = list(train, val)

horizontal axis represents the different evaluation measures, while the left vertical axis shows the obtained scores for the measures accuracy, precision, recall, and F1-score. On the contrary, the right vertical axis is used for Matthew's correlation coefficient since this measure ranges from -1 to 1 instead of 0 to 1. Analogous to the other four measures, a larger value of MCC indicates a better prediction. Accordingly, an MCC of 1 corresponds to a perfect prediction, while -1 represents the worst possible prediction. Each of the four different colors represents one of the machine learning methods, i.e., logistic regression (orange), Random Forest (green), SVM (blue), and XGBoost (purple). Furthermore, the brightness of the color denotes whether the resampling method was applied or not. Here, lighter shades represent $r = 1$ and, thus, that the resampling method was not used, while darker shades represent the results obtained with a resampling factor of $r = 4$.

Overall, the medians of accuracy, precision, recall, and F1-score range from about 0.45 up to about 0.80 (left y-axis). Concerning Matthew's correlation coefficient, the medians range from 0.00 to 0.50 (right y-axis). However, the SVM and the logistic regression without resampling obtained rather arbitrary results. Their obtained values are close to random guess (0.50 for accuracy, precision, recall, and F1-score and 0.00 for Matthew's correlation coefficient). Random Forest and XGBoost without resampling, by contrast, already produced promis-

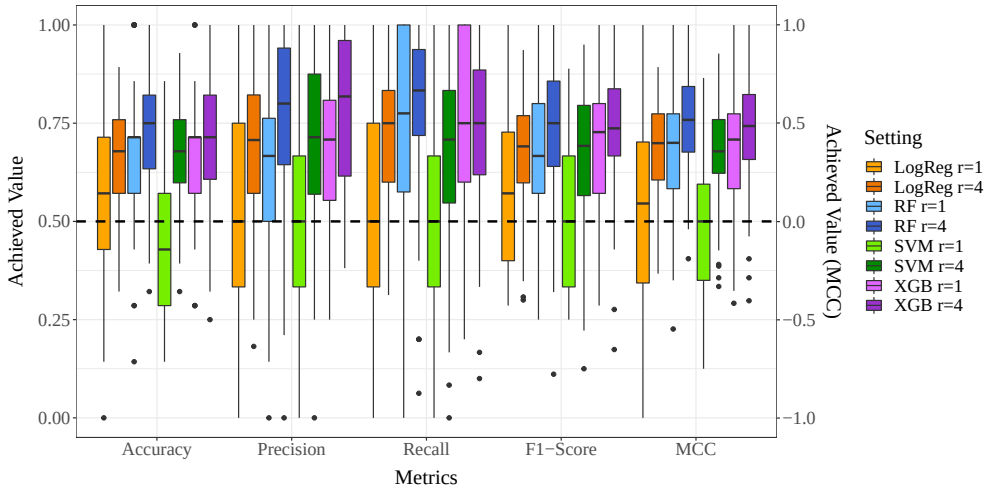


Figure 9.2: Achieved values of the machine learning methods for each evaluation measure for the first unbalance phase (i.e., phase P3). The dashed horizontal line shows the theoretical baseline of the application of random guess.

ing results that were significantly dissimilar to the random guesses. First, this proves that unbalance detection is possible. Second, Figure 9.2 demonstrates that applying the resampling approach with a factor of $r = 4$ improved the performance of all machine learning methods for all evaluation measures. This is evident from the fact that both the medians and the 25th percentiles (i.e., the lower end of the box) of the approaches with resampling are always higher than those of the respective approaches without resampling. Regarding the 75th percentiles (i.e., the upper end of the box), resampling outperforms the original signal in 18 out of 20 cases. Only for recall, a higher third quartile was achieved when applying Random Forest and XGBoost without resampling. With the application of resampling, the SVM and logistic regression also produced acceptable results, which are clearly distinguishable from simple, random guesses. Nevertheless, they did not reach the performance of Random Forest and XGBoost with resampling.

For a numerical comparison of the anomaly detection quality of the machine learning methods, Table 9.4 aggregates the average values obtained for each setting and each evaluation measure. Again, the results show that resampling the vibration signal improved the performance of all machine learning methods for all evaluation aspects. In addition, Random Forest outperformed the other machine learning methods with respect to accuracy, recall, F1-score, and Matthew's correlation coefficient. Only in terms of precision, XGBoost achieved

Table 9.4: Mean μ of the machine learning measures for resampled ($r = 4$) and non-resampled ($r = 1$) signals of the first unbalance phase (i.e., phase P3). The best values of each measure are printed in bold.

Model	**RF**		**SVM**		**LogReg**		**XGB**	
Res. Factor	$r = 4$	$r = 1$	$r = 4$	$r = 1$	$r = 4$	$r = 1$	$r = 4$	$r = 1$
Accuracy [μ]	**0.725**	0.661	0.669	0.461	0.671	0.529	0.717	0.667
Precision [μ]	0.761	0.665	0.705	0.523	0.688	0.532	**0.781**	0.702
Recall [μ]	**0.786**	0.769	0.674	0.482	0.709	0.534	0.752	0.762
F1-Score [μ]	**0.736**	0.677	0.670	0.505	0.672	0.569	0.735	0.693
MCC [μ]	**0.505**	0.393	0.352	-0.042	0.360	0.069	0.464	0.363

a higher value than Random Forest. SVM and logistic regression could not keep up with Random Forest and XGBoost in this experiment.

In addition to the average anomaly detection performance, the robustness of the learned models is also of great importance. For this purpose, the variation of the predictive power over the randomized experiments is considered. The advantage of more robust models is that the prediction quality can be trusted more, while less robust models show increased uncertainty. Table 9.5 contains the standard deviations of the anomaly detection performance across all 100 experiments for each machine learning method. Here, logistic regression with resampling achieved the lowest standard deviation with respect to accuracy, recall, F1-score, and MCC. Only with respect to precision, XGBoost with resampling achieved a slightly smaller value. However, as illustrated in Table 9.4, the overall predictive power of logistic regression cannot keep up with Random Forest and XGBoost in our use case. Moreover, the standard deviations of Ran-

Table 9.5: Standard deviation σ of the machine learning measures for resampled ($r = 4$) and non-resampled ($r = 1$) signals of the first unbalance phase (i.e., phase P3). The best values of each measure are printed in bold.

Model	**RF**		**SVM**		**LogReg**		**XGB**	
Res. Factor	$r = 4$	$r = 1$	$r = 4$	$r = 1$	$r = 4$	$r = 1$	$r = 4$	$r = 1$
Accuracy [σ]	0.153	0.173	0.130	0.178	**0.126**	0.196	0.147	0.152
Precision [σ]	0.221	0.226	0.212	0.264	0.192	0.314	**0.189**	0.200
Recall [σ]	0.205	0.262	0.217	0.260	**0.182**	0.299	0.195	0.239
F1-Score [σ]	0.167	0.179	0.166	0.169	**0.143**	0.181	0.147	0.165
MCC [σ]	0.253	0.325	0.251	0.364	**0.231**	0.417	0.276	0.313

dom Forest and XGBoost using resampling are only slightly larger than that of logistic regression with resampling. Furthermore, the table reveals that resampling reduced the standard deviation for all machine learning methods and evaluation measures. In some cases, such as the MCC for all machine learning methods and precision for logistic regression, the reduction is even substantial. Therefore, it can be concluded that the proposed resampling method improved the results for the first unbalance phase on average and made the machine learning models more robust.

Figure 9.3 presents the obtained machine learning measures for the second unbalance phase (i.e., phase P5). The layout and color-coding of the figure are identical to Figure 9.2. However, compared with Figure 9.2, the anomaly detection performance is much better and in most cases, the 75th percentile even reaches 1.00. Similar to the first unbalance phase, the predictive power of logistic regression learned without resampling scored by far the worst. Nevertheless, the SVM was able to keep up with Random Forest and XGBoost in this unbalance phase. Again, employing the resampling approach always performed better or at least as well as the approach that derived the characteristics merely on the original vibration data. Only in a single case, namely the 25th percentile of the F1-score for XGBoost, the original vibration data achieved a higher value than the resampling approach. The reason for the considerably improved unbalance detection is most likely the increased rotational speed. In

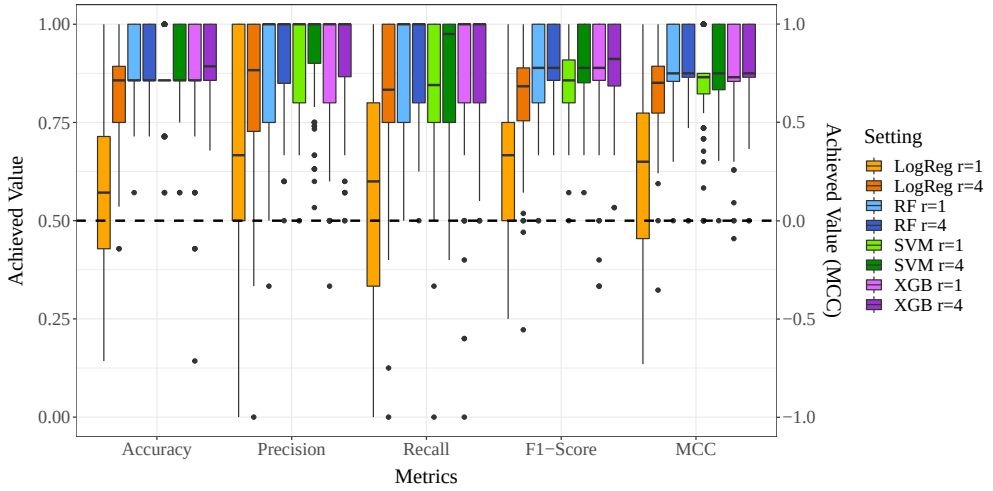


Figure 9.3: Achieved values of the machine learning methods for each evaluation measure for the second unbalance phase (i.e., phase P5). The dashed horizontal line shows the theoretical baseline of the application of random guess.

contrast to unbalance phase P3, which is presented in Figure 9.2, the rotational speed was increased from 2800 min^{-1} to 3500 min^{-1}.

Table 9.6 reports the mean values obtained for each evaluation measure by all machine learning methods with and without resampling. Again, the proposed resampling method significantly increased the average anomaly detection quality for all settings. As with unbalance phase P3, Random Forest with resampling outperformed the other machine learning methods on all measures except precision. However, for this second unbalance phase, SVM with resampling achieved the highest average precision. Nevertheless, XGBoost with resampling was only slightly worse than Random Forest with resampling on all evaluation measures. In contrast to the first unbalance phase, SVM also performed competitively and was able to keep up with Random Forest and XGBoost. Only logistic regression could not keep up with the other methods.

Table 9.6: Mean μ of the machine learning measures for resampled $(r = 4)$ and non-resampled $(r = 1)$ signals of the second unbalance phase (i.e., phase P5). The best values of each measure are printed in bold.

Model	**RF**		**SVM**		**LogReg**		**XGB**	
Res. Factor	$r = 4$	$r = 1$	$r = 4$	$r = 1$	$r = 4$	$r = 1$	$r = 4$	$r = 1$
Accuracy $[\mu]$	**0.908**	0.879	0.890	0.860	0.817	0.593	0.907	0.857
Precision $[\mu]$	0.926	0.875	**0.933**	0.908	0.834	0.623	0.921	0.889
Recall $[\mu]$	**0.917**	0.908	0.879	0.845	0.827	0.586	0.910	0.884
F1-Score $[\mu]$	**0.909**	0.875	0.891	0.860	0.813	0.609	0.902	0.869
MCC $[\mu]$	**0.809**	0.770	0.776	0.699	0.649	0.214	0.805	0.739

The standard deviations for all settings in the second unbalance phase P5 are reported in Table 9.7. Similar to the first unbalance phase P3, the resampling procedure increased the robustness of the machine learning methods, as demonstrated by the reduction in the standard deviation for all settings. However, unlike the first unbalance phase, logistic regression with resampling no longer achieved the lowest standard deviations. More specifically, compared with the other three machine learning methods with resampling, logistic regression performed considerably worse regarding the standard deviation for all evaluation measures. Random Forest achieved the lowest standard deviation for the measures accuracy, recall, F1-score, and MCC. However, SVM reached the same standard deviation for MCC and even outperformed Random Forest with respect to precision. Compared with the first unbalance phase, the standard deviations for all methods except logistic regression have decreased for all evaluation measures.

Table 9.7: Standard deviation σ of the machine learning measures for resampled ($r = 4$) and non-resampled ($r = 1$) signals of the second unbalance phase (i.e., phase P5). The best values of each measure are printed in bold.

Model	**RF**		**SVM**		**LogReg**		**XGB**	
Res. Factor	$r = 4$	$r = 1$	$r = 4$	$r = 1$	$r = 4$	$r = 1$	$r = 4$	$r = 1$
Accuracy $[\sigma]$	**0.086**	0.106	0.092	0.099	0.115	0.216	0.098	0.151
Precision $[\sigma]$	0.130	0.170	**0.119**	0.138	0.196	0.300	0.143	0.165
Recall $[\sigma]$	**0.123**	0.141	0.147	0.186	0.185	0.311	0.144	0.198
F1-Score $[\sigma]$	**0.088**	0.121	0.097	0.103	0.130	0.197	0.112	0.143
MCC $[\sigma]$	**0.196**	0.207	**0.196**	0.236	0.224	0.428	0.215	0.249

As a summary, the average anomaly detection performance increased from the first to the second unbalance phase and the standard deviation decreased. This behavior could occur due to the fact that the rotational speed increased from 2800 min^{-1} (first unbalance phase P3) to 3500 min^{-1} (second unbalance phase P5). Moreover, the proposed resampling technique increased the predictive power and considerably reduced the variation in predictive power. The best overall results were obtained by Random Forest with resampling, as it achieved the best average anomaly detection power in 8 out of 10 cases and showed the smallest variation in anomaly detection power in 4 out of 10 cases. Finally, the experiment demonstrated that the proposed workflow is able to accurately detect anomalies in machine tools when proper resampling is applied.

9.4 Discussion

As a recapitulation of the most important results, Section 9.4.1 briefly outlines the evaluation results. Although our goal was to introduce an optimal and universally applicable workflow, we are aware of certain limitations. These are discussed in Section 9.4.2.

9.4.1 Summary of Evaluation Findings

As a brief summary of the experimental real-world case study, the key findings are as follows:

(I) On the basis of the measured machine-internal data and vibration signals, conventional acoustic analysis cannot detect the anomalies well. This observation is mainly due to the requirements of such methods to receive data with

high sampling rate and signal resolution, which in turn requires expensive sensors. The minor differences were not sufficiently meaningful to act as classifiers without human assessment. However, our goal was to achieve automated detection using standard machine data and inexpensive sensors while keeping the required amount of training experiments low.

(II) The proposed machine learning-based end-to-end workflow for automatic anomaly detection is able to segment the data for each production step and detect the anomalies with an F1-score of up to 90.9%. Moreover, the case study demonstrates that the proposed resampling technique of the original signal considerably improves the quality of the anomaly detection.

(III) The proposed automatic workflow detects anomalies in machine tools with high accuracy even with little training data. This proves that the workflow can be quickly integrated into industrial production processes.

(IV) Random Forest yields the overall best anomaly detection results based on the features and parametrization used in this case study. The application of XGBoost also achieves comparably good results. In contrast, the results for SVM and logistic regression without resampling are close to random guessing. When resampling is used, their performance also increases but still does not reach the anomaly detection quality of Random Forest and XGBoost.

(V) Increasing the rotational speed results in more accurate anomaly detection rates. This is consistent with the physical properties of an unbalance, as higher rotational speeds imply larger resulting forces, which in turn affect the measured signals. Therefore, the type of detection may not be suitable for phases with significantly lower rotational speeds.

9.4.2 Threats to Validity

The goal of this contribution was not to develop a novel machine learning approach but to combine several existing base-level machine learning techniques in an intelligent way to automatize the process of anomaly detection in machine monitoring data, especially for standard machine signals and low-cost sensors. By demonstrating the applicability and effectiveness of such an automated workflow based on comparatively simple machine learning techniques, we aim to bring academic theory and industrial practice closer together.

We are aware that 30 measurement runs only provide a small data set for machine learning models. In practice, however, it is an important criterion that methods can be applied quickly without long measurement and training times. The workflow integrates the resampling technique to augment the training data artificially, whereby it achieves such good performance despite the limited training data. The experimental results show that the automatic

workflow already achieves good performance even with these few training data. However, increasing the amount of training data would most likely improve the models even further.

The main problem of most anomaly detection mechanisms for machine data is reusability and transferability, as they require in-depth domain knowledge and dedicated data that other practitioners cannot acquire. However, the proposed workflow requires only one input from the operator, which is the number of distinct production steps performed by the machine. Furthermore, the proposed automated workflow relies only on standard physical quantities (cf. Section 8.1) that can be measured on any comparable machine, making the approach transferable.

As an alternative to sensors with integrated analog-to-digital conversion, analog input signals were also sampled via the machine's PLC. Although this is possible in principle, it has some major downsides. The PLC code would have to be adapted, which can be a hindrance for existing production machines, if only for liability reasons. Furthermore, sampling the data at a high frequency imposes an additional load on the control. Depending on the use of the machine, e.g., controlling a high number of axes in real time, the computing resources could already be exhausted. Finally, the sampling rate would depend on the cycle time of the PLC. Typically, depending on the model and brand, the fastest possible cycle times are in the range of 1 ms to 10 ms, i.e., 1 kHz to 0.1 kHz. In practical applications, the cycle time is often intentionally relaxed, e.g., to 10 ms or 20 ms, to avoid stressing the system when fast cycles are not needed. Input modules with oversampling capabilities could make sampling more independent of PLC cycle time but require additional hardware. Therefore, using the PLC directly is not optimal for our scenario to acquire additional high-frequency data from external sensors.

While the resolution of the vibration sensor used proved insufficient for detailed acoustic analysis, a more elaborate measurement setup would have provided a better signal and, therefore, possibly better results for the acoustic features. We did not pursue this in favor of a more straightforward solution in terms of measurement and preprocessing. This allows to keep the approach manageable and the cost low while maintaining satisfactory performance.

Although more runs were recorded, only the runs that processed material were used for the experimental results. For runs where the machine processed no material, but the operations were done through air only, the vibration signals as well as some internal machine data, such as torque and DC link power, contain considerably less information, resulting in different data patterns. Therefore, the learned models were not applicable to these runs.

9.5 Concluding Remarks

This chapter provides results of the automated end-to-end workflow for anomaly detection of machine tools in a real-world case study. Thus, it contributes to research question **RQ B.1** by demonstrating the quality and effectiveness of the proposed approach. To this end, we emulated machine anomalies by attaching a small weight to a drill of a CNC machine in radial direction. The results revealed that the end-to-end workflow accurately segmented the raw machine signals of entire manufacturing processes into individual production and tool change phases. Moreover, by applying hierarchical clustering, similar production phases were perfectly mapped to common identifiers. Additionally, we compared conventional acoustic analysis with the machine learning models integrated into the proposed workflow to detect anomalies in machine tools. Here, we have shown that the conventional approach could not distinguish between phases with and without anomalies. In contrast, Random Forest provided the best overall anomaly detection quality with an F1-score of up to 90.9%. Finally, we demonstrated the benefits of resampling the original vibration data to augment the training data. This procedure not only improved the average quality of anomaly detection, but also the robustness and, consequently, the trustworthiness of the models.

However, there are also two noteworthy limitations of the case study conducted. First, only one machine and one type of anomaly was investigated. In the future, the applicability of the presented workflow should be studied for further machines and anomalies. The second limitation is that the results revealed that the higher the speed, the better the anomaly detection quality. In turn, this suggests that the proposed workflow may not provide satisfactory results for rotating machines with too low rotational speeds.

Chapter 10

Comparison of Modeling Alternatives for Time-to-Failure Prediction

Chapter 8 presented a novel approach for automated end-to-end detection of anomalies in machine tools. However, merely detecting current anomaly patterns does not provide enough information for many applications. Instead, the time-to-failure, sometimes also referred to as remaining useful lifetime, is desirable because it provides additional information on the estimated time of the breakdown. Thus, it allows for more detailed planning and scheduling of countermeasures. However, with respect to machine learning-based time-to-failure prediction, several modeling alternatives are conceivable. Therefore, this chapter focuses on four main time-to-failure prediction modeling alternatives on the use case of hard disk drives. However, to first introduce important fundamentals, Section 10.1 provides general information on hard disk drive monitoring. Given the fact that time-to-failure data is generally highly imbalanced, we then present three binary classification alternatives with different oversampling strategies in Section 10.2. Subsequently, we model the time-to-failure using multi-class classification instead of binary classes in Section 10.3. Therefore, a new labeling strategy is used to generate meaningful target values. In Section 10.4, time series forecasting is incorporated to estimate the development of the features to improve both binary and multi-class prediction. As a final time-to-failure modeling alternative, we employ regression to predict the time-to-failure as a continuous value, rather than considering only fixed time windows in the form of classes (cf. Section 10.5). Finally, we briefly summarize this chapter by answering the respective research questions in Section 10.6.

Parts of the content of this chapter are based on our previous work published as a full paper as part of the 20th International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB) [ZKE+20].

10.1 Introduction to Hard Disk Drive Monitoring

The increasing size of today's data centers along with the expectation of 24/7 availability is continuously increasing the complexity of managing hardware, as large IT companies such as Google, Amazon, Microsoft, and IBM have millions of servers worldwide. The administration of these servers is therefore becoming an increasingly expensive and time-consuming task. In particular, unexpected crashes of servers, for example due to hard disk drive failures, can cause unavailable services and data loss. For this reason, hardware is usually equipped with monitoring mechanisms to observe the current condition.

A system known as Predictive Failure Analysis for monitoring internal hard disk drive parameters to improve reliability was first implemented in hard disk drives by IBM in 1992 [Sea99]. Compaq, Seagate, Quantum, and Conner integrated another monitoring system called IntelliSafe [OP95] just a few years later. In 1995, Seagate sought a version compatible with other hardware manufacturers. Therefore, IBM, Quantum, Western Digital, and Seagate collaborated to develop a new hard disk drive monitoring system based on IntelliSafe and Predictive Failure Analysis. The result of this collaboration was the Self-Monitoring, Analysis, and Reporting Technology (S.M.A.R.T.) [Sea99]. Nowadays, this technology is implemented as a monitoring system in most hard disk drives (HDDs) and also solid state drives (SSDs). Here, various internal parameters and operations, such as head-flight height, spin up time, and number of drive calibration retries, are stored during runtime. Although most drive manufacturers include S.M.A.R.T. for drive monitoring, each drive manufacturer can define its own set of attributes to monitor and thresholds for these parameters that should not be exceeded. Nevertheless, there is a subset of S.M.A.R.T. attributes that are commonly monitored by most drive manufacturers, such as spin up time, read error rate, and start/stop count. However, S.M.A.R.T. lacks intelligent analysis and linking of the various parameters. In particular, predictive analytics would allow detecting potential failures or crashes in advance, avoid delays caused by unexpected failures, and, thus, improve reliability in cloud computing. Moreover, since it enables warnings before the failure actually occurs, it can support administration, for instance by offering the potential to guarantee the existence of a backup of the data and the availability of a replacement device.

In general, hard disk drive failures can be divided into two types: predictable and unpredictable failures. Predictable failures occur due to slow mechanical processes, such as wearing. Since the S.M.A.R.T. technology employs only static thresholds of individual parameters, it can only detect slow degradation effects caused by predictable mechanical failures. According to Seagate [Sea99],

mechanical and, thus, mostly predictable failures are responsible for about 60% of all failures. Therefore, the detection rate based on S.M.A.R.T. thresholds alone is insufficient. Unpredictable failures, by contrast, tend to occur spontaneously. The reasons for such unpredictable failures are typically of electronic nature or sudden physical failures. Although the mere use of manufacturer thresholds does not cover both types of failures, a study by Google [PWB07] showed that certain S.M.A.R.T. attributes, for instance the number of (offline) uncorrectable sectors and the number of reallocation events, are strongly correlated with hard disk drive failures.

10.2 Binary Classification for HDD Failure Prediction

In order to predict impending failures by means of a binary classification approach, we divide the data into two different classes. For this purpose, all instances with a time-to-failure of one week or less are given class label `1`, while the remaining S.M.A.R.T. data instances are labeled with class label `0`. This labeling procedure was also performed by T. Pitakrat *et al.* [PVHG13]. Although T. Pitakrat *et al.* conducted a broad comparison of different machine learning methods, they did not evaluate the impact of oversampling techniques to balance the number of instances of both classes. To this end, we apply three different ways of training the machine learning model: (I) binary classification without further data preparation (referred to as unmodified), (II) binary classification with Enhanced Structure Preserving Oversampling (ESPO) as oversampling technique, and (III) binary classification with Synthetic Minority Oversampling Technique (SMOTE) as oversampling mechanism. Figure 10.1 schematically illustrates the three different alternatives. After preparing the training data, we apply Random Forest as machine learning technique as it is typically comparatively fast for model learning and prediction, robust against overfitting, and can handle multiple classes efficiently. Although handling multiple classes is not explicitly required for comparing data preparation for binary classification, it is necessary for multi-class classification in Section 10.3 and, therefore, to maintain comparability between the binary and multi-class approaches. Moreover, Random Forest obtained strong and robust results in the comparison of machine learning methods by T. Pitakrat *et al.* [PVHG13].

10.2.1 Unmodified

In the binary classification approach without any further preprocessing steps, we pass the S.M.A.R.T. measurement instances along with their respective

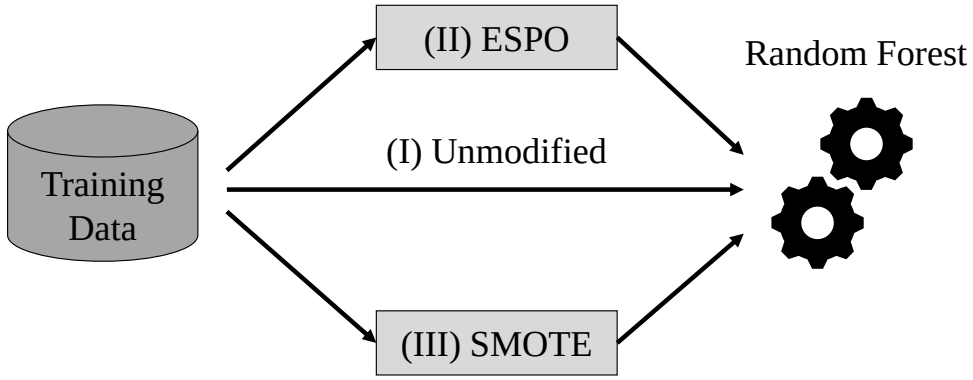


Figure 10.1: Schematic illustration of the three binary classification alternatives.

class label to Random Forest to learn the dependency between the S.M.A.R.T. measurements and the class labels. Upon learning the model, we provide unseen instances to the learned model to predict whether or not the S.M.A.R.T. attributes of the hard disk drive indicate an impending failure.

10.2.2 Enhanced Structure Preserving Oversampling

Dealing with imbalanced data is a non-trivial challenge for machine learning methods. In this context, imbalance refers to training data containing significantly more instances of one class (majority class) than of the other class (minority class). This often yields biased models that tend to overestimate in the direction of the majority class. In the case of hard disk drive failure prediction, the number of instances of the class representing hard disk drives that will fail within the next week is much smaller. There are two common methods to deal with such class imbalances, namely undersampling and oversampling. Undersampling involves dismissing instances of the majority class in order to achieve a balance in the number of class instances. Oversampling, in contrast, creates new instances of the minority class based on existing instances. For this approach, we utilize oversampling instead of undersampling since removing majority class instances would reduce the size of the training data set, which typically reduces the quality of the model[1]. In this second alternative of the binary classification approach, we employ ESPO as oversampling method. ESPO synthetically generates new instances of the minority class based on the multivariate Gaussian distribution [CLWN13]. To this end, ESPO estimates

[1]This does not apply if the data set is still sufficiently large after the undersampling.

the covariance structure of the minority class instances and regularizes the unreliable eigenspectrum [CLWN13]. Finally, ESPO expands the set of minority class instances by maintaining the main covariance structure and generating protection deviations in the trivial eigendimensions [CLWN13]. Once the minority class is oversampled, we follow the same model learning procedure as in alternative (I). However, with this alternative, the machine learning procedure receives more training data and approximately equally distributed training data with respect to the two classes.

10.2.3 Synthetic Minority Oversampling Technique

This alternative is similar to alternative (II), but uses SMOTE instead of ESPO as oversampling technique. SMOTE produces new minority class instances by combining instances that are close to each other in the feature space [CBHK02]. To this end, SMOTE determines the nearest neighbors of each minority class instance. Next, SMOTE picks a random subset of these neighboring instances and computes the differences between their features and the features of the respective instance. The computed feature differences are then weighted by a random number between 0 and 1. Finally, SMOTE adds the resulting weighted difference to the features of the instance under consideration and provides the result as a new minority class instance [CBHK02].

10.3 Classification of Multiple Failure Levels

In order to not only predict whether or not a hard disk drive will fail in the near future, but to more accurately predict the time-to-failure, multi-class classification can be used instead of binary classification.

10.3.1 Failure Level Labeling

For the prediction of upcoming failures in multiple time windows, the target variable must be re-defined. To this end, we have converted the binary failure variable of the original data into a new failure label that includes multiple classes. In the original data, the failure state is `0` if the hard disk drive is running properly, turning to `1` only if the hard disk drive is stopped due to a failure, i.e., the time-to-failure is zero. Given that the goal is to predict multiple failure levels, other class labels are required. Therefore, we define a set of relevant classes, each representing a different time-to-failure window: 0, (0,1], (1,2], (2,5], (5,12], (12,24], (24,48], (48,72], (72,96], (96,120], (120,144], (144,168], (168,∞). Thus, each of these labels represents the interval of the time-to-failure

in hours. The last class label (168,∞) indicates that no failure will occur within the next week. Thus, this class corresponds to class `0` of the binary classification approach. In contrast, the entire set of all other classes corresponds to class `1` of the binary classification approach. For the sake of simplicity and readability, we will refer to each of these time-to-failure classes only by its upper bound in the following. That is, if the time-to-failure prediction model yields label `48`, the failure is expected to occur at the earliest after 24 hours (i.e., after the next smallest class label), but no later than 48 hours. Algorithm 14 illustrates the re-labeling task flow, while Figure 10.2 presents an example of such a re-labeling process. The left-hand side shows the original data with the time-to-failure in the left column, S.M.A.R.T. attributes in the middle columns, and the two failure states in the right column. On the right-hand side, the figure displays the data with the newly created class labels in the right column.

Algorithm 14 Re-labeling

Input: time-to-failure ttf to be re-labeled and breakpoints $label\_breaks$ for different time-to-failure classes

Output: lower bound $lower\_bound$ and upper bound $upper\_bound$ of the respective time-to-failure class

```
### if the failure is currently present, the lower and upper bounds are the same
if (ttf == 0) then
    return [0, 0]
end if

i = 1
while (i < length(label_breaks)) do
    if (label_breaks[i] < ttf) then
        lower_bound = label_breaks[i]
    end if
    if (label_breaks[length(label_breaks) − i)] ≥ ttf) then
        upper_bound = label_breaks[length(label_breaks) − (i − 1)]
    end if
    i = i + 1
end while
return [lower_bound, upper_bound]
```

Time-to-Failure	S.M.A.R.T. Attributes	Failure State
172.634	…	0
170.651	…	0
168.651	…	0
166.651	…	0
…	…	…
97.167	…	0
95.167	…	0
…	…	…
0	…	1

Re-Labeling

Time-to-Failure	S.M.A.R.T. Attributes	Failure Classes
172.634	…	Inf
170.651	…	Inf
168.651	…	Inf
166.651	…	168
…	…	…
97.167	…	120
95.167	…	96
…	…	…
0	…	0

Figure 10.2: An example of the re-labeling technique.

10.3.2 Model Learning

Once the new class labels are computed based on the time-to-failure, the S.M.A.R.T. attributes are passed to the time-to-failure prediction model as features, while the newly created time-to-failure labels are used as target variables. Thus, the model not only learns whether or not the hard disk drive is about to fail, but also learns the time-to-failure in discretized form. Therefore, the binary classification (cf. Section 10.2) is extended to a multi-class scenario, with each class representing a certain time window in which the predicted failure is expected to occur.

10.3.3 Downscaling to the Binary Classification Case

Although the multi-class model is also evaluated for multi-class classification, the predictions of the multi-class time-to-failure prediction approach are also scaled down to the same two classes that are included in the binary approach. This is done because the macro measures of the multi-class classification are not directly comparable to the binary classification measures. For this purpose, the class ∞ of the multi-class approach is equivalent to class `0` of the binary approach. Instead, class `1` of the binary approach is expressed by the totality of all classes except class ∞ of the multi-class approach.

10.4 Integrating Forecasting into the Feature Generation Step

In order to further enhance the time-to-failure prediction quality, we integrate time series forecasting to estimate future S.M.A.R.T. readings. To this end, we interpret the sequence of S.M.A.R.T. measurements for each hard disk drive as a multivariate time series. This multivariate time series is partitioned into univariate time series, with each univariate time series containing the historical observations of a single S.M.A.R.T. feature. Next, the length of each time series is determined. Depending on the length of the time series, either Naïve forecasting or ARIMA is applied to produce a one-step-ahead forecast. Note that we did not choose Telescope because the S.M.A.R.T. time series do not exhibit seasonal patterns. If the S.M.A.R.T. time series has less than 10 values, Naïve forecasting is applied, since such time series are not sufficiently long to learn a more complex model. Otherwise, an ARIMA model is fitted on the time series of S.M.A.R.T. observations. After learning the forecast model, a one-step-ahead forecast is produced for each univariate time series and, thus, for each S.M.A.R.T. feature. These forecast S.M.A.R.T. features are added to the current S.M.A.R.T. features and passed to the time-to-failure prediction model. In order to compare the time-to-failure prediction models presented in Sections 10.2 and 10.3 with this time-to-failure prediction models using additional forecast information, the Random Forest prediction model is learned in both variants, binary and multi-class classification.

10.5 Regression for Time-to-Failure Prediction

The binary classification presented in Section 10.2 only provides predictions of whether or not a hard disk drive is likely to fail within the given horizon. The multi-class classification approach from Section 10.3 further provides a more concrete time window with lower and upper bounds, i.e., the predicted time-to-failure class, within which the failure is expected to occur. However, all these predictions are only discretized.

To obtain continuous predictions of the time-to-failure, regression must be utilized instead of classification, since classification can only predict a discrete output, which must belong to a certain set of classes. In addition, the output class must also be included in the training set so that the model can learn the association between the features and the target class. In contrast, regression can predict any continuous value from the input features. Hence, regression models can predict values that are not present in the training set. To this end, we implement two time-to-failure regression alternatives using Random

Forest. The first approach takes all available data to learn the Random Forest regression model. For this reason, we refer to this approach as *naïve*. The second alternative learns the Random Forest regression model exclusively on those training instances where the failure occurs within the next 168 hours. As we conduct this filtering step before learning the model, we refer to this approach as *pre-filtering*. Next, the Random Forests regression model is applied to only those S.M.A.R.T. measurements where a failure will occur within the defined horizon. To obtain this information, the approaches presented in the Sections 10.2 - 10.4 can be employed, for instance. The concept of pre-filtering the training set to hard disk drives that actually fail within the next week aims to focus the regression model on failing devices. For healthy hard disk drives, a time-to-failure regression is meaningless since, if there is not yet an indicator of impending failure, it is not possible to distinguish whether the hard disk drive will last, for instance, another year or two years. Therefore, the time-to-failure for healthy hard disk drives can only be guessed. By pre-filtering the data set, we explicitly target the regression model to the relevant parts of the data to achieve a more accurate time-to-failure prediction.

10.6 Summary and Discussion

To conclude this chapter, we briefly summarize the main contributions of this chapter by answering the modeling aspects of research question **RQ B.2**, which generally addresses the challenge of how to implement proactive prediction of critical events for technical systems. However, this research question consists of three individual parts, of which this chapter provides answers to the first two, namely research questions **RQ B.2.1** and **RQ B.2.2**. Research question **RQ B.2.1** specifically targets imbalances in the data set. In critical event prediction in general and specifically for technical systems, such critical events, e.g., severe failures causing breakdowns, occur quite infrequently. Therefore, the available training data for model learning typically contain only few instances of critical events, but a large set of instances representing a healthy system state. However, such data imbalances negatively affect the learning process of machine learning models. In fact, the machine learning models already achieve high prediction quality with respect to conventional measures by predicting only the majority class. Although this yields good evaluation measures, the predictions do not contain meaningful information. To remedy this issue, the minority class instances in the training set can be oversampled or the majority class instances in the training set can be undersampled. Here, we use two oversampling strategies, namely Enhanced Structure Preserving Oversampling and Synthetic Minority

Oversampling Technique. The next research question **RQ B.2.2** focuses on how to model the time until the critical event occurs. For this purpose, we presented three alternatives. On the one hand, if it is only necessary to predict the presence of upcoming critical events, a binary classification can be employed, where the positive class represents a critical event within a given time window. On the other hand, if the time-to-failure is required in bins, a multi-class classification should be used, with each class representing a particular time window with a lower and an upper bound. Finally, in case that fine-grained time-to-failure predictions are required, regression should be used. However, in the latter case, we still propose to incorporate classification as a pre-filtering step so that only instances with an impending critical event are passed to the regression model for training and prediction. For instances representing healthy machine states, the prediction of a time-to-failure does not provide meaningful results, but only results in high errors and, thus, confusion. Last, we proposed to integrate time series forecasting for feature generation. For this purpose, the features of successive instances can be considered as time series, to which a forecasting model can be applied. The benefit of including feature forecasts is that deterioration patterns can be detected even earlier, assuming that first indications of deterioration must nevertheless be apparent.

Chapter 11

Evaluation of Time-to-Failure Modeling Alternatives

This chapter provides experimental results for the time-to-failure modeling alternatives presented in Chapter 10. To this end, Section 11.1 first describes the overall evaluation design. In Section 11.2, the oversampling techniques for binary classification are evaluated and compared with using only the original, imbalanced data set. Subsequently, the prediction quality of the multi-class classification approach is assessed in Section 11.3, followed by the regression model in Section 11.4. Section 11.5 provides runtime comparisons of the three time-to-failure modeling alternatives. The evaluation of prediction quality when integrating time series forecasting for feature forecasts is shown in Section 11.6. Then, Section 11.7 summarizes the main evaluation findings and discusses threats to validity. Finally, we conclude this chapter in Section 11.8 by answering the respective research question.

Parts of the content of this chapter are based on our previous work published as a full paper as part of the 20th International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB) [ZKE+20].

11.1 Evaluation Design

In order to assess the performance of the classification and regression models, we use a data set comprising 369 HDDs[1]. This data set was first used by J. Murray *et al.* [MHKD05]. Although the data set includes 64 S.M.A.R.T. features for each measurement instance, we only consider 25 in our experiments. Many of the excluded parameters are constant throughout the entire measurement period and, therefore, do not contain information about the health state of the hard disk drives. The time-to-failure is of course excluded for the classification tasks and used as target variable for the regression task. The remaining 25

[1]HDD data set: `http://dsp.ucsd.edu/~jfmurray/software.htm`

features in the reduced data set are FlyHeight5-12, GList1-3, PList, ReadError1-3, ReadError18-20, Servo1-3, Servo5, Servo7-8, and Servo10. This reduced feature set was also used by T. Pitakrat *et al.* for their comparison of machine learning methods [PVHG13]. Out of the 369 hard disk drives included in the data set, 178 did not fail during the measurement period, while 191 suffered a critical failure. This high number of failed hard disk drives is due to consumers sending in their failed hard disk drives with the S.M.A.R.T. parameters from the previous weeks. As a result, the number of failed and intact hard disk drives in this data set is fairly balanced. However, the data set contains continuous, i.e., approximately two-hourly, measurements of each HDD. This results in a total of 68,411 measurement instances. However, since we not only predict for each hard disk drive whether it will fail one day in the future, but we predict the time-to-failure for each measurement instance of the S.M.A.R.T. parameters, this leads to an imbalanced data set. Figure 11.1 presents a histogram of the distribution of the time-to-failure classes in the considered data set. For this purpose, the time-to-failure classes are plotted on the horizontal axis and the number for each time-to-failure class is plotted on the vertical axis. The figure clearly demonstrates the dominance of the last class, i.e., the class indicating that there will be no failure within the next 168 hours. Furthermore, it can be seen that in particular the time-to-failure classes `0` to `5` are very small with

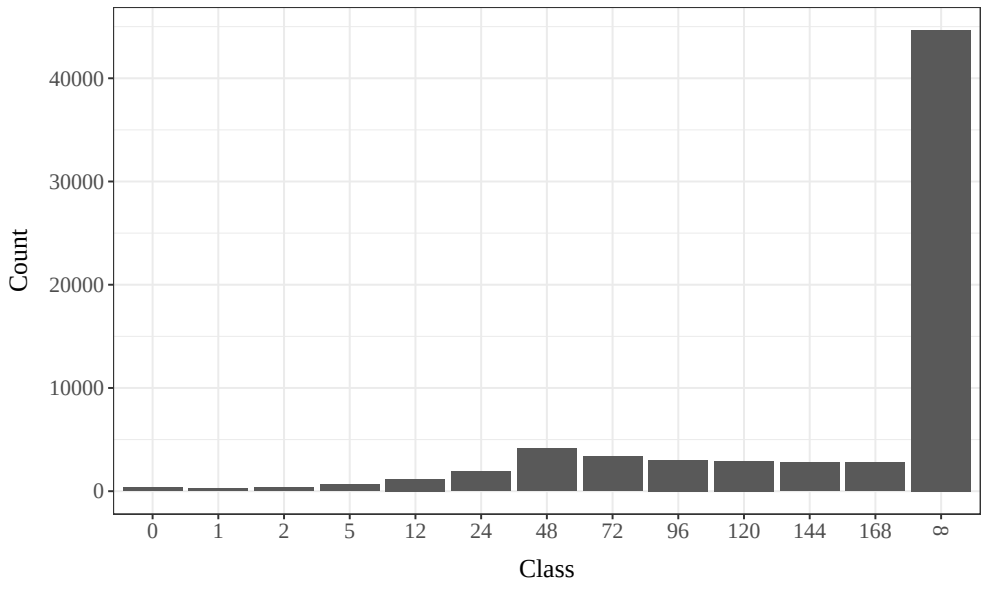


Figure 11.1: Histogram of the distribution of time-to-failure classes.

only 260 to 653 instances. For the binary classification, the class indicating an impending failure within the next 168 hours comprises 23,749 instances, while the class representing no failure within this time consists of 44,662 instances.

For the application of the oversampling methods considered and compared in these experiments, we used the `R` libraries `OSTSC` [DKW17] for ESPO and `unbalanced` [DPCB15] for SMOTE. Moreover, we used the Random Forest implementation provided by the `R` library `randomForest` [BCLW18].

In agreement with the literature, we use a split of approximately 80:20 for model training and model testing. Accordingly, about 80% of all data are used to learn the model, while the model is evaluated on the remaining roughly 20% of the data. As a single experiment alone is not significant, we run the experiment 25 times to reduce random bias, since most of the techniques and methods employed are based on random numbers.

11.2 Binary Failure Prediction

To ensure reproducibility, Table 11.1 specifies the parametrization of Random Forest and the oversampling methods used to compare the binary classification models. We set the number of decision trees constructed to 100 and the number of features randomly selected as candidates for each split to five for all three binary classification alternatives.

Table 11.1: Libraries and methods utilized along with the parametrization for the binary classification models.

Alternative	Library:Method	Parameters
(I) Unmodified	randomForest:randomForest()	ntree = 100, mtry = 5, replace = TRUE
(II) ESPO	OSTSC:OSTSC()	ratio = 1.0, r = 1.0
	randomForest:randomForest()	ntree = 100, mtry = 5, replace = TRUE
(III) SMOTE	unbalanced:ubBalance()	type = "ubSMOTE", percOver = 300, percUnder = 150, k = 5
	randomForest:randomForest()	ntree = 100, mtry = 5, replace = TRUE

Table 11.2 reports the average hard disk drive failure prediction quality achieved with respect to accuracy, precision, recall, and F1-score. The results show that alternative (II) ESPO obtained the highest accuracy, precision, and F1-score. Only in terms of recall, ESPO was outperformed by both the unmodified binary approach and the approach using SMOTE. However, all three approaches differed only slightly with respect to recall. Considering all four evaluation measures, ESPO achieved the best overall performance of the binary classification alternatives.

Table 11.2: Average achieved values of the three binary classification alternatives. The best values for each evaluation measure are highlighted in bold.

Alternative	Accuracy	Precision	Recall	F1-Score
(I) Unmodified	97.472%	94.347%	96.993%	95.649%
(II) ESPO	**97.642%**	**94.913%**	96.970%	**95.928%**
(III) SMOTE	95.734%	89.086%	**96.995%**	92.869%

As Table 11.2 only presents the average values and not the variation within the 25 repetitions, Figure 11.2 visualizes the prediction quality using box plots. For this purpose, the horizontal axis shows the evaluation measures accuracy, preci-

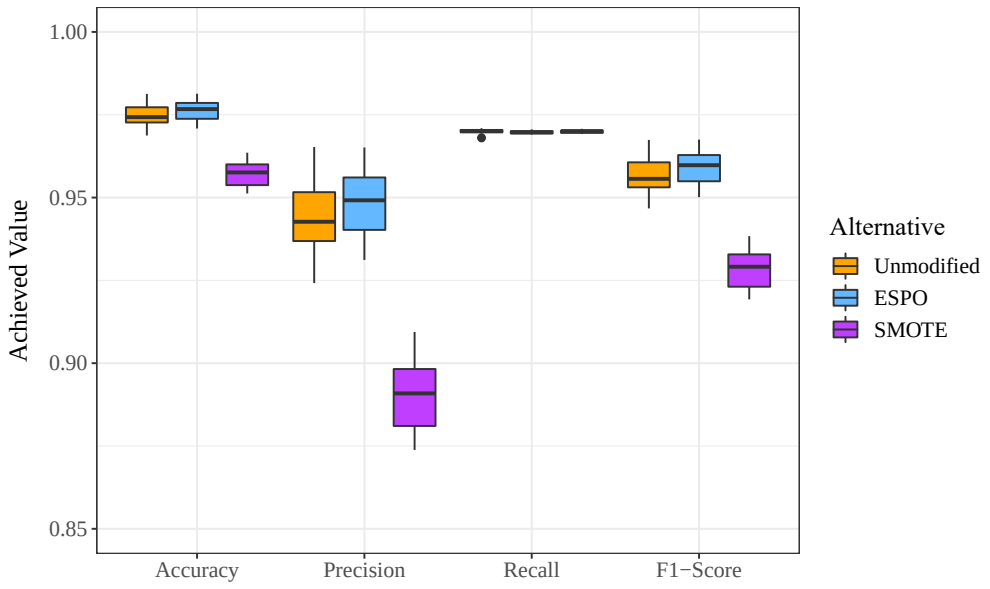


Figure 11.2: Box plots of the prediction quality for the binary classification alternatives.

sion, recall, and F1-score, while the vertical axis shows the achieved values. The orange, blue, and purple boxes represent the approach without oversampling, the ESPO oversampling approach, and the SMOTE oversampling approach, respectively. Again, the figure reveals that the unmodified and ESPO approaches outperformed SMOTE significantly with respect to all measures except recall. Regarding recall, all three approaches provided approximately the same value with only minor variations within repetitions. The highest variation is evident for the precision. Here, the interquartile range exhibited the largest value.

To conclude the comparison of class balancing strategies, employing ESPO oversampling obtained the overall best prediction quality, although the model without oversampling produced only slightly worse results. In contrast, SMOTE oversampling actually degraded the overall prediction quality compared with the model using no oversampling. Thus, oversampling can improve the predictive power of the model, but it depends on the particular technique applied.

11.3 Failure Level Classification

In contrast to the binary classification alternatives, we set the number of classification trees for Random Forest to 500, since predicting multiple classes is a more difficult task than distinguishing between only two classes. Table 11.3 presents the average confusion matrix over all 25 experiment runs. Here, the rows represent the actually observed time-to-failure classes and the columns reflect the predicted time-to-failure classes. The value in each cell illustrates the number of instances predicted for that particular combination of observed and predicted class labels. The confusion matrix clearly shows that most of the values fell on the main diagonal, meaning that most instances were predicted correctly. Moreover, most incorrect predictions were predicted in adjacent time-to-failure classes. In other words, the impending failure was detected, only the time windows were missed by a few hours to a day. Thus, these incorrectly predicted classes are less critical in practice as the critical event is nevertheless predicted with a relatively accurate time horizon. Furthermore, it is apparent that the actual time-to-failure classes `1`, `2`, and `5` could not be predicted as accurately as the others. Their class-wise accuracies are each less than 75%, while the class-wise accuracies of the other classes range from 87.1% up to 99.9%. This is most likely due to the fact that there are only very few training instances for these classes and further, the temporal distance between these classes is very small. In contrast, the white cells of the rightmost column show the number of cases in which the hard disk drive actually failed in the respective time window, but the multi-class approach did not predict a failure

Table 11.3: The confusion matrix for the multi-class time-to-failure classification approach. The rows indicate the actually observed (Ob) time-to-failure classes, while the columns represent the predicted (Pr) ones. In each cell, the value illustrates the number of instances predicted for that particular set of observed and predicted class labels. The green color highlights the correctly predicted instances. For instance, the second cell from the left in the third row displays a single instance that was predicted as "a failure will occur within the next hour", whereas the failure actually occurred in a time window of one to two hours after the measurement.

Ob \ Pr	**0**	**1**	**2**	**5**	**12**	**24**	**48**	**72**	**96**	**120**	**144**	**168**	**∞**
0	67	0	0	0	0	0	0	0	0	0	0	0	2
1	0	35	0	0	4	2	9	0	0	0	0	0	0
2	0	1	12	0	0	2	0	0	0	0	0	0	2
5	0	0	0	52	24	17	1	0	0	0	0	0	2
12	0	0	0	0	185	14	2	0	0	0	0	0	8
24	0	0	0	0	17	339	21	0	0	0	0	0	12
48	0	0	0	0	0	6	773	10	0	0	0	0	18
72	0	0	0	0	0	0	12	740	22	0	0	0	24
96	0	0	0	0	0	0	0	6	768	15	0	0	24
120	0	0	0	0	0	0	0	0	14	752	6	0	24
144	0	0	0	0	0	0	0	0	0	20	762	8	29
168	0	0	1	0	0	0	0	0	0	0	16	729	66
∞	0	0	1	1	0	1	0	1	1	0	1	5	14143

within the next week. These cells contain 211 out of 19829 instances, which can be explained by the fact that the S.M.A.R.T. parameters cannot cover all aspects that can lead to a failure [Sea99], such as sudden electronic or physical impacts.

To summarize the time-to-failure prediction quality of the Random Forest multi-class classification model, it achieved an average micro F1-score of 97.628%. The micro F1-score is the sum of the instances on the main diagonal (highlighted in green) divided by the total number of instances in the confusion matrix. Thus, the micro F1-score is equivalent to the multi-class accuracy.

Due to the fact that the micro F1-score of this multi-class approach cannot be directly compared to the F1-scores obtained in Section 11.2, we scaled down the multiple classes to the same two time-to-failure classes presented in Section 11.2. That is, we merge all classes except ∞ into one large class, i.e., the class indicating an impending failure within the next 168 hours. In this way, the classes match those used in Section 11.2, allowing us to compare them. However, since ESPO scored the best among the binary classification alternatives, Figure 11.3 displays only the comparison of the ESPO approach with the downscaled multi-class approach in the form of box plots. Again, the evaluation measures accuracy, precision, recall, and F1-score are shown on the horizontal axis, while the achieved values of ESPO and the downscaled multi-class ap-

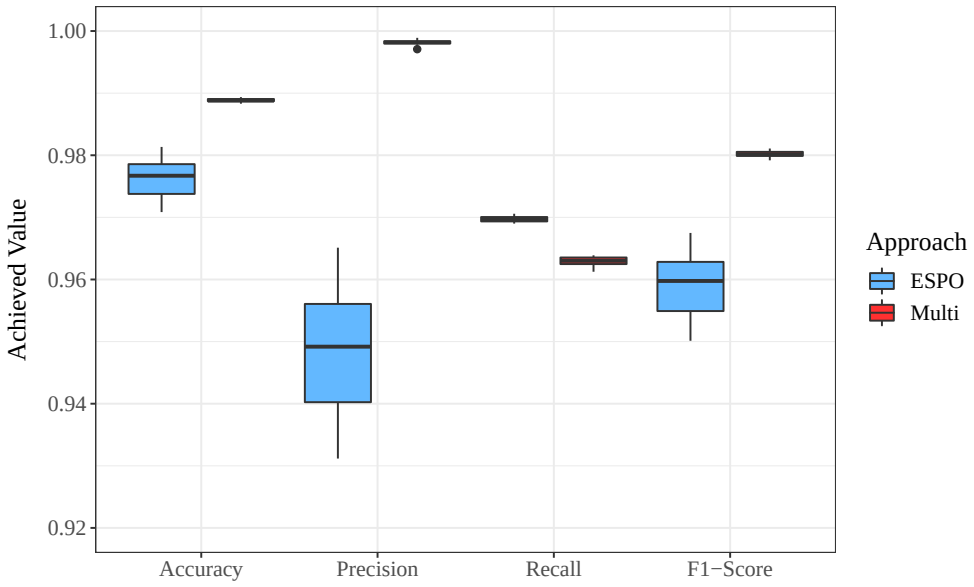


Figure 11.3: Box plots comparing the best binary classification approach (ESPO) with the multi-class classification approach downscaled to the same two classes.

proach are depicted on the vertical axis. The blue and red boxes represent the results of ESPO and the downscaled multi-class approach, respectively. The figure shows that the downscaled multi-class approach produced even better results with respect to accuracy (on average 98.885% vs. 97.642%), precision (99.818% vs. 94.913%), and F1-score (98.019% vs. 95.928%). However, binary classification with ESPO oversampling achieved a slightly higher recall (on average 96.970% vs. 96.283%). This fact reveals that, on the one hand, the binary ESPO approach detected more hard disk drives with impending failures. On the other hand, it demonstrates that ESPO also predicted more good hard disk drives as failing, i.e., ESPO had a higher false positive rate. In numbers, the downscaled multi-class approach achieved an average false positive rate of only 0.07%, while ESPO yielded an average false positive rate of 2.09%. The false positive rates of the other two binary classification approaches were even higher. Detecting more defective hard disk drives with a higher false positive rate can be useful if the cost of false alarms are negligible. However, in cases where false alarms result in high costs, the multi-class model is advantageous as it detects almost as many defective hard disk drives and incurs fewer costs for unnecessary hard disk drive replacements.

11.4 Time-to-Failure Regression

Similar to the evaluation of the multi-class time-to-failure classification approach, we set the number of decision trees for Random Forest to 500. Figure 11.4 presents box plots of the achieved hit rate, mean absolute error (MAE), and root mean square error (RMSE) for both regression approaches over the 25 experimental repetitions. By hit rate, we denote the ratio of correct time-to-failure regressions to the number of total regressions. Furthermore, we define a time-to-failure regression as correct if the actual time-to-failure falls within an interval of the predicted time-to-failure $\pm$ 10%. Thus, regarding the hit rate, a higher value indicates a better prediction quality, while the opposite applies for MAE and RMSE. The brown and green boxes represent the naïve regression and the proposed regression with pre-filtering, respectively.

The left subfigure of Figure 11.4 illustrates the hit rates achieved. Here, we can see that the approach of pre-filtering instances before model learning resulted in a higher hit rate than the naïve regression alternative. In terms of numbers, the pre-filtering approach achieved an average hit rate of about 84.9%, while the naïve version predicted a correct time-to-failure in only about 80.2% of all attempts. Opposite to the hit rate, for MAE and RMSE, a smaller value indicates a better regression quality. With respect to MAE (middle subfigure

of Figure 11.4), the pre-filtering approach yielded an average MAE of about 4.5 hours, whereas the naïve regression approach, on the contrary, exhibited an average MAE of about 10.0 hours. Moreover, the interquartile ranges for both approaches were extremely narrow. The large difference between the MAEs of the two regression alternatives and the small interquartile ranges suggest that the pre-filtering approach robustly predicted the time-to-failure more accurately. The same conclusion can be drawn by examining the obtained RMSE (right subfigure of Figure 11.4). The proposed pre-filtering approach showed an average RMSE of about 12.8 hours, while the RMSE of the naïve approach reached around 44.6 hours. Again, the interquartile ranges for both approaches were extremely narrow. For all three evaluation measures, even the best results of the naïve approach over all 25 runs performed inferior to the worst results of the pre-filtering-based regression approach.

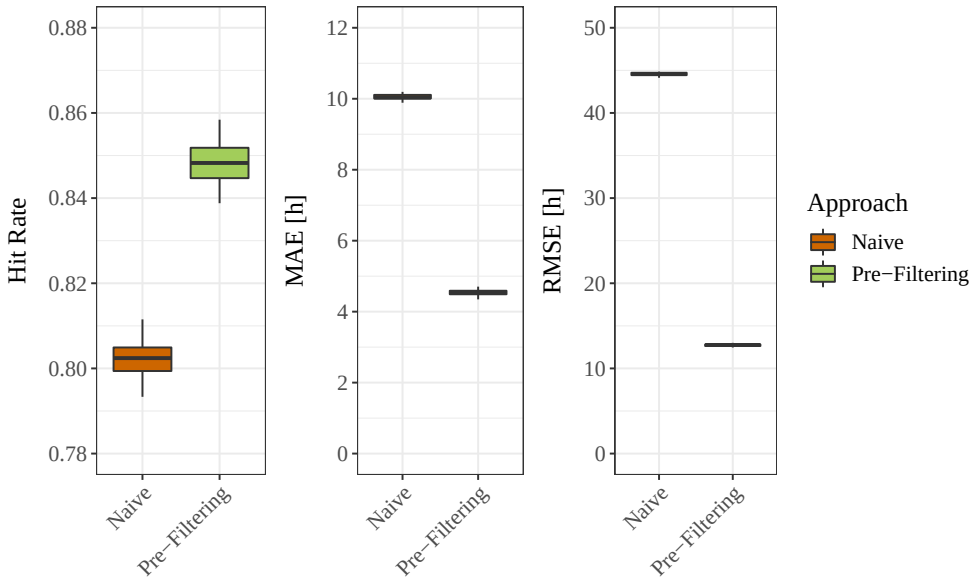


Figure 11.4: Box plots of the achieved time-to-failure regression quality of the naïve and pre-filtering-based regression alternatives.

Taking into account all three evaluation measures, it is evident that the proposed approach, which filters the instances before applying the regression model, predicts the time-to-failure with a much higher accuracy compared to the naïve version. With an average hit rate of almost 85% and a mean absolute error of only 4.5 hours, the time-to-failure regression is highly precise.

11.5 Runtime Comparison

After evaluating the prediction quality of the binary classification, multi-class classification, and regression approaches, we also analyzed the runtimes. Here, the runtime comprises both the time for model learning as well as for the prediction of the novel instances that were unknown during training. However, for all approaches, the time required for the prediction is negligible compared to the training time. We conducted the experiments in our private cloud using Apache CloudStack and a kernel-based virtual machine (KVM). The virtual machine is deployed on a host with 32 cores at 2.6 GHz each and 64 GB of memory, with hyperthreading enabled. The virtual machine runs Ubuntu 16.04 (64-bit) with 4 cores at 2.6 GHz each and 8 GB of memory. We have implemented the approach in R version `3.4.4`.

Figure 11.5 visualizes the runtimes over the 25 experimental runs for all classification and regression approaches by means of box plots. The horizontal axis represents the approaches, while the vertical axis depicts the required runtime. The boxes are colored according to the respective approach, with the coloring corresponding to the previous figures in this chapter. With an average runtime of approximately 27 seconds, the unmodified binary classification approach provided the shortest runtime, followed by our multi-class

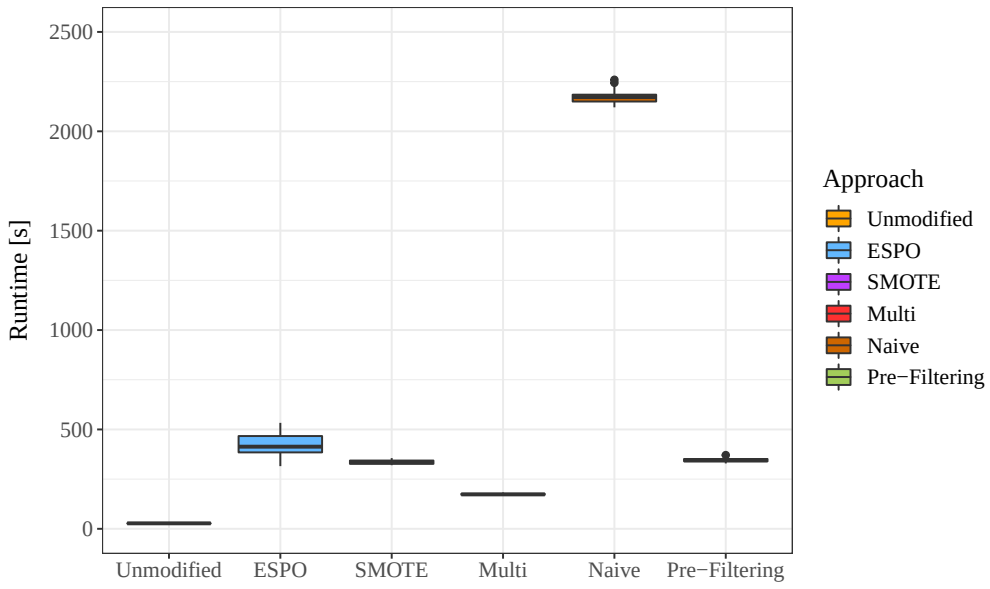


Figure 11.5: The required training times for all of the presented approaches.

classification approach with an average of about 174 seconds. Both oversampled binary classification approaches required a significantly longer runtime, averaging around 420 seconds for ESPO and 335 seconds for SMOTE. Thus, our multi-class classification model not only achieved better overall time-to-failure prediction quality, but also significantly shorter runtime than the best binary classification approach, namely ESPO. Among the regression approaches, the naïve approach needed on average roughly 2175 seconds, whereas the approach pre-filtering the data required only about 346 seconds on average. That is, our pre-filtering approach was more than 6 times faster than the naïve version and even faster than the binary classification approach with ESPO oversampling, while it significantly improved the hit ratio, MAE, and RMSE compared with the naïve regression version.

11.6 Feature Forecasting for Failure Prediction

In addition to using only the current S.M.A.R.T. features, we also assessed the effect of one-step-ahead forecast features on the prediction of impending hard disk drive failures. To this end, as described in Section 10.4, we applied ARIMA if the time series of S.M.A.R.T. features consisted of at least ten observations. Otherwise, we used the Naïve forecast model. Note that we intentionally did not apply Telescope and sNaïve, because S.M.A.R.T. features do not exhibit seasonal patterns. After deriving the forecasts, the Random Forest models were trained without employing an oversampling strategy. Again, 25 experimental runs were performed, with average prediction quality being reported below.

First, we compare the effect of adding forecasts as features to the binary models for hard disk drive failure prediction. As using ESPO oversampling performed best among the binary models, Figure 11.6 shows box plots of the achieved accuracy, precision, recall, and F1-score for ESPO in blue and Random Forest without oversampling strategy, but instead using one-step-ahead forecasts as additional features for hard disk failure prediction in yellow. A first observation is that the boxes, i.e., the interquartile ranges, of the model with forecast features are considerably smaller for accuracy, precision, and F1-score. Only with respect to recall, the interquartile range is slightly increased, although the interquartile range of the model incorporating forecasts also covers only 0.14 percentage points. Considering the value range of the box plots, the model using forecasts again substantially outperforms the ESPO model in terms of accuracy, precision, and F1-score, as the maximum values of the latter model are much smaller than the minimum values of the former model. The only exception is recall, where the ranges of achieved values overlap considerably.

Nevertheless, the box plot shows that the median recall of the model using forecasts is slightly higher. With respect to the average prediction quality, the model with forecast features achieved a mean accuracy, precision, recall, and F1-score of 98.821%, 98.856%, 97.005%, and 97.921%, respectively. In contrast, the binary ESPO model only achieved a mean accuracy, precision, recall, and F1-score of 97.642%, 94.913%, 96.970%, and 95.928%, respectively. Furthermore, integrating forecast features also reduced the average false positive rate from 2.089% to 0.451%. Thus, employing one-step-ahead forecasts in addition to the original features improved the prediction quality with respect to all evaluation measures. The application of paired t-tests underlined this statement, as the improvement of all evaluation measures except recall is significant with a confidence level of more than 99.999%. The enhancement with respect to recall is significant with a confidence level of 90% only.

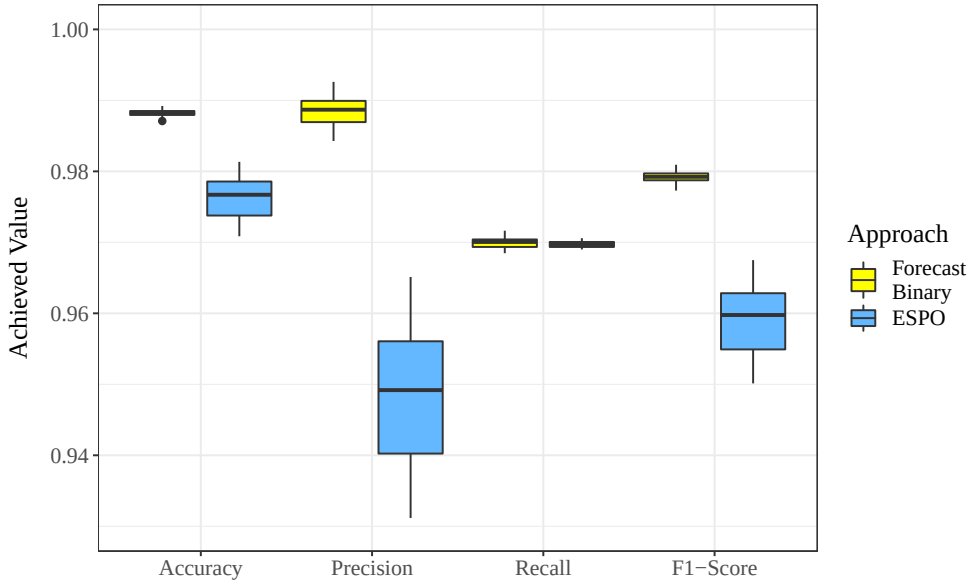


Figure 11.6: Box plots of the achieved prediction quality for the best binary classification approach (ESPO) and the binary model with one-step-ahead forecasts of the S.M.A.R.T. features in addition to the current S.M.A.R.T. features.

Next, we analyze whether forecasting S.M.A.R.T. features also improves the prediction quality of the downscaled multi-class models. To this end, Figure 11.7 displays box plots of the obtained accuracy, precision, recall, and F1-score of the downscaled multi-class model already evaluated in Section 11.3 in red and those of the downscaled multi-class model with one-step-ahead

S.M.A.R.T. feature forecasts as additional features in aquamarine. Compared with the binary models, both downscaled multi-class models performed much more homogeneously, which is evident from the fact that the boxes are very close to each other for all four evaluation measures. This can also be explained by the fact that the downscaled multi-class model has already achieved a very high prediction quality, which is hard to improve further. However, a general tendency can be observed that the model with the additional forecast features achieved a slightly better prediction quality with respect to accuracy, recall, and F1-score, while the achieved precisions of the two models are even closer to each other. Considering only the medians of the box plots, it can be seen that the model with the forecast features provides a slightly higher median prediction quality regarding all four evaluation measures. However, this is not the case for the average prediction quality. In terms of average prediction quality, the downscaled multi-class model already evaluated in Section 11.3 achieved a mean accuracy, precision, recall, and F1-score of 98.886%, 99.818%, 96.283%, and 98.019%, respectively. In contrast, the downscaled multi-class model using additional one-step-ahead forecast features provided an average accuracy, precision, recall, and F1-score of 98.921%, 99.813%, 96.412%, and

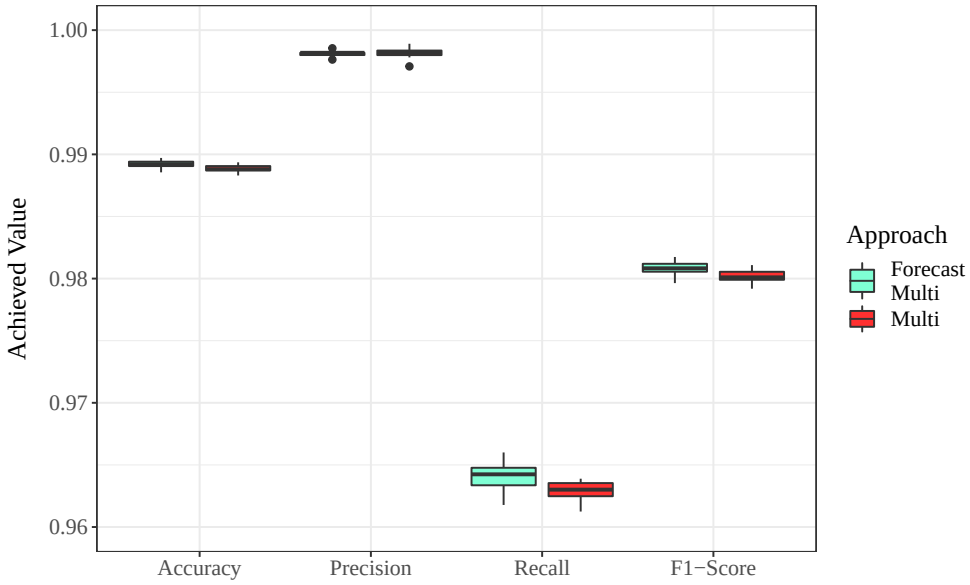


Figure 11.7: Box plots of the prediction quality of the downscaled multi-class classification (cf. Section 11.3) and the downscaled multi-class model using one-step-ahead forecasts of the S.M.A.R.T. features in addition to the current S.M.A.R.T. features.

98.083%, respectively. Thus, the mean accuracy, recall, and F1-score were slightly improved by using additional one-step-ahead forecast features, while the precision was marginally reduced. Therefore, we again applied paired t-tests to assess the significance of the improvements and degradation. The t-tests revealed that the enhancements by using the forecasts as additional features with respect to accuracy and F1-score were significant with a confidence level of 99.9%, whereas the improvement in terms of recall was significant with a confidence level of even 99.99%. Yet, the degradation in precision was not significant with a p-value of 0.5156. Finally, the average false positive rate of the model with forecasts also increased from 0.07% to 0.073%, although the variation was not significant with a p-value of 0.4975.

Lastly, Table 11.4 presents the average multi-class confusion matrix of the Random Forest model using both current S.M.A.R.T. features and forecast S.M.A.R.T. features. Although it looks very similar to Table 11.3 of the multi-class model not using forecast features, several differences can still be observed. First, both multi-class models achieved the same number of correctly predicted instances for the time-to-failure classes 0, 12, 24, and ∞. For the time-to-failure classes 2, 48, and 96, however, the multi-class model without forecast features yielded more correctly predicted instances, while the multi-class model with additional forecast features produced more correctly predicted instances for the time-to-failure classes 1, 5, 72, 120, 144, and 168. In total, the number of correctly predicted instances increased from 19357 to 19393 by adding the forecast features. Thus, the number of true positives was improved by 0.186%. Second, adding one-step-ahead forecast features of the S.M.A.R.T. measurements decreased the total number of instances that were not detected as failing in any of the twelve time-to-failure windows, although they did fail, from 211 to 205, representing a reduction of 2.844%. Another observation is that for the multi-class model with additional forecast features, the incorrectly predicted instances are more often close to the actual time-to-failure class. To put this statement into numbers, we determined the proportion of mispredictions made directly in an adjacent time-to-failure class. Graphically speaking, these instances are those located one cell to the left or right of the green diagonal of the confusion matrix in Table 11.4. Regarding the multi-class model without forecast features, 59.958% of all mispredictions were made in adjacent time-to-failure classes. When adding forecast features to the prediction model, this proportion increased to even 62.132%. In summary, the multi-class confusion matrix emphasizes the results already observed in the evaluation of the downscaled multi-class models, as extending the original feature set with one-step-ahead forecasts made the hard disk drive failure prediction even more accurate.

Table 11.4: The confusion matrix for the multi-class time-to-failure classification approach using current S.M.A.R.T. features as well as forecast S.M.A.R.T. features. The rows indicate the actually observed (Ob) time-to-failure classes, while the columns represent the predicted (Pr) ones. In each cell, the value illustrates the number of instances predicted for that particular set of observed and predicted class labels. The green color highlights the correctly predicted instances. For instance, the second cell from the left in the fourth row displays a single instance that was predicted as "a failure will occur within the next hour", whereas the failure actually occurred in a time window of two to five hours after the measurement.

Ob \ Pr	**0**	**1**	**2**	**5**	**12**	**24**	**48**	**72**	**96**	**120**	**144**	**168**	**∞**
0	67	0	0	0	0	0	0	0	0	0	0	0	2
1	2	42	0	0	6	0	2	0	0	0	0	0	0
2	0	0	13	1	1	0	1	0	0	0	0	0	2
5	0	1	0	70	21	2	1	0	0	0	0	0	2
12	1	0	0	1	185	10	0	0	0	0	0	0	12
24	1	0	0	0	19	339	14	0	0	0	0	0	17
48	0	0	0	0	0	12	765	11	0	0	0	0	19
72	0	0	0	0	0	0	12	742	20	0	0	0	24
96	0	0	0	0	0	0	0	6	763	21	0	0	24
120	0	0	0	0	0	0	0	0	8	758	8	0	22
144	0	0	0	0	0	0	0	0	0	19	768	9	22
168	0	0	0	0	0	0	0	0	0	0	15	738	59
∞	0	0	0	2	0	1	1	0	1	0	0	6	14143

11.7 Summary of Evaluation Findings and Threats to Validity

Based on the results of the experiments conducted, we derived the following main findings:

(I) Oversampling of minority class instances can improve the prediction quality of binary failure prediction models, although the prediction quality strongly depends on the oversampling methods employed. While ESPO enhanced the prediction quality in our case study, SMOTE actually worsened the prediction model. Moreover, the use of oversampling methods substantially increases the training time.

(II) Learning multi-class failure prediction models and downscaling the predictions to the same classes as when using a binary failure prediction model improves the prediction quality considerably. This phenomenon can be explained by the fact that the multi-class models are forced to learn more explicit patterns for the different time-to-failure windows.

(III) When using regression models, the prediction quality as well as the required runtime can be considerably improved by first employing a classification model that passes instances to the regression model only if the corresponding hard disk drive is expected to fail in the near future.

(IV) Adding one-step-ahead forecasts of features for model learning and prediction significantly improves the failure prediction quality for both binary as well as multi-class models.

Although we conducted an extensive evaluation, potential threats to validity are also apparent here. First, when comparing modeling alternatives, we did not attempt to optimize the machine learning model itself. Thus, we did not conduct an extensive hyperparameter optimization, nor did we employ different machine learning models. Instead, we used Random Forest with a fixed hyperparameter configuration. For more complex models, we manually increased the number of decision trees to meet the requirements of the more complex task. The results obtained from the conducted experiments should also be verified using other machine learning models, such as Support Vector Machine, XGBoost, or even neural networks.

Moreover, the oversampling strategies were only applied for binary classification models. Their effect on multi-class models has not been investigated yet. Given that SMOTE did not improve the binary prediction models, it is more likely that ESPO could also improve the prediction quality of multi-class models. However, this should be explored in further studies.

Finally, it has been shown that using ARIMA and Naïve forecast to estimate future S.M.A.R.T. features and using these one-step-ahead forecasts as addi-

tional features for the prediction models significantly improves the prediction quality. In the conducted experiments, however, the forecast models only forecast a single value for each point in time. Future studies could also investigate the effect of different forecasting horizons on the prediction quality.

11.8 Concluding Remarks

In this chapter, we evaluated the time-to-failure modeling alternatives described in Chapter 10. Therefore, this chapter contributes to the overall research question **RQ B.2**. On the one hand, the experimental results showed that oversampling is able to improve the predictive power of critical event prediction models. On the other hand, the results also revealed that the effect of oversampling highly depends on the technique used, as SMOTE oversampling actually degraded the quality of the binary prediction model. Second, we examined the prediction quality of multi-class prediction models for time-to-failure prediction. Here, the results demonstrated that these models could not only predict twelve fine-grained time-to-failure windows highly accurately, but that downscaling such a multi-class model to the same two classes used in a binary prediction model considerably improves the prediction quality. Regarding regression models for critical event prediction, we have shown that pre-filtering data for model learning, compared to the naïve approach of using all available data for regression model learning, both increases the prediction quality and reduces the training time.

Furthermore, we explicitly addressed research question **RQ B.2.3** by comparing the predictive power of time-to-failure prediction models with and without additional forecast features. Regarding the binary prediction models, the results showed that integrating time series forecasting into the prediction pipeline significantly improves the quality of time-to-failure predictions. Paired t-tests revealed that the improvement in terms of recall was significant with a confidence level of 90%, while for accuracy, precision, F1-score, and false positive rate, the improvements were significant with a confidence level of even more than 99.999%. For instance, with respect to the F1-score, the average prediction quality was increased from 95.928% to 97.921%, while the false positive rate was reduced from 2.089% to 0.451%. Considering the multi-class prediction models, the model using forecast features was again superior to the model without forecast features, although the models performed more homogeneously. While the differences in precision and false positive rate were not significant, the downscaled model using forecast features improved the average accuracy and F1-score with a confidence level of 99.9% and the average recall with a

confidence level of even 99.99%. The enhancement by using forecast features was also evident from the multi-class confusion matrix, as it yielded 0.186% more true positives, reduced the number of failing instances that were not covered by any time-to-failure class by 2.844%, and increased the percentage comparing the number of mispredictions in adjacent time-to-failure classes to all mispredictions from 59.958% to 62.132%. In conclusion, integrating time series forecasting into the feature engineering step significantly improved the prediction quality of both binary and multi-class prediction models, although the benefit was noticeably smaller for multi-class prediction models. However, although the runtime of integrating forecast features has not been studied in detail, it should be mentioned that it significantly increases the computational complexity of the model during on-line application, since the time series forecasting model needs to be re-trained for each prediction step. Therefore, a trade-off between computational complexity and prediction quality is required for multi-class prediction models.

Chapter 12

Time-to-Failure Prediction Methodology for Industrial Machines

The increasing digitalization and improvements in sensor-based data acquisition serve as drivers for many new applications in areas such as Industry 4.0 and Industrial IoT. Due to these developments, monitoring data of industrial machines can be stored easily, resulting in huge data sets. These can then be used to analyze the machines, with predictive maintenance being a typical application in this context.

Nowadays, many companies still follow a periodic maintenance approach. Accordingly, industrial machines are maintained at regular intervals to check for failures and take possible countermeasures. However, this often results in a waste of personnel and materials, since in many cases, maintenance is not necessary and could be postponed. A study by R. Mobley identified that this waste is responsible for about one-third of all maintenance-related costs [Mob02]. Furthermore, despite regular maintenance, sudden machine failures can nevertheless occur due to unexpected severe deterioration. Therefore, several companies have moved to condition-based maintenance. This maintenance strategy involves defining thresholds for particular sensors on industrial machines, which trigger maintenance actions when exceeded. However, defining such thresholds requires a high level of expert knowledge. Moreover, the analysis of threshold values is still a reactive maintenance approach. Due to these reactive maintenance strategies, unplanned downtimes still occur frequently. Emerson and the Wall Street Journal estimate the cost of these unplanned downtimes in manufacturing to be approximately 50 billion USD per year [Wal, Eme].

Unlike reactive maintenance strategies, predictive maintenance aims at predicting deteriorating health by analyzing previous monitoring data and learning from past machine failures. A report by PricewaterhouseCoopers and Mainnovation found that 95% of the 268 companies surveyed stated that using predictive maintenance improves one or more key maintenance value drivers [MH18]. The study also revealed that predictive maintenance increases

machine uptime by 9% and machine life by 20%. In addition, predictive maintenance reduces machine downtime by scheduling maintenance work on time, while keeping the risk of fatal breakdowns lower than reactive maintenance strategies. According to the report "Industrial Internet of Things: Unleashing the Potential of Connected Products and Services" by the World Economic Forum [Wor15], predictive maintenance reduces the cost of planned repairs by 12% and maintenance costs in general by 30%, while experiencing 70% less breakdowns. For this reason, according to Statista, 53% of the 272 companies surveyed in Germany attest that predictive maintenance has high or very high relevance for their company [Sta18]. In addition, the study by Pricewaterhouse-Coopers and Mainnovation showed that 60% of the companies surveyed already use predictive maintenance or plan to integrate it in the near future [MH18].

Due to this high practical relevance of predictive maintenance, much research has already been done in this field. In general, predictive maintenance can be divided into two components: time-to-failure prediction and maintenance scheduling based on predicted failures. Maintenance scheduling approaches are typically based on models, rules, or utility functions [YDN08, LW13, YM12, LVT16, VSSB13]. However, in this thesis, we focus only on time-to-failure prediction. Existing approaches in this area typically provide solutions developed for a specific machine. Examples include approaches explicitly developed for rotating machines, primarily bearings, based on Hidden Markov Models [TMMZT12, ZXK+05, LFH+18], Support Vector Machines [SM07, ZLZZ15, HZSG19], Decision Tree-based models [ASK13, ZLJ19, QLDL17], or neural networks [LCTH00, AFS+15, GCS16, LLT+17, HK19a]. Another intensively studied research area for time-to-failure prediction is the health state of lithium-ion batteries. Typically used algorithms in this area include Bayesian methods [HWOP11, NXT14, DHW21], Support Vector Regression [DJLW14, ZQZF18, XZCM20], and neural networks [LSG+10, ZXHP18, QLMF19].

In contrast to these highly specialized solutions, in this chapter, we propose a generic, end-to-end predictive maintenance methodology for time-to-failure prediction of industrial machines. The novel contribution compared to existing works is the end-to-end design, especially with respect to a universally applicable feature extraction based on integral values, which does not rely on profound expert knowledge. Furthermore, machine failures are predicted at different time horizons while investigating the application of different class labeling strategies. Nevertheless, the methodology proposed in this chapter is specifically designed for industrial machines. Therefore, the availability of data originating from sensors is assumed. Finally, we evaluate our proposed end-to-end methodology using real-world production data from a large-scale

press in Chapter 13. This type of machine has not been studied in detail before and differs significantly from the commonly studied machine types, namely bearings, lithium-ion batteries, and hard disk drives.

The content of this chapter is based on our previous work, which was published as a full paper as part of the 19th IEEE IES International Conference on Industrial Informatics (INDIN) [ZAG+21]. The remainder of this chapter is structured as follows: Section 12.1 presents the feature extraction from different categories of sensor data covered by the proposed critical event prediction methodology. Next, Section 12.2 shows the feature selection included in the methodology along with two possible feature normalization techniques. Section 12.3 explains three different target labeling methods, followed by model learning in Section 12.4. Section 12.5 describes the aggregation of predictions from multiple binary models. Finally, Section 12.6 summarizes the main contribution of this chapter.

12.1 Feature Extraction from Raw Sensor Data

In the context of industrial machines, a sensor is any digital measuring instrument that monitors either a particular component or the program executed by the machine. We distinguish sensors into four different categories, handling each of them in a different way: (I) physical units, (II) paired units, (III) temporal information, and (IV) program information.

Sensors of the *physical units* category are sensors that measure the current state or condition of a machine component, for instance, the position of an axis, the pressure, or the oil temperature.

Sensors belonging to the *paired units* category are similar, but they also provide a specified target value for the measured state or condition. The difference between the actual value and the target value can be used to measure the machine's quality when executing the implemented execution program.

In contrast, sensors of the category *temporal information* provide data that can be used to compute the duration of various events in the manufacturing process, typically the execution of specific manufacturing steps. Based on such data, a change in the health state of the machine can be detected. For example, in case of a leakage, the machine may need more time to build up a certain target pressure, which may be reflected in a longer duration.

Finally, sensors assigned to the category *program information* provide data that can be used as an indicator to split the measurement data of an entire manufacturing process into several phases, for example, drilling, milling, tool change. Note that in this time-to-failure prediction methodology for industrial

machines in general, we make the assumption that such program information is available, while in Chapter 8, which specifically targets rotating machines, this was not the case. In rotating machines, this program information can be extracted ex post due to the availability of meaningful vibration data, which is not necessarily the case for other industrial machines.

As part of the feature extraction process, the proposed methodology first utilizes this program information sensor data to partition all sensor measurements into several separate parts, each representing a different phase or activity in the manufacturing process. Subsequently, the first feature, namely duration, is computed for each phase by subtracting the first timestamp from the last timestamp. A different feature extraction approach is employed for the physical and paired units to gain insight into the performed execution. Most machine learning algorithms cannot utilize the monitored time series as input since the different executions' duration varies, resulting in time series with variable lengths. However, machine learning algorithms typically require the same input feature dimension. Moreover, passing entire monitoring time series as features leads to an explosion of the feature dimension, which slows down model learning tremendously, making it hardly applicable. Finally, the mere use of monitoring time series can lead to misclassifications if the recording is slightly shifted in time since the related features are no longer considered related by the machine learning model. Therefore, for physical and paired units sensors, the integral of the measured values is computed. For paired units sensors, the integral of the difference between the target values and the actual values is also computed. The integral-based feature extraction is illustrated in Figure 12.1, where a measured physical unit is plotted against the measurement time and the integral for an exemplary phase in the manufacturing process is

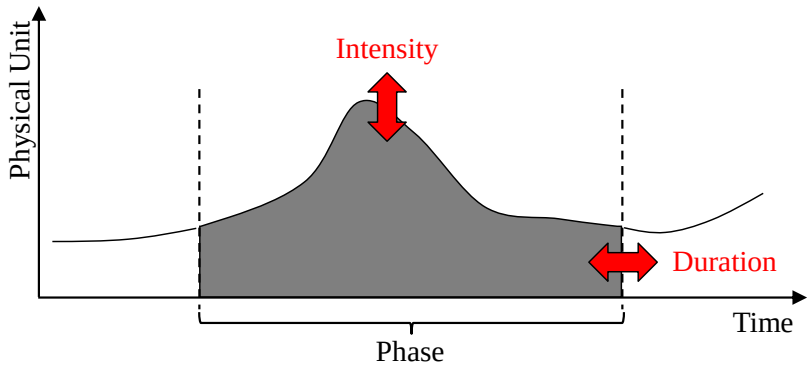


Figure 12.1: Schematic illustration of integral-based feature extraction.

shown in gray. The integral changes only when either the intensity (i.e., the range of values) of the physical unit or the duration of the respective phase increases or decreases, as indicated by the red arrows in Figure 12.1. Such changes usually indicate faulty machine behavior since the execution should follow a strict sequence of instructions, although slight variations are normal. The integral function provides a much more robust representation of these natural variations than the direct use of monitoring data.

12.2 Feature Handling

When applied to typical industrial machine scenarios with multiple physical unit sensors and multiple phases per manufacturing process, the described feature extraction procedure would result in a high number of extracted features. To avoid distortions during model learning and keep the model generation time as short as possible, only the most relevant features should be selected. For this purpose, the methodology includes a feature selection technique based on the Pearson correlation coefficient, which measures the linear correlation between two variables. The Pearson correlation coefficient ρ is defined as the covariance of the two variables X and Y divided by the product of their standard deviations σ_X and σ_Y, respectively:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \cdot \sigma_Y}. \tag{12.1}$$

Given that we can only consider samples of the actual distribution, we use an approximation of the Pearson correlation coefficient r computed based on the sample covariance and the sample standard deviations as

$$r_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \overline{x}) \cdot (y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}, \tag{12.2}$$

where n is the length of X and Y, x_i and y_i are the i-th observation of X and Y, and $\overline{x}$ and $\overline{y}$ are the sample means of X and Y, respectively. The correlation coefficient is in the range of $[-1, 1]$, where a value close to -1 indicates a strong negative linear correlation between the variables X and Y, and a value close to 1 indicates a strong positive linear correlation. A value around 0 indicates that there is no distinct linear correlation between the two variables.

In the context of the proposed time-to-failure prediction methodology for industrial machines, Y represents the target label to be predicted, while X constitutes one of the possible features extracted by applying the described

feature extraction procedure. The Pearson correlation coefficient is computed for each extracted feature and the target label to select the most relevant features. For this purpose, the operator can set a threshold $r_{\min}$ for r and specify whether only features with a correlation coefficient larger than $r_{\min}$ should be considered for model learning or also features with a correlation coefficient lower than $-r_{\min}$. In the latter case, strongly positively and strongly negatively correlated features are included for model learning.

However, since the values of different physical units are typically defined at different scales, the features must also be normalized prior to serving as input for machine learning models. To this end, the methodology incorporates two conventional feature normalization techniques: the min-max normalization and the Z-score normalization. The min-max normalization scales all values v of a particular feature into the range of $[0, 1]$ by subtracting the minimum value from each value v_i and dividing it by the difference between maximum and minimum values as

$$v_i^m = \frac{v_i - \min v}{\max v - \min v}. \tag{12.3}$$

However, to compute the min-max normalized values v^m, the global minimum and maximum of the feature must be known. In addition, the min-max normalization is sensitive to outliers since they may distort the minimum and maximum. The advantage of the second normalization method, namely Z-score normalization, is that neither the global minimum nor the global maximum of the feature needs to be known. Instead, this technique assumes that the observed sample is representative of the actual distribution with respect to the mean value and standard deviation. The Z-score normalization transforms all values v of a feature so that their new distribution has a mean value of 0 and a standard deviation of 1. To this end, it subtracts the mean value $\overline{v}$ from each value v_i and divides it by the standard deviation σ_v as

$$v_i^z = \frac{v_i - \overline{v}}{\sigma_v}. \tag{12.4}$$

Compared to the min-max normalization, the Z-score normalization is more robust against outliers, but it does not transform the values into a fixed range.

12.3 Target Class Mapping

After the feature engineering process, the target labels must be created and mapped to the monitoring data. To this end, the proposed methodology assumes the availability of failure records with timestamps. The timestamps of

those are matched to the timestamps in the sensor monitoring data, computing the time-to-failure of each instance in the data set. Although the time-to-failure can be used directly as a label to learn corresponding regression models, the methodology discretizes the time-to-failure in terms of classes, aiming at learning classification models instead of regression models. This design decision is based on the fact that most real-world scenarios for predictive maintenance in Industry 4.0 do not require time-to-failure information at the granularity of seconds or minutes, but rather hours or days. This time horizon lends itself well to modeling using classes.

To obtain the classification labels, the proposed methodology applies three different labeling techniques. These labeling techniques are illustrated by examples in Figure 12.2. In each case, the operator must first specify a fixed number of prediction windows of interest, which are mapped to different time-to-failure classes. For instance, if the operator specified the prediction windows as 12 hours and 1 day, the resulting discretized time-to-failure classes would represent failures in the three time horizons: $[0, 12)$, $[12, 24)$, and $[24, \infty)$ hours.

The first labeling technique is a *simple binary classification* per class. Accordingly, for each class, a vector of labels is created that is set to `1` if the time-to-failure of that particular instance is within the time-to-failure interval of the class, otherwise it is set to `0`. Only for the last class, no vector is created, since this class is already covered by all other classes. More precisely, if all other classes are `0`, this implies a `1` for the last class. This labeling approach is shown in Figure 12.2 in the middle. It requires a separate prediction model for each class except for the last one.

The second labeling technique also models the classes separately, but in contrast to the simple binary classification, the vector of class labels is set to `1` if the time-to-failure of the respective instance is smaller than the upper bound of the time-to-failure interval of the class. Thus, for a single instance, multiple classes can have a label of `1`, while in the simple binary case, only the highest class would be set to `1`. This is a valid labeling approach for predicting time-to-failure since a failure that occurs after, for instance, 5 hours can be sorted into the time window of up to 12 hours as well as the time window of up to 24 hours. The labeling approach is presented at the bottom of Figure 12.2. Similar to the simple binary labeling technique, this results in a separate prediction model for each class except the last one. Here, the last class would contain even less information since it would be `1` for each instance. In the following, we refer to this labeling technique as *stacked binary model*.

Finally, the third labeling technique is *multi-class labeling*, which creates a single vector of class labels containing the name of the time-to-failure interval

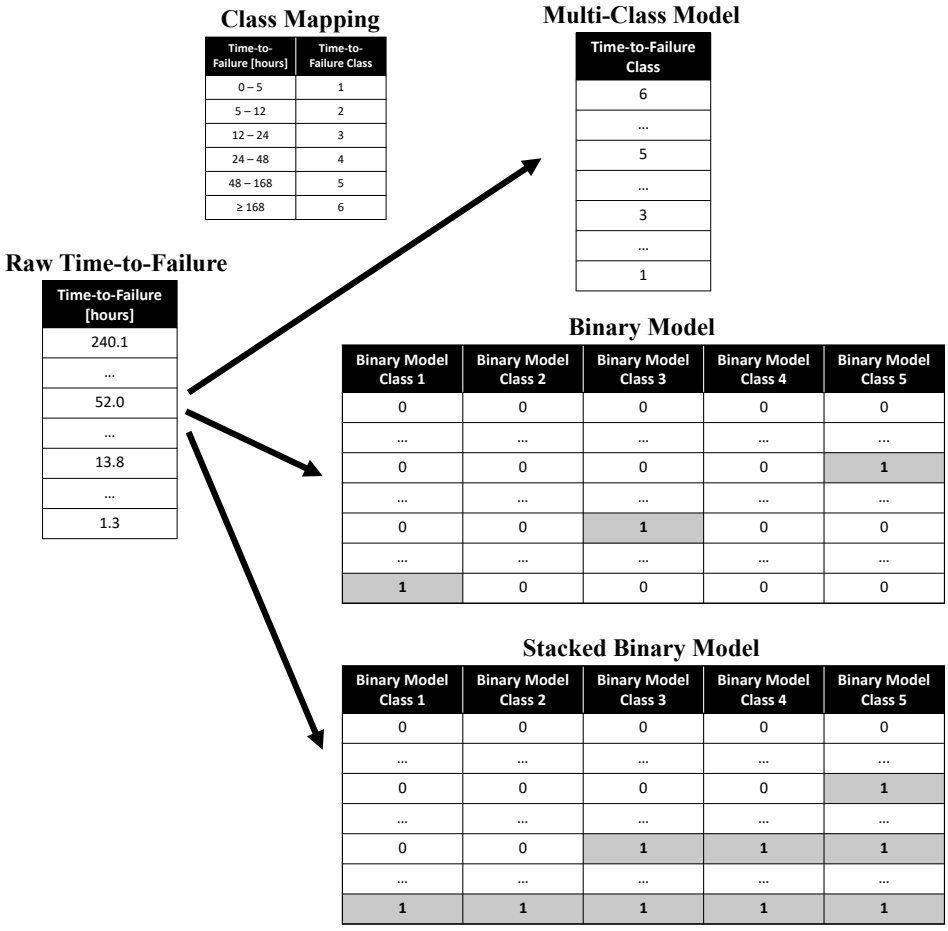


Figure 12.2: The labeling techniques included in the methodology.

class into which the time-to-failure of the corresponding instance falls. The top part of Figure 12.2 depicts this labeling approach. Unlike the first two labeling techniques, this technique requires only a single prediction model.

12.4 Model Learning

In order to learn a model for time-to-failure prediction, the methodology uses four different machine and deep learning methods. However, similar to the implemented feature selection and feature transformation techniques, other algorithms would be possible and could be incorporated into the methodology

as well. The four methods currently included in the time-to-failure prediction methodology are Random Forest (RF), eXtreme Gradient Boosting (XGBoost), Feed-Forward Neural Network (FFNN), and AutoML [HKV19]. For more details on the first three methods, we refer the reader to Sections 2.4 and 2.5. The latter model, namely AutoML, is a framework that automatically trains multiple prediction models of different machine learning methods, optimizes their hyperparameters, and creates stacked ensembles from the best of these individual models [HKV19].

For each of these machine and deep learning methods, hyperparameter tuning is carried out using the grid-search technique. The only exception is AutoML, because AutoML already performs such tuning internally. For hyperparameter tuning, the operator can specify a list of possible configurations for each parameter. However, in the case study presented in this thesis (cf. Table 13.1 in Section 13.1), we also provide parameter configurations that have already achieved satisfactory results in practice and can therefore be adopted if no dedicated expert knowledge is available. Regarding Random Forest, the adjustable parameters are the number of decision trees `ntree` and the number of predictors `mtry` to randomly choose from at each split. For XGBoost, the parameters to be defined are the step size shrinkage `eta`, the maximum tree depth `max_depth`, and the minimum loss reduction necessary to partition another leaf node of the tree `gamma`. As the number of configurable parameters of the FFNN is vast, a pre-defined architecture is implemented consisting of five dense layers with an adjustable dropout after each layer. The rectified linear unit (ReLU) is used as the activation function. For the final output, the sixth layer employs the softmax activation function. Here, the number of nodes corresponds to the number of classes defined by the operator. The design of this pre-defined FFNN architecture is based on F. Chollet and J. Allaire [CA18]. However, the resulting FFNN still provides a broad range of parameters, so the methodology splits the grid-search into two successive iterations. The first grid-search aims to estimate the required batch size and the overall complexity of the model by varying the batch size, the number of nodes per layer, and the dropout ratio. For this first iteration, the number of nodes and dropout are the same for all layers to keep the number of permutations manageable. Based on the top results of this first grid-search, a second grid-search is performed considering only the parameter settings included in these top results. Here, the number of nodes per layer and the dropout after each layer are set individually per layer. Thus, in this second iteration, the pre-selected parameters are fine-tuned. To learn the FFNN models, the `adam` optimizer is used with categorical cross entropy as the loss function.

12.5 Prediction Aggregation

Regardless of the labeling method analyzed, each prediction method returns only a single time-to-failure class when the models are applied. Thus, concerning the multi-class model, the output is simply the predicted class. However, for the binary and stacked binary labeling methods, during the prediction process, several of the binary models may predict `1`, that is, an impending failure within their respective time window. In this case, only the time-to-failure class with the shortest lead time is returned as the final prediction.

12.6 Summary and Discussion

We conclude this chapter by providing an answer to research question **RQ B.3**, thus summarizing the main contribution of this chapter. As research question **RQ B.3** aims at designing a generic, end-to-end workflow for critical event prediction of industrial machines while keeping the required domain and expert knowledge low, this chapter presented a methodology for precisely this purpose. To this end, the methodology assumes a database of sensor monitoring records as well as failure reports. The methodology distinguishes between four different sensor types, all of which are treated differently. While program information sensor data are used to partition the monitoring records into individual processing phases, timestamps are utilized to derive the duration of each processing step, and integral features are computed on both physical units sensors and paired units sensors. The advantage of integral features is that they do not require domain knowledge about the executions performed or the machine under consideration. Moreover, integral features cover variations in both the intensity of the monitored signal and its duration. Subsequently, a feature handling task offers the capability of correlation-based feature selection and feature normalization. To predict the time-to-failure in multiple time windows, the methodology provides three labeling procedures, namely simple binary classification, stacked binary classification, and multi-class classification. For this purpose, the operator can specify multiple time-to-failure windows. Currently, the methodology includes four prediction methods, namely Random Forest, XGBoost, Feed-Forward Neural Network, and AutoML. However, more prediction methods can be easily integrated in the future. For model learning, the methodology incorporates a grid-search-based hyperparameter tuning. Finally, in the case of the two binary classification alternatives, a prediction aggregation step is included in the time-to-failure prediction methodology to merge the predictions of the different binary models. Thus, the only domain

knowledge required to apply this time-to-failure prediction methodology is the database along with the sensor type mapping. With respect to machine learning expert knowledge, the only inputs required are a threshold for the correlation-based feature selection and possible hyperparameter configurations for the prediction models. Although this seems to be a substantial limitation, we provide a list of hyperparameter settings for the considered prediction methods that have proven to be effective in a real-world case study.

Chapter 13

Evaluation of the Time-to-Failure Prediction Methodology in a Real-World Case Study

As mentioned in Chapter 12, we assessed the prediction quality of the proposed generic, end-to-end time-to-failure prediction methodology for industrial machines employing a real-world case study. In the context of this case study, we investigated impending failures of a large-scale industrial press in six time-to-failure prediction windows. This type of machine differs considerably from the types of machines already studied in depth, such as bearings, lithium-ion batteries, and hard disk drives. First, Section 13.1 provides further insights into the case study, including a description of the monitored data and the evaluation design. Next, Section 13.2 presents the results of the case study in the form of a macro view, where the obtained prediction qualities are averaged across the multiple time-to-failure windows. Subsequently, Section 13.3 provides a more in-depth analysis of the prediction quality by reporting the prediction qualities per time-to-failure class. Section 13.4 focuses only on the best prediction method as well as the best labeling technique to present even more details on the prediction provided by this configuration. Next, Section 13.5 discusses the evaluation results and potential threats to validity. Finally, we conclude this chapter in Section 13.6 with a summary of the main contribution.

The contents of this chapter are based on our earlier work, which was published as full paper as part of the 19th IEEE IES International Conference on Industrial Informatics (INDIN) [ZAG+21].

13.1 Case Study Details

This case study analyzes data collected from a large-scale industrial press that was monitored for several years. During this time, every 100th stroke was recorded with a resolution of 1 millisecond. A stroke takes approximately four

seconds, during which the press processes 8 different phases. The phases are identified in the sensor recordings by the logged command, which can be considered as sensor data with *program information*. Subsequently, the timestamps of the sensor measurements are used as *temporal information* to compute the duration of each processing phase and the entire stroke, resulting in 9 duration features. In addition to the timestamps and program command, 118 sensor measurements are available for each recording entry. This results in 118 physical unit sensors, of which 32 are paired units. That is, there are 16 pairs of target and actual values. For these pairs, the deviation is computed and utilized as physical unit sensor data. In total, this yields $9 + (118 + 16) \times 8 = 1081$ features per stroke in a data set of 47,152 strokes.

For training the Feed-Forward Neural Network (FFNN), the features were transformed using Z-score normalization. Both normalization techniques were tested for the other prediction methods, but the best predictions were obtained without any normalization. In addition, correlation-based feature selection was employed to reduce the number of features. However, since the reduced feature set did not improve the prediction quality, in the following, we present the results of using all features for model learning.

In order to predict the time-to-failure in several time windows, 5 hours, 12 hours, 1 day, 2 days, and 1 week were chosen as class boundaries, resulting in classes `1` to `6`, where the time-to-failure ranges from $[0, 5)$, $[5, 12)$, $[12, 24)$, $[24, 48)$, $[48, 168)$, and $[168, \infty)$ hours, respectively. These limits were selected because predictions in these time windows allow technicians sufficient lead time to take countermeasures.

Table 13.1 lists the parameter settings for the applied grid-search-based hyperparameter tuning. Note that the default for `mtry` for Random Forest (RF) is set to $\lfloor \sqrt{\#\text{features}} \rfloor$ for classification, resulting in 32. The other two settings for `mtry` are obtained by multiplying the square root of the number of features by factors of 2 and 4, respectively, with subsequent rounding. The XGBoost parameter specification is based on the recommendations of A. Jain [Jai16]. Although AutoML carries out hyperparameter tuning internally, one parameter, namely the maximum runtime, must be specified. We set the maximum runtime to 6 hours per binary class for the binary and stacked binary models, resulting in a maximum total runtime of 30 hours. For the multi-class model, the maximum runtime was set to 22 hours. Although this seems like a bias, it does not affect the final prediction results because the maximum runtime was not reached due to the early stopping.

In order to assess the prediction quality of the time-to-failure prediction methods, we compute accuracy, recall, precision, F1-score, and Cohen's kappa

Table 13.1: Parameter settings of the prediction methods for the grid-search-based hyperparameter tuning.

Method	Parameter	Settings
Random Forest	`ntree`	500, 1000, 2000
	`mtry`	32, 65, 131
XGBoost	`eta`	0.05, 0.10, 0.30
	`max_depth`	6, 8, 10, 15
	`gamma`	0, 2, 10
Feed-Forward Neural Network	batch size	100, 500, 1000
	number of nodes	64, 128, 256, 512
	dropout ratio	0.0, 0.2, 0.4

coefficient [McH12] as evaluation measures. For more details on the evaluation measures, we refer to Section 2.6.2. However, note that Cohen's kappa coefficient is more expressive than ordinary accuracy for imbalanced data sets since it incorporates class frequencies. To this end, Figure 13.1 shows the distribution of classes in the data set used in this case study. It is evident that the classes are

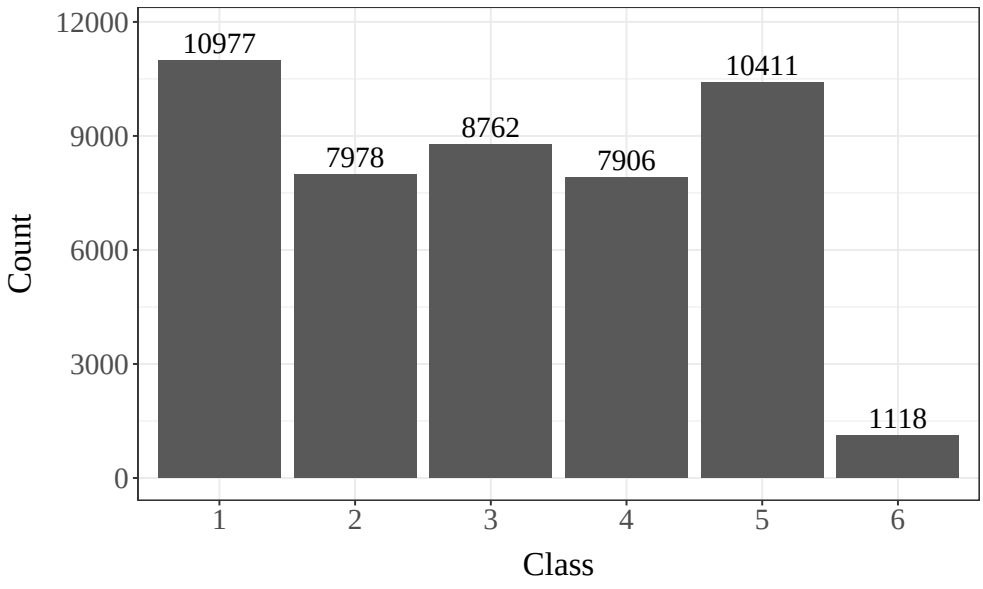


Figure 13.1: The distribution of time-to-failure classes.

highly imbalanced. For instance, the time-to-failure class 6 (i.e., the machine will not fail within the next week) occurs 1118 times in the data set, while all other classes are in the range of 7900 to 11000 instances.

Finally, 5-fold cross-validation [HTF09] was used for the overall evaluation. Accordingly, randomly selected 80% of the entire data set were used to learn the model, while the remaining 20% of the data set were used as a test set to evaluate the obtained prediction quality. This procedure was repeated five times, such that each instance in the data set was included in the test set exactly once. Note that the models were completely rebuilt from scratch between the five repetitions. Consequently, the models could not remember an instance from the training set for a subsequent repetition where that particular instance might be part of the test set.

13.2 Macro Results

This section compares the best models for the three different labeling techniques throughout all classes. Thus, the measures recall, precision, and F1-score are computed for each class and, then, combined using the arithmetic mean, resulting in macro recall, macro precision, and macro F1-score. The class-wise results are presented subsequently in Section 13.3.

Note that for the binary or stacked binary labeling, the best model may actually consist of five different prediction methods and hyperparameter settings corresponding to the time-to-failure classes 1 to 5 (cf. Section 12.3), while for the multi-class labeling, the best model involves only a single prediction method with its best hyperparameter configuration. Table 13.2 presents the accuracy, kappa, macro F1-score, macro recall, and macro precision obtained by the best models for each combination of labeling and prediction method. In addition, for the binary and stacked binary labeling methods, the best results obtained by combining different prediction methods are also reported.

Regarding the binary models, it can be observed that Random Forest provided by far the worst result with respect to accuracy, kappa, macro F1-score, and macro recall. However, it achieved the second highest macro precision. XGBoost and FFNN performed fairly similarly with respect to all measures except macro precision, where XGBoost yielded the best result and FFNN the worst. Finally, AutoML exhibited the best overall prediction quality when using binary labeling. With respect to all evaluation measures except macro precision, it considerably outperformed the other prediction methods. For this reason, the overall best combination of binary models contains only AutoML models, all of which were based on boosting, i.e., mainly XGBoost models and some Gradient

Table 13.2: The achieved accuracy, kappa, macro F1-score, macro recall, and macro precision for the best parameter settings of each individual prediction method as well as their overall best combination for each labeling approach. The best values per labeling method are highlighted in bold, while the overall best values are highlighted in bold and written in italics.

Labeling Method	Prediction Method	Accuracy	Kappa	Macro F1-Score	Macro Recall	Macro Precision
Binary	Best Random Forest	0.5670	0.5042	0.5796	0.6140	0.8065
	Best XGBoost	0.7183	0.6672	0.6933	0.7476	**0.8070**
	Best FFNN	0.7129	0.6519	0.6657	0.7320	0.6931
	Best AutoML	**0.7934**	**0.7475**	**0.7372**	**0.8036**	0.7555
	Best Combination	**0.7934**	**0.7475**	**0.7372**	**0.8036**	0.7555
Stacked Binary	Best Random Forest	0.7576	0.7009	0.7749	0.7698	0.8053
	Best XGBoost	0.8378	0.7995	0.8526	**0.8522**	**0.8621**
	Best FFNN	0.7568	0.6983	0.7703	0.7610	0.7835
	Best AutoML	0.8280	0.7863	0.8417	0.8371	0.8472
	Best Combination	**0.8409**	**0.8023**	**0.8533**	0.8492	0.8590
Multi	Best Random Forest	0.8763	0.8462	0.8797	0.8770	0.8845
	Best XGBoost	0.8862	0.8585	0.8920	0.8907	0.8946
	Best FFNN	***0.8972***	***0.8724***	***0.9000***	***0.9041***	***0.8951***
	Best AutoML	0.8408	0.8021	0.8519	0.8482	0.8571

Boosting Machine models. Given that the best combination is equal to the application of the five best binary models of AutoML, both rows in Table 13.2 display the same values.

Concerning the stacked binary models, the first finding is that the prediction quality of all methods increased greatly. Here, Random Forest and FFNN provided comparable results with respect to all measures, although the achieved values of Random Forest were nevertheless slightly higher. Moreover, XGBoost outperformed AutoML with respect to all measures, although all models learned by AutoML were still based on boosting methods only. However, a mixture of XGBoost and AutoML models was the best combination of models. The best combination model utilized XGBoost with `eta` = 0.05, `max_depth` = 15, and `gamma` = 0 for the time-to-failure classes `2` and `3`, while AutoML models were used for the time-to-failure classes `1`, `4`, and `5`.

In contrast to this dominance of AutoML and XGBoost for the binary and stacked binary models, FFNN significantly outperformed AutoML in the case of multi-class labeling, even though AutoML learned a complex ensemble model consisting of eight Gradient Boosting Machine models, seven XGBoost models, one linear model, two Random Forest models, and seven deep neural networks. Yet, XGBoost provided the second best results on all evaluation measures. Moreover, even Random Forest clearly outperformed AutoML. The results obtained by the multi-class FFNN were the best among all labeling techniques and prediction methods with an accuracy, kappa, macro F1-score, macro recall, and macro precision of 89.72%, 87.25%, 90.00%, 90.41%, and 89.51%, respectively. The configuration of this optimized FFNN was as follows:

- batch size = 500
- number of nodes per layer = [256, 256, 512, 512, 256]
- dropout per layer = [0.2, 0.2, 0.0, 0.0, 0.2]

Finally, when comparing the kappa values of the best prediction methods per labeling strategy, only the multi-class labeling approach reached a value greater than 0.81. According to J. Landis and G. Koch [LK77], this value is considered the lower bound of a near-perfect match. In terms of numbers, the best multi-class labeling model exceeded this limit considerably, showing a kappa value of 0.8724, while the best stacked binary labeling model almost reached this limit with a kappa value of 0.8023, and the best simple binary labeling model remained far from it with a kappa value of only 0.7545. These results might be explained by relationships between the time-to-failure classes. When modeling all six classes with a single prediction model, subtle differences

between the classes can be learned that are likely to be missed when analyzing each class individually. Therefore, the prediction results for the individual classes are presented in the next section.

13.3 Results by Class

In this section, the achieved prediction quality per class is evaluated in detail. For this purpose, Table 13.3 reports the F1-score, recall, and precision for the binary (B), stacked binary (S), and multi-class (M) labeling for each of the six time-to-failure classes. The best values per evaluation measure and time-to-failure class are highlighted in bold, while the worst values are underlined. Regarding the F1-score, it is apparent that the multi-class labeling performed best, as it achieved the highest value for five of the six classes. Only for the last class (i.e., no machine failure within the next week), the stacked binary labeling approach performed better by 0.0081 percentage points. The binary labeling approach yielded the worst F1-scores for all time-to-failure classes. Nevertheless, it achieved the same F1-score as the stacked binary labeling approach for time-to-failure class 1. This is due to the fact that the labeling procedure is the same for the first class in both labeling strategies and both use the same AutoML model for class 1. This also applies to recall and precision in the following. Moreover, the F1-score of binary labeling for classes 1 to 5 was close to the F1-score of stacked binary labeling, while it dropped significantly for class 6.

Table 13.3: The achieved F1-score, recall and precision per class for the best models of each labeling method (LM), i.e., binary (B), stacked binary (S) and multi-class (M). The best values are written in bold, while the worst values are underlined.

Measure	LM	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
F1-Score	B	0.8155	0.6935	0.7955	0.8595	0.9356	0.3234
	S	0.8155	0.7273	0.8142	0.8765	0.9403	**0.9460**
	M	**0.8807**	**0.8353**	**0.8772**	**0.9111**	**0.9589**	0.9379
Recall	B	0.8356	0.6161	0.7319	0.8129	0.9085	0.9168
	S	0.8356	0.6980	**0.8538**	0.8679	0.9158	0.9240
	M	**0.8913**	**0.8420**	0.8495	**0.9161**	**0.9587**	**0.9669**
Precision	B	0.7964	0.7933	0.8712	**0.9116**	0.9643	0.1964
	S	0.7964	0.7591	0.7781	0.8853	**0.9662**	**0.9690**
	M	**0.8703**	**0.8206**	**0.9068**	0.9062	0.9561	0.9105

While the stacked binary labeling and the multi-class labeling achieved an F1-score of 0.9460 and 0.9379, respectively, the simple binary labeling obtained an F1-score of just 0.3234 for time-to-failure class 6.

When analyzing the achieved recall, the multi-class labeling approach again performed best for five of the six time-to-failure classes. This time, however, the stacked binary labeling surpassed the multi-class labeling for time-to-failure class 3. Similar to the F1-score results, the difference is relatively small, with only 0.0043 percentage points. Again, the binary labeling performed worst for all six time-to-failure classes, while it yielded the same results as the stacked binary labeling approach for class 1. Nevertheless, the binary labeling did not provide as significantly poor results as for the F1-score of class 6. The most considerable difference between the binary labeling and the second best labeling method is for class 3, where the binary labeling provided a recall of 0.7319 and the multi-class labeling resulted in a recall of 0.8495. As recall is the number of instances correctly predicted as positive relative to the total number of actual positive instances, these results suggest that all three labeling methods predict imminent machine failures fairly well while keeping the number of missed imminent failures low.

Finally, the multi-class labeling approach obtained the highest precision for three time-to-failure classes (i.e., classes 1 to 3), while the stacked binary labeling approach performed best for two time-to-failure classes (i.e., classes 5 and 6), and the simple binary labeling approach performed best for time-to-failure class 4. In contrast, the stacked binary labeling performed worst for four time-to-failure classes, i.e., classes 1 to 4, while the binary labeling provided the same precision for class 1. Remember that the precision is the number of instances correctly predicted as positive relative to the total number of instances predicted as positive. Thus, the stacked binary labeling approach provided comparatively many false positives for classes with a short time-to-failure, while it provided the lowest number of false positives for the two classes with the highest time-to-failure. Similarly, the binary labeling approach resulted in the fewest false alarms for the classes indicating failures within the next 24 to 48 hours. However, the precision of binary labeling for class 6 is by far the worst with only 0.1964, resulting in the low F1-score of binary labeling. Accordingly, the binary labeling approach predicted many instances as not failing within the next week, even though the actual time-to-failure was much shorter. This demonstrates the necessity of comparing different labeling strategies, as their performance varies considerably between classes as well as in the aggregate. Finally, the multi-class labeling produced the fewest false alarms for the most urgent, and thus, most relevant time-to-failure classes.

13.4 Details on the Best Predictions

This section provides more detailed insights into the overall best predictions, which were obtained by the Feed-Forward Neural Network using multi-class labeling. Therefore, Table 13.4 reports the accumulated confusion matrix over all five folds, such that each instance in the data set is used exactly once in the test set. While the columns of the confusion matrix show the actually observed (Ob) time-to-failure classes, the rows represent the predicted (Pr) classes. The value in each cell displays the number of instances predicted for that particular combination of observed and predicted class labels. Finally, the cells highlighted in green indicate the correctly predicted instances.

The confusion matrix demonstrates that most instances were predicted correctly. In particular, only a few mispredictions were made for the higher time-to-failure classes. In contrast, most of the mispredictions were for the shorter time-to-failure classes, where the temporal distance between classes is also shorter. Moreover, the majority of the mispredictions are located directly in adjacent classes. Thus, the time-to-failure mispredictions mostly fall in the closest time-to-failure windows, which leads to only minor discrepancies during practical applications. Only relatively few failure instances show a higher prediction error (i.e., several cells apart in the confusion matrix).

Furthermore, for many practical applications, such a fine-grained time-to-failure prediction might not be necessary, allowing a binary prediction instead with a prediction horizon of two days to be sufficient. To evaluate our time-to-failure prediction methodology for such a scenario, we ex post reduced the confusion matrix with six classes to the binary prediction of whether or not the machine failure will occur within the next two days. Consequently, the

Table 13.4: The accumulated confusion matrix over all five folds for the Feed-Forward Neural Network with multi-class labeling. The green color highlights the correctly predicted instances.

Pr \ Ob	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
Class 1	9719	755	243	131	144	20
Class 2	654	6635	462	86	78	12
Class 3	262	398	7671	235	42	18
Class 4	132	109	297	7283	166	11
Class 5	190	69	74	163	9959	18
Class 6	20	12	15	8	22	1039

time-to-failure classes 1 to 4 represent the "failure class", whereas the time-to-failure classes 5 and 6 are merged to form the "no failure class". For this binary prediction, our predictive maintenance methodology predicted impending failures with an even substantially higher accuracy, F1-score, and kappa of 97.79%, 98.54%, and 94.03%, respectively.

13.5 Discussion and Threats to Validity

This section first summarizes the main evaluation findings, followed by a discussion of potential threats to validity.

13.5.1 Summary of Evaluation Findings

Drawing on the results of the real-world case study conducted, we derived the following main findings:

(I) The integral features allow for high prediction quality in the context of critical event prediction for industrial machines while requiring little to no domain knowledge.

(II) Given the prediction of six time-to-failure classes, all labeling strategies provided satisfactory results, although there were substantial differences in the obtained prediction quality. While the simple binary classification provided the worst results, the stacked binary classification clearly enhanced the prediction quality. However, both versions of the binary models were considerably outperformed by the multi-class model. The differences in the achieved prediction quality are particularly evident when considering the prediction quality per time-to-failure class.

(III) While boosting methods were superior for the binary and stacked binary labeling strategies, the Feed-Forward Neural Network surpassed the other models with respect to the multi-class model, which eventually achieved the highest prediction quality.

(IV) By downscaling the six fine-grained time-to-failure classes to the binary prediction of whether or not the machine is likely to fail within the next two days, the obtained accuracy, F1-score, and kappa increased even further from 89.82%, 90.00%, and 87.24% to 97.79%, 98.54%, and 94.03%, respectively.

13.5.2 Threats to Validity

Although it is common practice to apply multiple binary prediction models to predict impending failures with different lead times (e.g., J. Li *et al.* [LJJ+14]), the results of our case study revealed that such an approach does not necessarily

yield good results. In fact, the simple binary models performed the worst in our case study. However, this result only represents our conducted case study, while the ranking of best labeling methods and best machine learning algorithms for another use case might be completely different. Nevertheless, the results emphasize the importance of testing different class labeling strategies, as their influence can be even more significant than the choice of the machine learning method itself, as it was the case in our case study.

As only integral features and duration features were used, the methodology does not require in-depth domain knowledge, which makes it easily applicable to other application scenarios as well. In addition, the case study considered failures of a large-scale industrial press, which has hardly been analyzed in the literature so far. However, we evaluated the prediction quality of the methodology only on one use case. Further investigations of other use cases should be conducted in this respect.

Regarding prediction aggregation for the stacked binary model, we also compared using a strategy that analyzes the consistency of class labels ones across the binary models' predictions. However, using the time-to-failure class with the shortest lead time provided the best results. Nevertheless, there exist many other ways to aggregate the predictions of the binary models, such as incorporating class probabilities. Therefore, future research should develop different methods for prediction aggregation and investigate their effects on the overall prediction quality.

Finally, we also employed Recurrent Neural Networks, or more specifically Gated Recurrent Units and Long Short-Term Memory networks, to predict the time-to-failure. However, their prediction quality was unsatisfactory. The LSTM network achieved a maximum macro F1-score of 75%, while the GRU network provided an even lower maximum macro F1-score of only 66%. This can be explained by the nature of the data set used in this case study. First, the sensor readings of individual strokes could not be used as time series for the Recurrent Neural Networks, as they varied greatly in length. Consequently, the derived integral features were used as time series spanning across multiple strokes. However, the temporal difference between successive strokes is far from equidistant. Therefore, modeling the integral features of a stroke as an observation in a time series covering multiple consecutive strokes resulted in an inferior prediction quality compared with Feed-Forward Neural Network, XGBoost, Random Forest, and AutoML. Nevertheless, Recurrent Neural Networks should also be considered for the prediction of time-to-failure due to their ability of modeling temporal dependencies. However, the varying time differences between successive strokes should be explicitly taken into account.

13.6 Concluding Remarks

To conclude this chapter, we summarize its main contribution in the context of this thesis. Given that this chapter provides experimental results of the proposed generic, end-to-end time-to-failure prediction methodology of industrial machines on the basis of real-world monitoring data from a large-scale industrial press, this chapter contributes to research question **RQ B.3**. To demonstrate the effectiveness of the proposed methodology, we performed a five-fold cross-validation, assessing the prediction quality both at macro level as well as by class. For this purpose, six time-to-failure classes were defined, each representing a different lead time for impending machine failures. The macro level examination already showed that the multi-class model performed considerably better compared with the two binary labeling strategies. This insight was amplified when examining the prediction results on a per class basis. Here, the results revealed that the multi-class labeling consistently provided high quality for all time-to-failure classes. The multi-class labeling outperformed the other two labeling alternatives especially for the classes representing a short time-to-failure, which are the most critical from a practical point of view. When comparing the prediction methods, the boosting methods achieved the best prediction quality for both binary labeling alternatives, while the Feed-Forward Neural Network significantly surpassed the other prediction methods for the multi-class model. Thus, the Feed-Forward Neural Network provided the best overall predictions with an average accuracy, macro F1-score, and kappa value of 89.82%, 90.00%, and 87.24%, respectively. Downscaling the six time-to-failure classes to a binary prediction of whether or not the machine will fail within the next two days enhanced these numbers even further to 97.79%, 98.54%, and 94.03% in terms of accuracy, F1-score, and kappa, respectively. In addition, we have provided a set of potential hyperparameter configurations for hyperparameter tuning that have been proven to provide well-performing prediction models. Thus, practitioners can adopt these configurations for hyperparameter tuning in case of a lack of expert knowledge when applying the proposed methodology to other machines.

Although the presented results are auspicious, some limitations should still be mentioned. First, the prediction methodology was investigated only with respect to one type of machine, namely a large-scale industrial press. However, since we did not include any domain knowledge about presses in the proposed methodology, the methodology should be applicable to other industrial machines as well. In Chapter 12, we also discussed why integral features are universally applicable to industrial machines. Second, other techniques regarding prediction aggregation for the binary labeling procedures could lead

to different results. However, we have already tested another approach that considers the consistency of class label ones in the predictions of the binary models. Still, this approach performed inferior to using only the predicted time-to-failure class with the shortest time-to-failure. Nevertheless, future research should consider other aggregations and investigate their effects on the prediction quality. Finally, other prediction methods, such as Recurrent Neural Networks, could also be incorporated. However, the applicability of such methods is highly dependent on the data quality, making the applicability of the methodology less universal.

Chapter 14

On-line Update Strategies for Critical Event Prediction Models

Although the models presented so far have proven to predict critical events for real-world data with high accuracy, they lack a mechanism for updating the prediction models during runtime. However, such on-line updating mechanisms are required for the application of the proposed system model in real-world applications. The main reason for prediction models requiring such on-line updates is concept drift [WK96]. In machine learning, concept drift describes a shift in feature distribution over time due to a change in the underlying data generation process. This drift means that the mapping between features and the target value learned on old data no longer fits new incoming data. Therefore, prediction models should be updated according to an appropriate strategy to counter such concept drift. In general, there are two approaches to implement such updates. First, the model can be learned incrementally. However, such incremental learning procedures must be developed individually for each machine learning method, as they require modifications to the machine learning methods themselves. Second, the model can be completely re-trained. Here, the machine learning methods do not need to be adapted, but a trigger defining when the particular model should be re-trained is necessary.

In this chapter, we present on-line update strategies for prediction models using model re-training. Due to the availability of huge data sets for monitoring hard disk drives, in the following, we focus on predicting critical events for hard disk drives based on S.M.A.R.T. monitoring data.

To achieve proactivity based on S.M.A.R.T. monitoring data, several approaches have been proposed that apply machine learning algorithms with S.M.A.R.T. features as input to either predict the time-to-failure in the form of regression [CdPL+18, YHL+15, XWL+16] or to classify whether or not the hard disk drives will fail within a given prediction window [BGBW16, MHKD05, AJG+17]. However, most of these approaches do not reflect real-world applications in data centers, as they consider hard disk failure prediction only

as a static task where the prediction model is learned on a random subset of the available data, while the remaining subset of the data is used for testing. However, in real-world applications, temporal partitioning is required, as the prediction model must be learned on data that lies in the past. However, in the case of a random split between training and testing, this is not guaranteed. Furthermore, it is also not realistic to assess the prediction quality of a hard disk drive failure prediction model on the entire test set at once, as it should rather be analyzed on a regular basis. Moreover, model updates would be required if the observed prediction quality is not sufficient. Therefore, we address these requirements for the realistic application of hard disk drive failure prediction models by developing hard disk drive failure prediction models and presenting as well as comparing different update strategies.

The content of this chapter is based on our previous work, which was published as a full paper at the 20th IEEE International Conference on Machine Learning and Applications (ICMLA) [ZEK21]. The remainder of this chapter is organized as follows: Section 14.1 describes the models for hard disk drive failure prediction in general. Next, Section 14.2 introduces the update triggers, which determine when an update is required. Finally, Section 14.3 briefly summarizes this chapter and answers the associated research question.

14.1 Hard Disk Drive Failure Prediction

This section describes the design of the general hard disk drive failure prediction model. Therefore, we first outline the data preprocessing, then present the target label assignment, and finally describe the model learning workflow.

14.1.1 Data Set Generation

First, to learn a time-to-failure prediction model for hard disk drives, we built a data set comprising only S.M.A.R.T. measurements of a single hard disk drive model. This is due to the fact that different manufacturers may still define certain S.M.A.R.T. features differently. In addition, the patterns of degrading hard disk drives may vary from hard disk drive model to hard disk drive model, depending on their manufacturing characteristics. As M. Botezatu *et al.* have shown, prediction models learned on one hard disk drive model cannot be directly applied to other hard disk drive models [BGBW16].

After selecting a hard disk drive model (cf. Section 15.1), we narrowed the set of hard disk drives to the subset of hard disk drives exhibiting at least

five measurement samples. Hard disk drives with fewer than five S.M.A.R.T. measurements can be assumed to have behaved anomalously anyway.

Subsequently, we divided the data set into a training set and a testing set. Due to the fact that we are analyzing the effects of different update strategies, we split the initial data set according to the measurement time point. Accordingly, an initial time period is used for training, while all S.M.A.R.T. measurements thereafter are dedicated to on-line testing, thus also verifying whether or not the update trigger fires.

Existing research on hard disk drive failure prediction has already shown the favorable effects of S.M.A.R.T. feature selection [LSW+17, XXW+18]. Therefore, we employed the same feature selection as J. Xiao *et al.* [XXW+18], resulting in the feature set shown in Table 14.1. Here, *normalized* represents the normalization of the S.M.A.R.T. measurements performed by Backblaze, while *raw* illustrates the raw S.M.A.R.T. measurements. Upon defining these relevant S.M.A.R.T. features, we removed all daily measurement instances containing NaN values for any of the remaining S.M.A.R.T. features.

Table 14.1: The set of S.M.A.R.T. features selected for model learning following J. Xiao *et al.* [XXW+18].

S.M.A.R.T. Feature	Normalized	Raw
Read Error Rate	✓	
Reallocated Sectors Count	✓	✓
Seek Error Rate	✓	
Power-On Hours		✓
Power Cycle Count		✓
Runtime Bad Block		✓
End-to-End Error	✓	✓
Reported Uncorrectable Errors	✓	✓
High Fly Writes	✓	
Load Cycle Count	✓	✓
Current Pending Sector Count	✓	✓
Uncorrectable Sector Count	✓	✓
UltraDMA CRC Error Count		✓

Furthermore, since the recorded values of the various S.M.A.R.T. features show highly varying ranges, they are transformed using min-max normalization (cf. Section 12.2). Despite Backblaze having already applied a form of normalization, min-max normalization is required for both the normalized and

raw S.M.A.R.T. features. For this purpose, the maximum and minimum values only are determined on the training set to normalize all feature values in the training set. The same minimum and maximum values are also employed to normalize S.M.A.R.T. features in the test sets, since updating the minimum and maximum values would distort the patterns already learned. However, this may result in normalized values outside the range of $[0, 1]$ if the test sets include values lower than the minimum value of the training set or higher than the maximum value of the training set.

14.1.2 Time-to-Failure Prediction Window

In order to proactively predict impending hard disk drive failures, we model this task as a binary classification problem. A prediction window of ten days was chosen as this provides sufficient lead time for operators to back up the data stored on the hard disk drives and replace the hard disk drive in a timely manner. A prediction window of ten days refers to the classification task of predicting whether a particular hard disk drive will fail within the next ten days, considering the daily S.M.A.R.T. measurements. Therefore, we label all instances within ten days prior to the failure as "failed" (positive class), while all other instances are labeled as "good" (negative class). Note that this results in a highly imbalanced data set, with many more "good" instances than "failed" instances in the data set.

14.1.3 Model Learning

After selecting and normalizing the relevant S.M.A.R.T. features, the training set must be sampled. This sampling step is necessary since the original training set contains many more "good" instances than "failed" instances, which in turn leads to an inferior classification quality for the minority class, i.e., to a poor prediction of impending failures. Therefore, we limit the number of "good" instances in the training set with respect to the number of "failed" instances in the training set. More specifically, we define a parameter γ that specifies the maximum imbalance ratio. Given that T denotes the entire training set, with T_g being the subset of "good" instances and T_f being the subset of "failed" instances, the imbalance ratio ρ of the original training set is defined to be

$$\rho = \frac{|T_g|}{|T_f|}. \tag{14.1}$$

For the purpose of controlling the imbalance ratio, γ specifies the upper bound of ρ. If ρ is larger than γ, we randomly sample $\gamma \times |T_f|$ "good" instances from

the training set and create a new training set T^s that consists of this set of randomly sampled "good" instances T_g^s and the entire set of "failed" instances T_f. In this way, we ensure that the maximum imbalance ratio is

$$\rho \leq \gamma = \frac{|T_g^s|}{|T_f|}. \tag{14.2}$$

Using the sampled training set T^s, the different machine learning models are trained in the form of binary classification models. Here, the machine learning methods considered include logistic regression (LogReg), Support Vector Machine (SVM), Random Forest (RF) and XGBoost (XGB). Given that the main focus of this chapter is to analyze the effects of different update strategies, a fixed parametrization is used for each method instead of performing a time-consuming hyperparameter tuning. The hyperparameter settings for the different machine learning methods as well as the Python libraries used are presented in Table 14.2.

Table 14.2: The Python libraries used along with the parametrization of the machine learning methods.

Method	Library	Parameters
LogReg	`sklearn`	penalty = "l2", solver = "lbfgs"
SVM	`sklearn`	loss = "squared_hinge", penalty = "l2", tol = 1e-4, max_iter = 1000
RF	`sklearn`	criterion = "gini", num_estimators = 30, max_features = $\sqrt{\#\text{features}}$
XGB	`xgboost`	num_estimators = 30, objective = "binary:logistic", eval_metric = "logloss"

14.1.4 Prediction

Unlike typical, static machine learning applications, which learn the model on a given training set and evaluate it on a dedicated test set, here, the test set has to be partitioned into a number of batches to evaluate the impact of model update strategies at runtime. Therefore, the test set is divided into batches of equal size in time. Then, the model learned on the training set is applied to the first test batch. Once all instances in the test batch are predicted, the prediction quality obtained by the machine learning models is evaluated and

the update strategies are applied to determine whether or not the triggers fire. In case a trigger indicates the necessity of an update, the training set T^s is expanded to include the new "failed" instances as well as the undersampled "good" instances of the test batch. Note that the same γ used for the original training set is employed for adding the test batch to the subsequent training set. However, if the trigger does not signal the need for an update, the same model will be used for the next test batch until a trigger is fired.

14.2 Update Strategies

In order to adjust the machine learning models to changes in the data and drifting failure patterns, the models are re-trained if an update strategy triggers. For this purpose, this section describes the considered prediction quality measures as well as the different update triggers.

14.2.1 Prediction Quality Measures

While some update triggers do not depend on the achieved prediction quality of the critical event prediction models, analyzing the current predictive power and comparing it with previous test batches provides the most insight into the prediction quality of the models as well as potential concept drifts. Therefore, we compute two commonly used measures to evaluate hard disk drive failure prediction quality, namely the failure detection rate (FDR) and the false alarm rate (FAR). The false alarm rate compares the number of instances incorrectly predicted as "failed" with the number of instances correctly predicted as "good" plus the number of instances incorrectly predicted as "failed". Formally, this can be expressed as

$$\mathrm{FAR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}}, \tag{14.3}$$

where FP denotes the number of instances that were incorrectly predicted as "failed" and TN exemplifies the number of instances that were correctly predicted as "good".

On the contrary, the failure detection rate takes into account the number of instances correctly predicted as "failed" and puts it into relation to the sum of the number of instances correctly predicted as "failed" and the number of instances incorrectly predicted as "good". Consequently, the failure detection rate is defined as

$$\mathrm{FDR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{14.4}$$

where TP represents the number of instances that were correctly predicted as "failed" and FN reflects the number of instances that were predicted as "good", although "failed" would have been correct.

14.2.2 Update Triggers

In this section, we present four distinct triggers for updating machine learning models based on either temporal constraints or the achieved prediction quality.

14.2.2.1 Periodic

The first and most intuitive update trigger merely observes the time Δt elapsed since the last update and triggers as soon as it exceeds a specified temporal threshold λ_t. The periodic update trigger criterion is therefore defined as

$$\Delta t \geq \lambda_t. \tag{14.5}$$

When the temporal threshold is hit, the data gathered since the last update is added to the training set T^s and the "good" instances are undersampled as described in Section 14.1.4. Hence, the machine learning models are re-trained based on a periodic schedule.

14.2.2.2 Absolute Prediction Quality

In contrast to the periodic update trigger, this update trigger considers the achieved prediction quality of the current test batch. Based on a pre-defined absolute threshold λ_a for the evaluation measures, such as FDR or FAR, the trigger is fired in case the achieved measure $\overline{r}_i$ of the current test batch i drops below the threshold. Thus, the trigger criterion is defined by

$$\overline{r}_i < \lambda_a. \tag{14.6}$$

Analogous to the periodic update trigger, the new training set is updated to re-train the model based on the new data received since the last update.

14.2.2.3 Relative Prediction Quality

Whereas the prior update trigger only considers the prediction quality of the current test batch, this update trigger also accounts for the prediction quality of the previous test batch. To this end, the achieved prediction quality of the current test batch $\overline{r}_i$ is compared with the prediction quality of the previous

test batch $\overline{r}_{i-1}$ multiplied by a pre-defined threshold λ_r. Formally, the relative prediction quality update trigger criterion is defined by

$$\overline{r}_i < \lambda_r \cdot \overline{r}_{i-1}, \quad \text{with } 0 < \lambda_r \leq 1. \tag{14.7}$$

The procedure for updating the training set as soon as the trigger fires is the same as for the other update triggers.

14.2.2.4 Hoeffding Bound

This update trigger is related to the relative prediction quality update trigger, except that the Hoeffding bound trigger relies on statistics tailored to detect concept drift rather than expert knowledge for threshold determination. The Hoeffding bound is widely used to detect such concept drifts when applying machine learning models to streaming data [DH00, RPDJ12, XW16]. For the purpose of applying the Hoeffding bound to estimate the update requirement of machine learning models, the monitored prediction quality measure, such as FDR or FAR, is considered as the random variable r to be observed, with $\overline{r}$ denoting the sample mean of r and R representing the range of r, which is 1 in the case of a probability distribution and $\log c$ for information gain with c classes [DH00]. As only samples of the random variable r can be observed, a confidence interval based on the sample mean $\overline{r}$ is formed to determine statistically significant outliers. More specifically, the Hoeffding bound states that with a confidence level of $1 - \alpha$, the true mean of the observed random variable r is at least $\overline{r} - \epsilon$, where ϵ is defined by

$$\epsilon = \sqrt{\frac{R^2 \cdot \ln 1/\alpha}{2 \cdot n}}, \tag{14.8}$$

with n denoting the number of independent observations of r. In order to convert the Hoeffding bound into a trigger criterion, two successive test batches must be compared. If the prediction quality of the more recent test batch is lower than the prediction quality of the previous test batch by more than ϵ, a change in the data generation process is assumed, which in turn requires a model update. Otherwise, i.e., if the difference between the two test batches is less than or equal to ϵ or if the more recent test batch exhibits better prediction quality, no update is required. Therefore, the trigger criterion based on the Hoeffding barrier is defined as

$$\overline{r}_i < \overline{r}_{i-1} - \epsilon, \tag{14.9}$$

where $\overline{r}_i$ indicates the prediction quality of the current test batch and $\overline{r}_{i-1}$ denotes the prediction quality of the previous test batch.

14.3 Summary and Discussion

The main contribution of this chapter is the introduction of four update strategies for on-line application of critical event prediction models. Thus, the contribution of this chapter addresses research question **RQ B.4**. To this end, we first discussed the necessity of model updates during runtime. Subsequently, we focused on the domain of hard disk drive failure prediction, which has also been studied previously in Chapters 10 and 11. However, since the focus of this chapter is on the on-line update of critical event prediction models, we only used a binary model to predict hard disk drive failures in this chapter. In terms of update strategies, we described the extent to which the data set needs to be treated differently compared to typical, static machine learning applications in order to reflect the real-world application more realistically. First and foremost, the data set must be partitioned with respect to the monitoring time, rather than randomly. Furthermore, the test set needs to span a long time horizon so that it can be divided into multiple consecutive test batches, each covering the same time span. Regarding model learning based on highly imbalanced data, we also included a parameter to limit the maximum imbalance ratio in the training set. The four update triggers proposed in this chapter are based either on temporal limits or on the prediction quality achieved by the prediction models. While the first update trigger considers only the time elapsed since the last update, the second update trigger solely observes the prediction quality achieved on the current test batch. In contrast, the third update trigger compares the prediction quality of the current test batch with that of the previous test batch. Finally, the fourth update trigger again compares the prediction qualities of the current and previous test batches. However, instead of a fixed threshold, it employs the Hoeffding bound to determine whether the difference between the prediction qualities of the two test batches is statistically significant. Once an update trigger indicates the need for a model update, the new monitoring data collected since the last update is added to the old training set and used to re-train the prediction model. However, the same parameter for regulating the maximum imbalance ratio is applied as for the initial training set.

Although the update strategies were presented in the context of hard disk drive failure prediction and use prediction quality measures tailored to hard disk drive failure prediction, the general design can nevertheless be applied to other critical event prediction scenarios as well. For this purpose, only the observed evaluation measures within the update triggers need to be changed.

Chapter 15

Evaluation of Model Update Strategies

In this chapter, we evaluate the prediction quality of deploying on-line update strategies for critical event prediction models. To this end, we use a real-world data set from Backblaze [Bac20] that includes monitoring data of a large number of hard disk drives over several years. The particular subset of hard disk drive monitoring data provided by Backblaze used in these experiments is described in more detail in Section 15.1. Subsequently, Section 15.2 compares the prediction qualities achieved by the four model update strategies with each other and with the baseline that uses no update mechanisms at all. For this purpose, the update strategies are applied to four machine learning algorithms. In addition, we propose a novel evaluation measure, namely the ζ-value, to combine the failure detection rate and the false alarm rate into a single measure of prediction quality. In Section 15.3, we compare the prediction qualities of the different machine learning methods using only the best update strategy. Next, we summarize the main findings and discuss threats to validity in Section 15.4. Finally, we conclude this chapter in Section 15.5.

The contents of this chapter are based on our previous work, which was published as a full paper at the 20th IEEE International Conference on Machine Learning and Applications (ICMLA) [ZEK21].

15.1 Data Set

For the experiments conducted in this case study, we made use of the well-known data set from Backblaze [Bac20], which comprises a large number of different hard disk models and S.M.A.R.T. features monitored over a period of several years. However, since different manufacturers may define certain S.M.A.R.T. features differently and M. Botezatu *et al.* have shown that models learned on one hard disk drive model cannot be directly applied to other hard disk models [BGBW16], we utilized only one hard disk drive model, namely *STM4000DM000*. We chose this hard disk drive model due to the fact that hard disk drives of this model have been used for a comparatively long period

of time and it includes a large number of individual hard disk drives (i.e., a large number of unique serial numbers). More specifically, the monitoring data spans from February 1, 2017 to December 31, 2020. While we used the first six months of this data for model training, we evaluated the prediction quality for the remaining 41 months on a monthly basis. In total, the data set includes 35,170 unique hard disk drive serial numbers, out of which 2,246 hard disk drives failed during the monitoring period.

Yet, by analyzing the data set by measurement instance rather than hard disk drive serial number, the imbalance is even more pronounced. For model learning and evaluation, we labeled all instances of each hard disk drive that failed within the next ten days as positive, while we discarded all earlier instances of those hard disk drives. All other instances were labeled as negative. After this labeling procedure, the test set exhibits a prevalence, i.e., the ratio of positive instances compared to the total number of instances, of only 0.0639%.

Finally, we omitted the instances from the last ten days since we cannot guarantee that the hard disk drives in this last data period did not fail shortly after the monitoring period. Therefore, we cannot ensure the ground truth labels of these last measurement instances. As evaluation measures for prediction quality, we chose failure detection rate and false alarm rate (cf. Section 14.2.1).

15.2 Comparison of Update Strategies

As described in Section 15.1, the hard disk drive failure prediction models were initially trained on the first six months of data. The remaining data, namely the test set, were also sorted by time and separated into test batches, which each contained a single month. This means that after training the prediction models on the first six months, the seventh month was used to assess the prediction quality and to determine whether or not the update trigger fires. Depending on the decision of the particular update trigger, the old prediction model was either used again for the next month, or the prediction model was re-trained for the next test batch. This procedure was repeated for the entire test set, which comprised a total of 41 months.

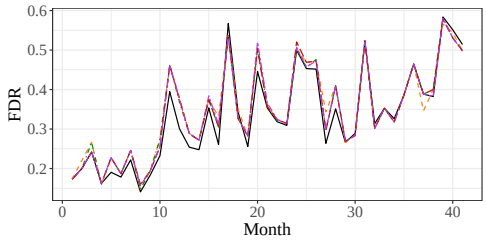
In order to deploy the four update triggers introduced in Section 14.2.2, their thresholds needed to be defined. Thus, we set the temporal threshold of the periodic update trigger to $\lambda_t = 1$ month, which corresponds to the duration of individual test batches. With respect to the update trigger based on the absolute prediction quality, we set two different thresholds for FAR and FDR as the machine learning algorithms yielded two different prediction quality patterns, as shown in the following. Therefore, we defined $\lambda_a^{\text{FAR}} = 0.015$ and

$\lambda_a^{\text{FDR}} = 0.35$ for logistic regression and SVM, whereas we specified $\lambda_a^{\text{FAR}} = 0.03$ and $\lambda_a^{\text{FDR}} = 0.45$ for Random Forest and XGBoost. Regarding the relative prediction quality update trigger, we set the relative threshold, which also considers the prediction quality of the previous test batch, to $\lambda_r = 0.95$. Finally, the confidence level of the Hoeffding bound was also specified to be 0.95. In other words, we set $\alpha = 0.05$ for the fourth update trigger.

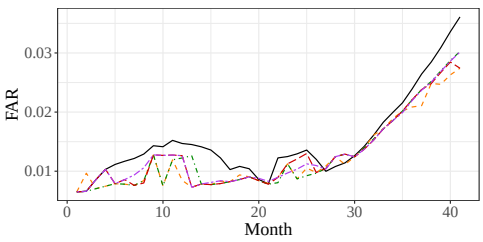
Figure 15.1 shows the prediction quality obtained by the four machine learning algorithms using no updates, the periodic update trigger, the absolute prediction quality update trigger, the relative prediction quality update trigger, and the Hoeffding bound update trigger represented as black (solid), orange (dashed), green (dot-dash), red (long dash), and purple (dash-dash) lines, respectively. The subfigures display the test batches on the horizontal axes, while the achieved predictive qualities, i.e., either with respect to FDR or FAR, are depicted on the vertical axes. Each row of subfigures illustrates the same machine learning algorithm, where the obtained FDR is always shown in the left column and the obtained FAR is always shown in the right column. In the interest of space, Figure 15.1 only depicts the achieved prediction quality using FDR as decision measure for the update triggers. However, Table 15.1 reports the overall achieved prediction quality over the entire test set of 41 months for both decision measures of the update triggers.

First, it is generally evident that the prediction quality of the static model exhibited a similar shape for all machine learning algorithms, even though shifted along the vertical axis. Concerning the FDR, the prediction quality is highly jagged. With respect to the FAR, the prediction quality is fairly constant for about the first half of the test period, while it increases drastically thereafter.

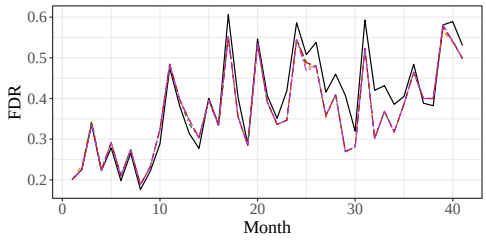
With respect to logistic regression, it is observable in Figure 15.1a that the achieved FDR improved for numerous test batches when using update strategies compared to using only the static prediction model without updates. However, the gain was rather small. A similar inference can be drawn when analyzing the achieved FAR of logistic regression (Figure 15.1b), as the update strategies showed the same pattern as the static model, but slightly improved. By contrast, considering the achieved FDR of the SVM (Figure 15.1c), the update strategies often yielded even worse prediction quality than the static model, while the FAR improved slightly by applying the update strategies, although it still followed the same pattern as without any update strategy over the entire test set (Figure 15.1d). Moreover, all update strategies performed almost equally for the SVM. A considerably different result was obtained by employing the update strategies in combination with Random Forest as prediction method. In this case, the different update strategies reached noticeably different FDRs,



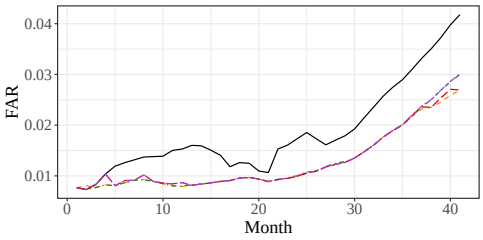
(**a**) FDR achieved using logistic regression.



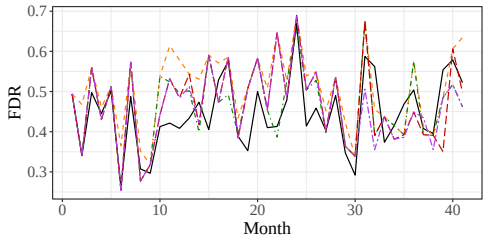
(**b**) FAR achieved using logistic regression.



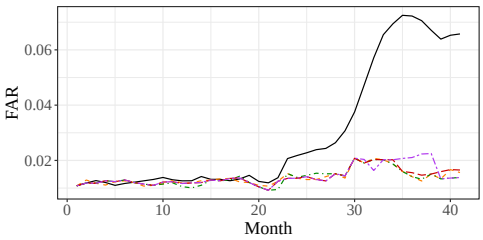
(**c**) FDR achieved using SVM.



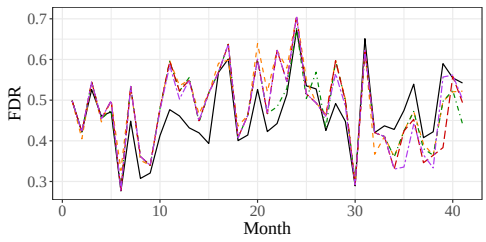
(**d**) FAR achieved using SVM.



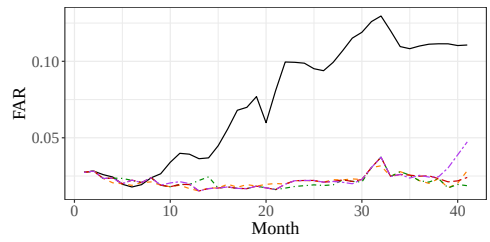
(**e**) FDR achieved using Random Forest.



(**f**) FAR achieved using Random Forest.



(**g**) FDR achieved using XGBoost.



(**h**) FAR achieved using XGBoost.

Figure 15.1: The comparison of the achieved FDR and FAR of the four update strategies using FDR as decision measure for each machine learning algorithm considered. The static machine learning model without any updates is displayed in black (solid), while the periodic, absolute prediction quality, relative prediction quality, and Hoeffding bound update strategies are presented as orange (dashed), green (dot-dash), red (long dash), and purple (dash-dash) lines, respectively.

though it is hard to determine which performed best due to the jagged shape (Figure 15.1e). However, it is evident that all update strategies outperformed the static prediction model for almost all test batches. The favorable effects of employing update strategies were even more significant with respect to the obtained FAR (Figure 15.1f), as the FAR rose only slightly toward the final test batches, while it rose manyfold in the case of no update strategy being employed. A similar pattern appears when using XGBoost as prediction algorithm, as the deployment of update strategies significantly improved the FDR of most test batches (Figure 15.1g), although an overall best update strategy could hardly be determined. Only for the last test batches, the static XGBoost model achieved a slightly higher FDR, although at the cost of a significantly worse FAR. On the contrary, the XGBoost models using update strategies could almost maintain their FAR (Figure 15.1h), while still achieving a competitive FDR.

In summary, the analysis of Figure 15.1 demonstrates that the application of update strategies significantly enhanced the FAR for all machine learning algorithms considered. In particular, the beneficial effect was considerably amplified for Random Forest and XGBoost. On the contrary, updating SVM models during runtime did not increase the achieved FDR and only marginally improved the obtained FDR for logistic regression. However, these two machine learning algorithms only achieved a lower FDR compared to Random Forest and XGBoost anyway. Using on-line update strategies for Random Forest and XGBoost not only reduced the FAR but also considerably increased the FDR.

Given the jagged nature of the achieved FDR for all prediction models and the close FAR of the prediction models using update strategies, we also assessed the overall prediction quality over the entire 41 months test set. To this end, Table 15.1 reports the obtained FDR as well as FAR for each combination of decision measure (either FDR or FAR), machine learning algorithm, and update strategy. On top of the achieved FDR and FAR, we also computed an aggregate measure that indicates the overall prediction quality for the prediction of hard disk drive failures, namely the ζ-value. Following the idea of the F1-score, which utilizes the harmonic mean to average precision and recall, we define the ζ-value as the harmonic mean of FDR and $1 - \text{FAR}$. The latter subtraction is necessary to transform the FAR into a measure indicating the best prediction quality for a value of 1 and the worst prediction quality for a value of 0. Formally, we define the ζ-value as

$$\zeta = 2 \cdot \frac{\text{FDR} \cdot (1 - \text{FAR})}{\text{FDR} + (1 - \text{FAR})}. \tag{15.1}$$

Moreover, Table 15.1 presents not only the obtained prediction quality, but also the number of model updates performed to achieve the respective prediction

quality. In this way, a trade-off between prediction quality and update cycles, which can be considered as update costs, can be made. Note that to determine the number of updates required, the update triggers were also applied after the last test batch to assess whether or not an update would be required to continue using the models. Hence, the maximum number of updates equals the number of test batches, which is 41.

At first, Table 15.1 demonstrates that the static machine learning models mostly performed worst with respect to the prediction quality measures FDR, FAR, and ζ-value, meanwhile naturally exhibiting the lowest number of updates. Only in the case of SVM as prediction method, the static model actually reached the highest FDR and ζ-value. Yet, as mentioned above, the prediction quality obtained by SVM can by no means keep up with Random Forest and XGBoost. A further observation is that for all machine learning algorithms, no updates were carried out when the Hoeffding bound was employed as update strategy in combination with FAR as decision measure, resulting in the same prediction quality as for the static model. The reason for this behavior is that the FAR changed slowly, so that the FAR of two successive test batches never exceeded an increase of ϵ. Apart from using SVM, the best FDR and ζ-values were always obtained by the periodic update strategy, as this performs a model update after each test batch and, thus, offers the highest potential for capturing relevant changes in the data. However, this update strategy also results in the highest number of required updates by far. While the maximum number of updates among all other settings was 21, the periodic update required 41 update cycles, constituting a costly update strategy.

Among the three update strategies relying on the prediction quality obtained, the application of FDR as decision measure seems to yield a higher ζ-value and, consequently, a higher prediction quality than the application of FAR as decision measure. Only for the absolute prediction quality trigger and the Hoeffding bound trigger or the relative prediction quality trigger and the Hoeffding bound trigger of the two inferior machine learning algorithms, i.e., logistic regression and SVM, respectively, using FAR delivered more accurate predictions than using FDR as decision measure. However, for the two superior prediction algorithms, namely Random Forest and XGBoost, employing the FDR as decision measure consistently provided a higher ζ-value.

Lastly, when comparing the ζ-values of the different update strategies for each machine learning method individually, it appears that they performed very similarly, despite varying considerably among the different machine learning algorithms. Whereas the update trigger based on the absolute prediction quality scored best for logistic regression and SVM, the update trigger based

Table 15.1: The overall prediction quality achieved across the entire 41 months test set and the number of updates performed (#U) for each combination of machine learning algorithm (ML), update strategy (US), and decision measure (DM). The best values per machine learning algorithm are highlighted in bold.

ML	US	DM	FDR [%]	FAR [%]	ζ [%]	#U
LogReg	Static	–	29.823	1.431	45.791	**0**
	Periodic	Time	**31.855**	1.155	**48.182**	41
	Absolute	FDR	31.545	1.184	47.823	21
		FAR	31.334	1.269	47.571	12
	Relative	FDR	31.462	1.215	47.724	19
		FAR	31.573	**1.115**	47.863	18
	Hoeffding	FDR	31.501	1.229	47.767	10
		FAR	29.823	1.431	45.791	**0**
SVM	Static	–	**35.687**	1.734	**52.359**	**0**
	Periodic	Time	34.535	**1.175**	51.183	41
	Absolute	FDR	34.513	1.193	51.157	19
		FAR	34.374	1.308	50.989	12
	Relative	FDR	34.480	1.195	51.120	19
		FAR	34.496	1.189	51.139	17
	Hoeffding	FDR	34.463	1.211	51.100	13
		FAR	**35.687**	1.734	**52.359**	**0**
RF	Static	–	44.175	2.627	60.777	**0**
	Periodic	Time	**50.991**	1.344	**67.233**	41
	Absolute	FDR	46.628	**1.330**	63.329	17
		FAR	43.228	1.475	60.091	1
	Relative	FDR	46.860	1.359	63.537	19
		FAR	46.168	1.336	62.902	13
	Hoeffding	FDR	46.661	1.390	63.347	14
		FAR	44.175	2.627	60.777	**0**
XGB	Static	–	45.952	6.665	61.584	**0**
	Periodic	Time	**49.347**	2.147	**65.608**	41
	Absolute	FDR	48.372	2.162	64.737	15
		FAR	45.670	2.300	62.244	3
	Relative	FDR	48.544	2.183	64.886	18
		FAR	46.611	**2.022**	63.170	9
	Hoeffding	FDR	48.538	2.275	64.861	13
		FAR	45.952	6.665	61.584	**0**

on the relative prediction quality reached the highest ζ-values for the two well-performing machine learning algorithms, namely Random Forest and XGBoost. However, the ζ-values obtained by the Hoeffding bound update trigger were only marginally lower compared with the update trigger based on the relative prediction quality. By contrast, the relative prediction quality update trigger required substantially more updates for all machine learning algorithms.

For this reason, we determined the Pareto-optimal configurations by aiming for a high ζ-value together with a low number of updates required. As a result, we came up with a set of seven configurations, from which we removed the edge cases, i.e., the configurations that resulted in a very low number of updates, but also a poor ζ-value, or requiring an update after each test batch, as those configurations result in too many updates and, hence, in high costs. This left only three configurations, namely the relative prediction quality update trigger using either FAR or FDR as decision measure, and the Hoeffding bound update trigger using FDR as decision measure. All three configurations employed XGBoost as prediction method. Despite using FAR as decision measure for the relative prediction quality update trigger required only 9 updates, its ζ-value was too low to be competitive to the other two solutions. In contrast, using the Hoeffding bound update trigger with FDR as decision measure required 13 updates and resulted in a ζ-value of 64.861%, whereas employing the relative prediction quality update trigger with FDR as decision measure required 18 updates and yielded a ζ-value of 64.886%. Consequently, the improvement in the ζ-value was only 0.025 percentage points, while 38.5% more updates were required. Thus, we conclude that the update strategies based on both the prediction quality of the current test batch as well as the prediction quality of the previous test batch performed best, with the Hoeffding bound achieving the better trade-off between prediction quality and number of updates required.

15.3 Comparison of Machine Learning Algorithms

In order to compare the prediction qualities of the different machine learning algorithms for each test batch, Figure 15.2 illustrates the ζ-values obtained using the Hoeffding bound update trigger with FDR as decision measure. The test batches are displayed on the horizontal axis, while the ζ-values achieved are depicted on the vertical axis. The prediction qualities of logistic regression, SVM, Random Forest and XGBoost are represented by orange (dashed), green (dot-dash), red (long dash), and purple (dash-dash) lines, respectively.

Figure 15.2 again illustrates the two different prediction quality patterns. Whereas Random Forest and XGBoost exhibited a similar pattern of prediction

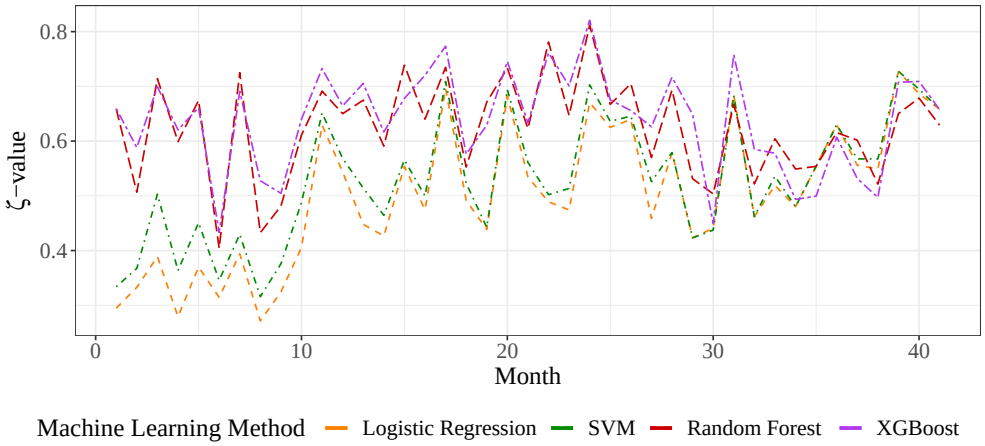


Figure 15.2: The comparison of machine learning algorithms using the Hoeffding bound update trigger relying on FDR as decision measure.

quality, logistic regression and SVM also performed fairly similarly, but considerably inferior to Random Forest and XGBoost. This is especially evident for about the first half of all test batches. Nevertheless, SVM appears to have performed slightly better than logistic regression. For the first few months of the test set, the ζ-values reached by Random Forest and XGBoost were about twice as high as those obtained by logistic regression and SVM. As time progressed, however, the ζ-values of the four methods converged against each other and fluctuated around a ζ-value of approximately 60%. More specifically, Random Forest and XGBoost always outperformed the other two prediction methods on the first 30 test batches, whereas logistic regression and SVM outperformed Random Forest and XGBoost on six and five of the last eleven test batches, respectively. Moreover, although the difference was rather small, XGBoost obtained a higher ζ-value than Random Forest for more than 60% of the test batches. Consequently, the time-dependent comparison of the machine learning algorithms supports the findings from Table 15.1, since XGBoost reached the best overall prediction quality, closely followed by Random Forest, with logistic regression and SVM far behind.

15.4 Discussion

This section briefly summarizes the main evaluation findings derived from the experimental results and discusses potential threats to validity.

15.4.1 Summary of Evaluation Findings

Relying on the results of the conducted experiments, we have extracted the following main findings:

(I) The application of update strategies substantially improved the prediction quality of Random Forest and XGBoost for on-line application.

(II) While the periodic update trigger typically yielded the highest prediction quality, it also required the most updates and, thus, resulted in the highest update costs. In contrast, the update triggers considering both current and previous prediction quality achieved a better trade-off between prediction quality and number of updates required. Overall, in our case study, the Hoeffding bound emerged as the best update strategy with respect to the trade-off between prediction quality and number of updates required.

(III) The higher the FDR achieved, the higher the FAR. Accordingly, more aggressive models correctly detected more hard disk drives with impending failures, but also more often falsely raised alarms for hard disk drives that actually did not fail within the defined prediction window.

(IV) XGBoost provided the most accurate hard disk drive failure prediction, followed closely by Random Forest. In contrast, logistic regression and SVM could not keep up with the other two machine learning algorithms.

15.4.2 Threats to Validity

The main contribution of this chapter is the comparison of the update strategies for critical event prediction models introduced in Chapter 14. Therefore, our goal was not to optimize individual machine learning algorithms and their parametrizations. Instead, we examined the effect of the presented update triggers on the prediction quality of the considered machine learning algorithms, using only a fixed set of hyperparameters. In future research, the update triggers could also consider adapting the hyperparameters of the machine learning algorithms. However, this was beyond the scope of this thesis.

Moreover, we did not conduct a comprehensive feature examination, but relied on the feature selection of J. Xiao *et al.* [XXW+18]. Although they analyzed the same hard disk drive model type of the same data set and, therefore, their feature selection should be adoptable, other feature selection methods could suggest another feature subset and, consequently, yield different results.

Despite the achieved FDR appearing relatively low for all prediction methods, it should be noted that the prevalence in the test set is only 0.0639%. Thus, the test set is highly imbalanced. In addition, the S.M.A.R.T. features were

monitored only on a daily basis, resulting in a low feature resolution. Taking these facts into account, the achieved prediction quality is satisfactory.

Furthermore, we used only one hard disk drive model in this case study. The ranking of update strategies and the ranking of Random Forest and XGBoost were very close. When using a different hard disk drive model or data from a completely different domain, these rankings might turn out to be different.

The thresholds for the update triggers were determined based on a small experiment using only a subset of the data and expert knowledge. A broad parameter study, however, could provide different insights in terms of the ranking of update triggers and machine learning algorithms.

Finally, the update triggers based on prediction quality currently only consider the achieved FDR or FAR. Future research might also consider combining them by, for instance, using the proposed ζ-value as decision measure for the update triggers. In addition, the update triggers including previous prediction qualities could be extended to consider not only the directly preceding prediction quality, but several, e.g., to identify a trend pattern therein.

15.5 Concluding Remarks

In this chapter, we evaluated the prediction quality obtained by applying the proposed update strategies using four different machine learning algorithms and compared their prediction qualities with each other and with the static baseline that used no updates at all. Thus, this chapter contributes to answering research question **RQ B.4**. To demonstrate the necessity of on-line update strategies for critical event prediction models, we used a large real-world data set of hard disk drive monitoring data. While we used the first six months of data for initial learning of the prediction models, we evaluated the prediction quality for each month in the test set, which spans a total of 41 months. To evaluate the prediction quality for hard disk drive failure prediction with a single measure, we introduced the ζ-value as the harmonic mean of the failure detection rate and the false alarm rate. Furthermore, we reported the achieved prediction quality both on a monthly basis and aggregated over the entire test set. Although the application of update strategies could not improve the prediction quality for SVM, on-line model updating considerably enhanced the prediction quality of logistic regression, Random Forest, and XGBoost. Moreover, the prediction quality of SVM could not keep up with Random Forest and XGBoost anyway. Regarding update strategies, the periodic update trigger provided the highest overall prediction quality, but at the cost of requiring by far the most updates. Thus, in practice, the periodic update trigger results in

the highest application costs. Therefore, we determined the Pareto-optimal configurations, aiming for a high ζ-value and a low number of required updates. Here, we found that the overall best trade-off between prediction quality and update cycles was the deployment of XGBoost using the Hoeffding bound as update trigger with the failure detection rate as decision measure. This configuration performed 13 model re-trainings, whereas the maximum number of updates would have been 41, and reached a ζ-value of 64.861%, compared to 61.584% obtained by the static baseline model, improving the ζ-value achieved by almost 3.3 percentage points.

Part IV

Conclusions

Chapter 16

Conclusion and Outlook

This chapter concludes this thesis by briefly summarizing its contributions in Section 16.1. Furthermore, in Section 16.2, it provides an outlook on potential future work that could build upon the contributions of this thesis.

16.1 Thesis Summary

The contributions of this thesis address two main research goals, both dealing with the prediction of critical events. First, we approach **Goal A**, namely improving the accuracy of automated time series forecasting over state-of-the-art methods through novel hybrid forecasting methods, in Part II by introducing a novel hybrid, component-based forecasting method and two forecasting method recommendation systems. In Part III of this thesis, we address **Goal B**, namely the development of generalizable end-to-end workflows for modeling, detecting, and predicting failures of technical systems with minimized expert knowledge required, by proposing an end-to-end workflow for detecting machine tool anomalies in rotating machines, presenting different modeling alternatives for time-to-failure prediction, introducing a generalizable workflow for predicting failures of industrial machines in multiple time-to-failure windows, and examining the effects of different update strategies for critical event prediction models for on-line application. To provide more details on each of the presented contributions related to the two major research goals, we briefly summarize them here.

Contribution 1: **System Model for Critical Event Prediction**

As a first contribution, this thesis presents a generic system model for critical event prediction to overcome the shortcoming of most approaches in the literature, namely the highly problem-tailored solutions. The design of this system model is based on the LRA-M loop of Self-Aware Computing Systems. To process data for critical event prediction, the system model distinguishes between two types of input data: univariate

and multivariate. For univariate input data, time series forecasting is applied in combination with meta-learning to provide threshold-based prediction of critical events. In the case of multivariate input data, a more complex model learning is performed, which may also include time series forecasting to estimate future sensor readings as features for critical event prediction by means of classification or regression models. Finally, a feedback loop is incorporated to update the model during runtime.

Contribution 2: **Component-based Forecasting Method**

This contribution addresses research questions **RQ A.1** and **RQ A.2** by introducing a novel hybrid time series forecasting method for seasonal time series based on time series decomposition, called Telescope (cf. Chapter 4). To this end, we present a method to automatically determine the frequency of seasonal time series. After decomposing the time series according to the structural time series model, the trend and seasonal components are forecast using different methods that show their strength for this particular type of time series. In order to obtain additional information for remainder learning, Telescope derives additional categorical information, which is predicted using a neural network-based method. Finally, XGBoost is employed to combine the forecasts of each component and learn the remainder component. Chapter 5 presents experimental results on Telescope and compares its forecasting quality and the time required for model learning and forecasting with eight state-of-the-art forecasting methods. The results demonstrate Telescope's superiority in terms of both forecast accuracy and required runtime. Moreover, Telescope also achieved an improved robustness when analyzing the variation in forecast accuracy and runtime across the different time series in the data set. Finally, we deployed Telescope in the task of virtual machine auto-scaling and showed its substantial improvement with respect to average response time and percentage of service level objective violations.

Contribution 3: **Forecasting Method Recommendation Frameworks**

In addition to Telescope, this thesis also proposes two recommendation systems for time series forecasting methods (cf. Chapter 6), addressing research questions **RQ A.3** and **RQ A.4**. While one of these recommendation frameworks merely uses the historical observations of the time series to be forecast and divides them into an internal training and validation part to estimate which forecasting method performs best for that particular time series, the other relies on a large and diverse data set of time series.

More specifically, the second framework for recommending forecasting methods trains a machine learning model for each considered forecasting method to learn the dependency between the time series characteristics and the suitability of the particular forecasting method. Lastly, we propose the combination of ensemble forecasting and forecasting method recommendation. That is, multiple forecasting methods are applied and combined, but only if their respective weights satisfy a certain activation function. In Chapter 7, we demonstrate the effectiveness and superiority of the proposed approaches by comparing them with state-of-the-art individual forecasting methods as well as the state-of-the-art time series forecasting method recommendation system.

Contribution 4: **Critical Event Detection for Machine Tools**

Apart from time series forecasting-based critical event prediction, this thesis further contributes to the field of critical event prediction by proposing end-to-end workflows for detecting and predicting such events using classification and regression models. This contribution tackles research question **RQ B.1** by introducing a generalizable end-to-end workflow for anomaly detection of rotary machine tools. To this end, Chapter 8 presents the automated five-step workflow starting with raw data processing, through phase segmentation, data sampling, and feature extraction, to machine learning-based anomaly detection. This workflow requires only the number of different production steps performed by the machine as input and is highly generalizable because it is based only on standard machine monitoring data. To evaluate the critical event detection quality of this workflow, we built a real-world setup using an industrial CNC machine and emulated an anomalous machine tool condition by attaching an unbalance to the spindle (cf. Chapter 9). The results demonstrate the effectiveness of the proposed end-to-end workflow, even though conventional frequency analysis techniques were not able to distinguish very well between normal and anomalous machine tool behavior.

Contribution 5: **Critical Event Prediction Modeling Alternatives**

In order to not only detect critical events, but to predict them at an early stage, Chapter 10 presents several alternatives for modeling critical events for proactive prediction, thus providing an answer to research question **RQ B.2**. A major challenge in critical event prediction is class imbalance, as critical events typically occur much less frequently than normal conditions. To this end, we compare the impact of different oversam-

pling strategies on the overall critical event prediction quality in the use case of hard disk drive failure prediction. In addition, we assess the prediction quality of multi-class models and compare them as well as their downscaled version with the binary prediction models. Besides the classification models, we train regression models to estimate the time-to-failure as a continuous variable rather than in discretized time-to-failure windows. Finally, we employ univariate time series forecasting to forecast each feature in the data set and examine the effect of using such forecast features compared with using only the current features. Chapter 11 presents the results of the experiments conducted, which show that multi-class labeling achieves superior prediction quality for critical events compared with oversampled binary models, both in terms of granularity and prediction quality. Furthermore, the results prove that integrating forecasting to estimate future feature observations significantly enhances the critical event prediction quality.

Contribution 6: **Critical Event Prediction for Industrial Machines**

This contribution addresses research question **RQ B.3** by proposing a novel generalizable end-to-end workflow for critical event prediction of industrial machines based on sensor monitoring data (cf. Chapter 12). Due to the end-to-end design of the workflow, it does not rely on any domain knowledge, except for the grouping of sensors into four categories. Therefore, the workflow is also highly generalizable. Specifically, the workflow follows an automated four-step process, starting with the extraction of general-purpose features from the sensor monitoring data, continuing with the processing of the extracted features and the assignment of the target classes to their respective training instances, and ending with the model learning for critical event prediction including grid-search-based hyperparameter tuning. To predict impending critical events in multiple time windows, the workflow employs multiple time-to-failure classes. For this purpose, three different labeling and, hence, modeling alternatives and four different machine learning algorithms are deployed. In order to evaluate the proposed critical event prediction workflow for industrial machines, we conducted experiments using a real-world data set of operational monitoring data from a large-scale industrial press. The results of this case study are reported in Chapter 13 and show the effectiveness of the proposed workflow for both multi-class prediction of critical events in multiple time windows and downscaled binary prediction of impending critical events.

Contribution 7: **Update Strategies for Critical Event Prediction Models**

The last contribution of this thesis addresses research question **RQ B.4** by introducing four different update triggers for on-line re-training of critical event prediction models (cf. Chapter 14). These update triggers take into account either the time elapsed since the last update or the prediction quality achieved on the current or current as well as previous test batches. In Chapter 15, we compare the prediction qualities achieved by the four update triggers as well as that obtained by a static model without updates using four different machine learning models for predicting impending hard disk drive failures. To assess the overall prediction quality, we introduce a novel measure for hard disk drive failure prediction that combines the two most commonly used measures in this area into a single measure. The results clearly demonstrate the necessity of model updates during on-line application, with update triggers that take into account the prediction qualities obtained on the current and previous test batches resulting in the overall best trade-off between prediction quality and number of updates required.

We strongly believe that the contributions provided by this thesis are major impetus for both the academic research community as well as practitioners. First, to the best of our knowledge, we are the first to propose a fully automated, end-to-end, hybrid, component-based forecasting method for seasonal time series that also incorporates time series preprocessing, such as frequency estimation, anomaly removal, and transformation. By combining reliably high forecast accuracy and reliably low time-to-result, it offers many new opportunities in tasks that require accurate forecasts within a fixed period of time in order to take timely countermeasures. Moreover, the promising results of the recommendation systems for forecasting methods offer new opportunities to advance forecasting performance for all types of time series, not only for seasonal ones. Furthermore, we were the first to highlight the shortcomings of the up to then state-of-the-art recommendation system for forecasting methods.

Regarding the contributions of **Goal B**, we have already worked closely with industrial partners, which supports the practical relevance of the contributions of this thesis. The automated end-to-end design of the proposed workflows that do not require distinct domain or expert knowledge constitutes a milestone in bridging the gap between academic theory and industrial application. To emphasize the practical relevance of this thesis, the workflow for the prediction of critical events in industrial machines is currently being operationalized in a real-world production system.

16.2 Outlook

This section discusses future work that can draw on the contributions of this thesis. Although this thesis addresses several aspects of data processing and modeling related to critical event prediction, we still identify open challenges that could further improve the proposed system model. In addition, we present future use cases where the proposed system model could be applied.

16.2.1 Future Work

Here, we present future work that can build upon this thesis to further extend the presented system model and its applicability.

Time Series Imputation: A typical challenge when dealing with real-world time series are missing data. Such gaps can occur, for instance, due to transmission errors, measurement failures, or monitored values outside the considered range. However, most time series analysis and forecasting methods require complete time series as input and cannot handle missing values. Therefore, such missing values must be imputed. For this purpose, we have introduced an intuitive imputation algorithm that considers both the seasonal pattern and the local trend of the time series around the missing value (cf. Section 6.2). However, deep learning architectures have recently proven beneficial for time series imputation [MCJ+20, CKPW20, ZZC+21]. Therefore, future work should consider deep bidirectional Recurrent Neural Networks or Generative Adversarial Networks for this task.

Recurrent Neural Networks: Recurrent Neural Networks could also be used for time series forecasting in general as well as for classification-based critical event prediction. Although initial experiments with Long Short-Term Memory networks and Gated Recurrent Units did not improve the prediction quality of the proposed system model (cf. Section 13.5), we are confident that the integration of such neural networks may enhance the overall system model and its predictive power. We assume that the issue in applying Recurrent Neural Networks in the use case of Chapter 13 is the irregularity of the recordings, whereby the time intervals between two measurements are arbitrarily large or small. In the case of uniform time intervals between measurements, or the integration of appropriate mechanisms to cope with these irregularities, we strongly expect that this type of neural networks will be beneficial, as they are specifically designed to process data with temporal dependencies.

Meta-Learning for Prediction Method Selection: In Chapters 6 and 7, we have proposed meta-learning frameworks for the recommendation of time series forecasting methods and demonstrated their superiority over the application of state-of-the-art individual forecasting methods, respectively. However, regarding the application of machine learning methods for classification-based prediction of critical events, we have built models for each considered method and evaluated their prediction quality. Here, the knowledge acquisition mode [VGCBS04] could also be adopted to learn a dependency between the prediction quality obtained by a particular machine learning method and the characteristics of the training data and their correlation with the target variable. This would reduce the overall runtime of the system model, since only the presumably most suitable machine learning method would be applied. In case the considered task is not time-critical, this time saving could also be exploited to further tune the hyperparameters of the machine learning model.

Updating Training Data: The update strategies presented in Chapter 14 only consider the point in time when the critical event prediction models need to be re-trained. With respect to neural networks, future research could also investigate mechanisms to distinguish when the weights of the network only need to be updated and when it is necessary to completely re-train the neural network. Furthermore, future work on update strategies should also consider the training data used to update the critical event prediction models. In this thesis, the training data set was increased over time, as the new data was merely added to the previous training data set, while maintaining a maximum imbalance ratio. However, new updating strategies can also explore what part of the previous training data set should be omitted or when it is beneficial to even completely discard the previous training data set and use only the newly acquired data. By selecting only the most representative data for model updating, model training time should decrease and predictive power should increase as old patterns that are no longer present in the new incoming data are forgotten.

Root Cause Analysis: The current version of the system model predicts impending critical events using threshold-based time series forecasting or classification as well as regression models. This already provides helpful information for service or machine operators, but does not point to the specific component that will cause the critical event. For this purpose, future research could focus on root cause analysis methods as an extension of the proposed system model to additionally identify the reason for the impending critical event. This would further assist the operator in practice, as the time-consuming process of

troubleshooting would be handled directly by the system model and would no longer need to be performed manually.

Planning of Countermeasures: After predicting critical events and potentially also their cause, planning and scheduling appropriate countermeasures would also constitute an interesting research topic. Such a mechanism represents the next step in the LRA-M loop of Self-Aware Computing Systems, namely reasoning. The planning and scheduling of countermeasures for critical events can be considered as an optimization problem with many different sources of costs, such as acquisition costs, personnel costs, and downtime costs. Possible approaches to tackle this challenge could rely on rules (e.g., [KDM+18]), models (e.g., [PKWB17]), goals (e.g., [KM07]), or utility functions (e.g., [VSSB13]).

16.2.2 Future Application Scenarios

Although the proposed meta self-aware system model for critical event prediction was mainly applied to technical systems, it is not limited to this domain. For instance, it could also be applied to predict impending critical events of natural systems. One such application scenario is the prediction of winter mortality of bee colonies. To this end, colony weight would be monitored along with weather information. These data could be processed by the system model to derive early indicators of increased risk of winter mortality. Another application scenario in the field of nature and biology is the early prediction of human heart failures. For this purpose, we have already conducted preliminary experiments, in which we analyzed electrocardiogram data to detect and predict different types of arrhythmias. To this end, various features need to be computed in the preprocessing step of the system model in order to obtain a representative insight into the human heart. Although these experiments are beyond the scope of this thesis, we can state that our experiments were not only able to detect arrhythmias, but were also able to predict them several heartbeats in advance. Thus, employing the proposed meta self-aware system model to predict critical events for such application scenarios seems promising.

List of Figures

List of Tables

Bibliography

[AA14] Ratnadip Adhikari and RK Agrawal. A Combination of Artificial Neural Network and Random Walk Models for Financial Time Series Forecasting. *Neural Computing and Applications*, 24(6):1441–1449, 2014. [see page 64]

[ACAK01] Monica Adya, Fred Collopy, J Scott Armstrong, and Miles Kennedy. Automatic Identification of Time Series Features for Rule-based Forecasting. *International Journal of Forecasting*, 17(2):143–157, 2001. [see page 59]

[AFS+15] Jaouher Ben Ali, Nader Fnaiech, Lotfi Saidi, Brigitte Chebel-Morello, and Farhat Fnaiech. Application of Empirical Mode Decomposition and Artificial Neural Network for Automatic Bearing Fault Diagnosis based on Vibration Signals. *Applied Acoustics*, 89:16–27, 2015. [see page 244]

[AJG+17] Nicolas Aussel, Samuel Jaulin, Guillaume Gandon, Yohan Petetin, Eriza Fazli, and Sophie Chabridon. Predictive Models of Hard Drive Failures based on Operational Data. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 619–625. IEEE, 2017. [see page 269]

[AKA97] Bay Arinze, Seung-Lae Kim, and Murugan Anandarajan. Combining and Selecting Forecasting Models using Rule based Induction. *Computers and Operations Research*, 24(5):423–433, 1997. [see pages 60 and 142]

[AN00] Vassilis Assimakopoulos and Konstantinos Nikolopoulos. The Theta Model: A Decomposition Approach to Forecasting. *International Journal of Forecasting*, 16(4):521–530, 2000. [see page 59]

[Ari94] Bay Arinze. Selecting Appropriate Forecasting Models using Rule Induction. *Omega*, 22(6):647–658, 1994. [see page 60]

[ASK13] M Amarnath, V Sugumaran, and Hemantha Kumar. Exploiting Sound Signals for Fault Diagnosis of Bearings using Decision Tree. *Measurement*, 46(3):1250–1256, 2013. [see page 244]

[AVK15] Ratnadip Adhikari, Ghanshyam Verma, and Ina Khandelwal. A Model Ranking based Selective Ensemble Approach for Time Series Forecasting. *Procedia Computer Science*, 48:14–21, 2015. [see page 56]

[Bac20] Backblaze. Hard Drive Data and Stats. `https://www.backblaze.com/b2/hard-drive-test-data.html`, 2020. Accessed: 2021-07-15. [see page 279]

[Bau16] André Bauer. Design and Evaluation of a Proactive, Application-Aware Elasticity Mechanism. Master Thesis, Julius-Maximilians-Universität Würzburg, 2016. [see page 137]

[Bau20] André Bauer. *Automated Hybrid Time Series Forecasting: Design, Benchmarking, and Use Cases*. Phd thesis, University of Würzburg, Germany, 2020. [see page 97]

[BC64] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964. [see page 35]

[BCLW18] Leo Breiman, Adele Cutler, Andy Liaw, and Matthew Wiener. *Breiman and Cutler's Random Forests for Classification and Regression*, 2018. R package: `https://cran.r-project.org/web/packages/randomForest/randomForest.pdf`. Accessed: 2021-07-15. [see pages 205 and 227]

[BG69] John M Bates and Clive WJ Granger. The Combination of Forecasts. *Journal of the Operational Research Society*, 20(4):451–468, 1969. [see pages 56 and 142]

[BGBW16] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. Predicting Disk Replacement Towards Reliable Data Centers. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 39–48. ACM, 2016. [see pages 77, 269, 270, and 279]

[BHB16] Christoph Bergmeir, Rob J Hyndman, and José M Benítez. Bagging Exponential Smoothing Methods using STL Decomposition and Box–Cox Transformation. *International Journal of Forecasting*, 32(2):303–312, 2016. [see page 67]

[BJ70] George EP Box and Gwilym M Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1970. [see pages 34 and 35]

[BP70] George EP Box and David A Pierce. Distribution of Residual Autocorrelations in Autoregressive-integrated Moving Average Time Series Models. *Journal of the American Statistical Association*, 65(332):1509–1526, 1970. [see page 23]

[Bre96] Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996. [see page 40]

[Bre01] Leo Breiman. Random Forests. *Machine learning*, 45(1):5–32, 2001. [see pages 40 and 196]

[Bro56] Robert G Brown. Exponential Smoothing for Predicting Demand. *Operations Research*, 5(1):145–145, 1956. [see page 33]

[Bro17] Jason Brownlee. Feature Selection for Time Series Forecasting with Python. `http://machinelearningmastery.com/feature-selection-time-series-forecasting-python/`, 2017. Accessed: 2021-07-15. [see page 122]

[BSdM+18] Marcia Baptista, Shankar Sankararaman, Ivo P de Medeiros, Cairo Nascimento Jr, Helmut Prendinger, and Elsa MP Henriques. Forecasting Fault Events for Predictive Maintenance using Data-driven Techniques and ARMA Modeling. *Computers & Industrial Engineering*, 115:41–53, 2018. [see page 71]

[BSNA21] Xanthi Bampoula, Georgios Siaterlis, Nikolaos Nikolakis, and Kosmas Alexopoulos. A Deep Learning Model for Predictive Maintenance in Cyber-Physical Production Systems using LSTM Autoencoders. *Sensors*, 21(3):972, 2021. [see page 85]

[Bun75] Derek W Bunn. A Bayesian Approach to the Linear Combination of Forecasts. *Journal of the Operational Research Society*, 26(2):325–329, 1975. [see page 56]

[BZE+21] André Bauer, Marwin Züfle, Simon Eismann, Johannes Grohmann, Nikolas Herbst, and Samuel Kounev. Libra: A Benchmark for Time Series Forecasting Methods. In *Proceedings of the 12th ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 189–200. ACM, 2021. [see page xxv]

[BZG+20] André Bauer, Marwin Züfle, Johannes Grohmann, Norbert Schmitt, Nikolas Herbst, and Samuel Kounev. An Automated Forecasting Framework based on Method Recommendation for Seasonal Time Series. In *Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 48–55. ACM, 2020. [see pages xxvi, 97, and 143]

[BZH+20a] André Bauer, Marwin Züfle, Nikolas Herbst, Samuel Kounev, and Valentin Curtef. Telescope: An Automatic Feature Extraction and Transformation Approach for Time Series Forecasting on a Level-Playing Field. In *Proceedings of the 36th International Conference on Data Engineering (ICDE)*, pages 1902–1905. IEEE, 2020. [see pages xxvi and 97]

[BZH+20b] André Bauer, Marwin Züfle, Nikolas Herbst, Albin Zehe, Andreas Hotho, and Samuel Kounev. Time Series Forecasting for Self-Aware Systems. *Proceedings of the IEEE*, 108(7):1068–1093, 2020. [see pages xxv and 97]

[BZHK19] André Bauer, Marwin Züfle, Nikolas Herbst, and Samuel Kounev. Best Practices for Time Series Forecasting. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 255–256. IEEE, 2019. [see page xxvii]

[CA92] Fred Collopy and J Scott Armstrong. Rule-based Forecasting: Development and Validation of an Expert Systems Approach to Combining Time Series Extrapolations. *Management Science*, 38(10):1394–1414, 1992. [see pages 59, 142, and 152]

[CA18] François Chollet and Joseph J Allaire. *Deep Learning mit R und Keras: Das Praxis-Handbuch von den Entwicklern von Keras und RStudio*. MITP-Verlags GmbH & Co. KG, 2018. [see page 251]

[CAS12] Thanyalak Chalermarrewong, Tiranee Achalakul, and Simon Chong Wee See. Failure Prediction of Data Centers using Time Series and Fault Tree Analysis. In *2012 IEEE 18th International*

Conference on Parallel and Distributed Systems, pages 794–799. IEEE, 2012. [see page 70]

[CBHK02] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002. [see page 219]

[CC08] Jonathan D Cryer and Kung-Sik Chan. *Time Series Analysis: With Applications in R*. Springer, 2008. [see page 17]

[CC20] João R Campos and Ernesto Costa. Fault Injection to Generate Failure Data for Failure Prediction: A Case Study. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 115–126. IEEE, 2020. [see page 83]

[CCMT90] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. STL: A Seasonal-trend Decomposition Procedure based on Loess. *Journal of Official Statistics*, 6(1):3–73, 1990. [see pages 29 and 113]

[CdPL+18] Iago C Chaves, Manoel Rui P de Paula, Lucas GM Leite, Joao Paulo P Gomes, and Javam C Machado. Hard Disk Drive Failure Prediction Method Based On A Bayesian Network. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2018. [see pages 79 and 269]

[Cen20] Centre for Aviation. European Airline Capacity Planning: Uncertainty is Increasing. `https://centreforaviation.com/analysis/reports/european-airline-capacity-planning-uncertainty-is-increasing-533951`, 2020. Accessed: 2021-06-20. [see page 2]

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKIDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016. [see pages 41 and 196]

[CH19] Tianqi Chen and Tong He. *Extreme Gradient Boosting*, 2019. R package: `https://cran.r-project.org/web/packages/xgboost/xgboost.pdf`. Accessed: 2021-07-15. [see page 205]

[Che19] Tao Chen. All Versus One: An Empirical Comparison on Retrained and Incremental Machine Learning for Modeling Performance of Adaptable Software. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 157–168. IEEE, 2019. [see page 90]

[CJF+20] Haoshu Cai, Xiaodong Jia, Jianshe Feng, Wenzhe Li, Laura Pahren, and Jay Lee. A Similarity based Methodology for Machine Prognostics by using Kernel Two Sample Test. *ISA Tansactions*, 103:112–121, 2020. [see page 73]

[CKH+15] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive. `www.cs.ucr.edu/~eamonn/time_series_data/`, 2015. Accessed: 2021-07-15. [see page 162]

[CKPW20] Zhiyong Cui, Ruimin Ke, Ziyuan Pu, and Yinhai Wang. Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Forecasting Network-wide Traffic State with Missing Values. *Transportation Research Part C: Emerging Technologies*, 118:102674, 2020. [see page 298]

[Cle89] Robert T Clemen. Combining Forecasts: A Review and Annotated Bibliography. *International Journal of Forecasting*, 5(4):559–583, 1989. [see pages 56 and 142]

[CLWN13] Hong Cao, Xiao-Li Li, David Yew-Kwong Woon, and See-Kiong Ng. Integrated Oversampling for Imbalanced Time Series Classification. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2809–2822, 2013. [see pages 218 and 219]

[CMJG+18] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 233–248. Springer, 2018. [see pages 89 and 91]

[CPEM05] Antonio J Conejo, Miguel A Plazas, Rosa Espinola, and Ana B Molina. Day-ahead Electricity Price Forecasting using the Wavelet Transform and ARIMA Models. *IEEE Transactions on Power Systems*, 20(2):1035–1042, 2005. [see pages 64 and 142]

[CSGK19] Christos Constantinides, Stavros Shiaeles, Bogdan Ghita, and Nicholas Kolokotronis. A Novel Online Incremental Learning Intrusion Prevention System. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–6. IEEE, 2019. [see page 90]

[CV95] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995. [see pages 38 and 196]

[DBK+97] Harris Drucker, Chris JC Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support Vector Regression Machines. *Advances in Neural Information Processing Systems*, 9:155–161, 1997. [see page 40]

[DC03] Christopher P Diehl and Gert Cauwenberghs. SVM Incremental Learning, Adaptation and Optimization. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2685–2690. IEEE, 2003. [see page 86]

[DH00] Pedro Domingos and Geoff Hulten. Mining High-Speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM, 2000. [see pages 8 and 276]

[DHW21] Guangzhong Dong, Weiji Han, and Yujie Wang. Dynamic Bayesian Network based Lithium-ion Battery Health Prognosis for Electric Vehicles. *IEEE Transactions on Industrial Electronics*, 68(11):10949–10958, 2021. [see page 244]

[DJLW14] Hancheng Dong, Xiaoning Jin, Yangbing Lou, and Changhong Wang. Lithium-ion Battery State of Health Monitoring and Remaining Useful Life Prediction based on Support Vector Regression-particle Filter. *Journal of Power Sources*, 271:114–123, 2014. [see page 244]

[DKW17] Matthew Dixon, Diego Klabjan, and Lan Wei. *OSTSC*, 2017. R package: `https://cran.r-project.org/web/packages/OSTSC/OSTSC.pdf`. Accessed: 2020-01-28. [see page 227]

[DLHS11] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. Forecasting Time Series with Complex Seasonal Patterns us-

ing Exponential Smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011. [see pages 35 and 122]

[DMBT00] Lilian M De Menezes, Derek W Bunn, and James W Taylor. Review of Guidelines for the Use of Combined Forecasts. *European Journal of Operational Research*, 120(1):190–204, 2000. [see pages 56 and 142]

[DPCB15] Andrea Dal Pozzolo, Olivier Caelen, and Gianluca Bontempi. *unbalanced*, 2015. R package: `https://cran.r-project.org/web/packages/unbalanced/unbalanced.pdf`. Accessed: 2020-01-28. [see page 227]

[DSJ20] Maren David Dangut, Zakwan Skaf, and Ian K Jennions. Rare Failure Prediction using an Integrated Auto-encoder and Bidirectional Gated Recurrent Unit Network. *IFAC-PapersOnLine*, 53(3):276–282, 2020. [see page 83]

[DYW14] Chongli Di, Xiaohua Yang, and Xiaochao Wang. A Four-stage Hybrid Model for Hydrological Time Series Forecasting. *PloS one*, 9(8):e104663, 2014. [see page 66]

[Eme] Emerson. Emerson Tackles Costly Downtime Losses in Industrial Process Facilities, Launches Reliability Management Consulting Service. `https://www.emerson.com/en-us/news/corporate/reliability-consulting`. Accessed: 2020-12-01. [see page 243]

[Eme16] Emerson. 2016 Cost of Data Center Outages. Technical report, Emerson Network Power & Ponemon Institute, 2016. `https://www.emerson.com/en-us/news/corporate/network-power-study`. Accessed: 2021-07-15. [see page 1]

[EW74] AG Evans and SM Wiederhorn. Proof Testing of Ceramic Materials–An Analytical Basis for Failure Prediction. *International Journal of Fracture*, 10(3):379–392, 1974. [see page 4]

[FML+21] Nils Finke, Marisa Mohr, Alexander Lontke, Marwin Züfle, Samuel Kounev, and Ralf Möller. Recommendations for Data-Driven Degradation Estimation with Case Studies from Manufacturing and Dry-Bulk Shipping. In *Proceeedings of the 15th International Conference on Research Challenges in Information Science (RCIS)*, pages 189–204. Springer, 2021. [see page xxv]

[For65] Edward W Forgy. Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications. *Biometrics*, 21:768–769, 1965. [see page 37]

[FS97] Yoav Freund and Robert E Schapire. A Decision-theoretic Generalization of On-line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. [see page 42]

[Fuc20] Hannah Fuchs. Making Sense of Coronavirus Infection Statistics. *Deutsche Welle*, 2020. `https://www.dw.com/en/making-sense-of-coronavirus-infection-statistics/a-55274839`. Accessed: 2021-07-15. [see page 2]

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*. MIT Press, 2016. [see pages 17 and 42]

[GCS16] Xiaojie Guo, Liang Chen, and Changqing Shen. Hierarchical Adaptive Deep Convolution Neural Network and its Application to Bearing Fault Diagnosis. *Measurement*, 93:490–502, 2016. [see page 244]

[Geb06] Nagi Gebraeel. Sensory-Updated Residual Life Distributions for Components with Exponential Degradation Patterns. *Transactions on Automation Science and Engineering*, 3(4):382–393, 2006. [see page 73]

[GEB+19] Johannes Grohmann, Simon Eismann, Andre Bauer, Marwin Züfle, Nikolas Herbst, and Samuel Kounev. Utilizing Clustering to Optimize Resource Demand Estimation Approaches. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, page 134–139. IEEE, 2019. [see page xxvii]

[GJ85] Everette S Gardner Jr. Exponential Smoothing: The State of the Art. *Journal of Forecasting*, 4(1):1–28, 1985. [see page 33]

[GLJ+17] Liang Guo, Naipeng Li, Feng Jia, Yaguo Lei, and Jing Lin. A recurrent neural network based health indicator for remaining useful life prediction of bearings. *Neurocomputing*, 240:98–109, 2017. [see page 78]

[GPR10] Fausto P García, Diego J Pedregal, and Clive Roberts. Time Series Methods Applied to Failure Prediction and Detection. *Reliability Engineering & System Safety*, 95(6):698–703, 2010. [see page 70]

[GTLT+19] Antonio Galicia, R Talavera-Llames, A Troncoso, Irena Koprinska, and Francisco Martínez-Álvarez. Multi-step Forecasting for Big Data Time Series based on Ensemble Learning. *Knowledge-Based Systems*, 163:830–841, 2019. [see page 58]

[GWM10] Jose A Guajardo, Richard Weber, and Jaime Miranda. A Model Updating Strategy for Predicting Time Series with Seasonal Patterns. *Applied Soft Computing*, 10(1):276–283, 2010. [see page 87]

[HA18] Rob J Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2018. [see pages 17, 24, and 36]

[HAB+18] Rob J Hyndman, George Athanasopoulos, Christoph Bergmeir, Gabriel Caceres, Leanne Chhay, Mitchell O'Hara-Wild, Fotios Petropoulos, Slava Razbash, Earo Wang, and Farah Yasmeen. *forecast: Forecasting Functions for Time Series and Linear Models*, 2018. R package: `http://pkg.robjhyndman.com/forecast`. Accessed: 2021-07-15. [see pages 36, 122, and 162]

[HCLX11] Haibo He, Sheng Chen, Kang Li, and Xin Xu. Incremental Learning from Stream Data. *IEEE Transactions on Neural Networks*, 22(12):1901–1914, 2011. [see page 88]

[HHKA14] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. *Concurrency and Computation: Practice and Experience*, 26(12):2053–2078, 2014. [see page 123]

[Hil00] Robert C Hilborn. *Chaos and Nonlinear Dynamics: An Introduction for Scientists and Engineers*. Oxford University Press on Demand, 2000. [see page 24]

[HK06] Rob J Hyndman and Anne B Koehler. Another Look at Measures of Forecast Accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. [see page 46]

[HK19a] Duy-Tang Hoang and Hee-Jun Kang. A Survey on Deep Learning based Bearing Fault Diagnosis. *Neurocomputing*, 335:327–335, 2019. [see page 244]

[HK19b] Duy-Tang Hoang and Hee-Jun Kang. Rolling Element Bearing Fault Diagnosis using Convolutional Neural Network and Vibration Image. *Cognitive Systems Research*, 53:42–50, 2019. [see page 82]

[HKOS08] Rob J Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with Exponential Smoothing: The State Space Approach*. Springer, 2008. [see page 33]

[HKV19] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature, 2019. [see page 251]

[HLYJ19] Te Han, Chao Liu, Wenguang Yang, and Dongxiang Jiang. Learning Transferable Features in Deep Convolutional Neural Networks for Diagnosing Unseen Machine Conditions. *ISA Transactions*, 93:341–353, 2019. [see page 82]

[HMSZ20] Jiangpeng He, Runyu Mao, Zeman Shao, and Fengqing Zhu. Incremental Learning in Online Scenario. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13926–13935. IEEE, 2020. [see page 91]

[Ho95] Tin Kam Ho. Random Decision Forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, pages 278–282. IEEE, 1995. [see page 40]

[Hol57] Charles C Holt. Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages. *ONR Memorandum*, 52, 1957. [see page 33]

[HRBA+18] Jürgen Herp, Mohammad H Ramezani, Martin Bach-Andersen, Niels L Pedersen, and Esmaeil S Nadimi. Bayesian State Prediction of Wind Turbine Bearing Failure. *Renewable Energy*, 116:164–172, 2018. [see pages 4 and 9]

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural Computation*, 9(8):1735–1780, 1997. [see page 46]

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009. [see pages 17, 37, and 258]

[HVK17] Jordan Hochenbaum, Owen S Vallis, and Arun Kejariwal. Automatic Anomaly Detection in the Cloud Via Statistical Learning. *arXiv preprint arXiv:1704.07706*, 2017. [see page 105]

[HWOP11] Wei He, Nicholas Williard, Michael Osterman, and Michael Pecht. Prognostics of Lithium-ion Batteries based on Dempster–Shafer Theory and the Bayesian Monte Carlo Method. *Journal of Power Sources*, 196(23):10314–10321, 2011. [see page 244]

[HY18] Rob J Hyndman and Yangzhuoran Yang. tsdl: Time Series Data Library, 2018. v0.1.0. `https://pkg.yangzhuoranyang.com/tsdl/`. Accessed: 2021-07-15. [see page 122]

[Hyn11] Rob J Hyndman. Cyclic and Seasonal Time Series. `https://robjhyndman.com/hyndsight/cyclicts/`, 2011. Accessed: 2021-07-15. [see page 24]

[Hyn18] Rob J Hyndman. *Data for "Forecasting: Principles and Practice"*, 2018. R package: `https://pkg.robjhyndman.com/fpp2-package`. Accessed: 2021-07-15. [see page 122]

[HZJH20] Shao Haidong, Ding Ziyang, Cheng Junsheng, and Jiang Hongkai. Intelligent Fault Diagnosis Among Different Rotating Machines using Novel Stacked Transfer Auto-encoder Optimized by PSO. *ISA Transactions*, 105:308–319, 2020. [see page 84]

[HZS99] Joseph L Hellerstein, Fan Zhang, and Perwez Shahabuddin. An Approach to Predictive Detection for Service Management. In *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, pages 309–322. IEEE, 1999. [see pages 3 and 68]

[HZSG19] Zhiqiang Huo, Yu Zhang, Lei Shu, and Michael Gallimore. A New Bearing Fault Diagnosis Method based on Fine-to-coarse Multiscale Permutation Entropy, Laplacian Score and SVM. *IEEE Access*, 7:17050–17066, 2019. [see page 244]

[HZZ+20] Stefan Herrnleben, Bernd Zeidler, Marwin Züfle, Christian Krupitzer, and Samuel Kounev. A Concept for Crowd-sensed Prediction of Mobile Network Connectivity. In *GI/ITG Workshop on Machine Learning in the Context of Communication Networks 2020*, 2020. [see page xxvii]

[IBM16] IBM Research Editorial Staff. Predicting Disk Failures for Reliable Clouds. `https://www.ibm.com/blogs/research/2016/08/predicting-disk-failures-reliable-clouds/`, 2016. Accessed: 2021-07-15. [see page 1]

[IHL18] Md Mojahidul Islam, Guoqing Hu, and Qianbo Liu. Online Model Updating and Dynamic Learning Rate-based Robust Object Tracking. *Sensors*, 18(7):2046, 2018. [see page 89]

[Jai16] Aarshay Jain. Complete guide to parameter tuning in xgboost with codes in python. `https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/`, 2016. Accessed: 2021-07-15. [see page 256]

[JPBŚ20] Andrzej Janusz, Mateusz Przyborowski, Piotr Biczyk, and Dominik Ślęzak. Network Device Workload Prediction: A Data Mining Challenge at Knowledge Pit. In *15th Conference on Computer Science and Information Systems* (*FedCSIS*), pages 77–80. IEEE, 2020. [see page 173]

[JYJ13] Daniel Jacobson, Danny Yuan, and Neeraj Joshi. Scryer: Netflix's Predictive Auto Scaling Engine. `https://netflixtechblog.com/scryer-netflixs-predictive-auto-scaling-engine-a3f8fc922270`, 2013. Accessed: 2021-06-20. [see page 2]

[KC03] Jeffrey O Kephart and David M Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003. [see page 17]

[KCF16] Mirko Kück, Sven F Crone, and Michael Freitag. Meta-learning with Neural Networks and Landmarking for Forecasting Model Selection an Empirical Evaluation of Different Feature Sets Applied to Industry Data. In *International Joint Conference on Neural Networks* (*IJCNN*), pages 1499–1506. IEEE, 2016. [see page 62]

[KDM+18] Christian Krupitzer, Guido Drechsel, Deborah Mateja, Alina Pollkäsener, Florian Schrage, Timo Sturm, Aleksandar Tomasovic, and Christian Becker. Using Spreadsheet-defined Rules for Reasoning in Self-adaptive Systems. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 289–294. IEEE, 2018. [see pages 8, 20, and 300]

[KK01] John F Kolen and Stefan C Kremer. *A Field Guide to Dynamical Recurrent Networks*. John Wiley & Sons, 2001. [see page 45]

[KKZ+00] Tiruvalam Natarajan Krishnamurti, CM Kishtawal, Zhan Zhang, Timothy LaRow, David Bachiochi, Eric Williford, Sulochana Gadgil, and Sajani Surendran. Multimodel Ensemble Forecasts for Weather and Seasonal Climate. *Journal of Climate*, 13(23):4196–4216, 2000. [see page 142]

[KLB+17] Samuel Kounev, Peter Lewis, Kirstie Bellman, Nelly Bencomo, Javier Camara, Ada Diaconescu, Lukas Esterle, Kurt Geihs, Holger Giese, Sebastian Götz, Paola Inverardi, Jeffrey Kephart, and Andrea Zisman. The Notion of Self-Aware Computing. In Samuel Kounev, Jeffrey O Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu, editors, *Self-Aware Computing Systems*, pages 3–16. Springer, 2017. [see pages 6, 17, 18, and 19]

[Kle20] Klein, Andy (Backblaze). Backblaze Hard Drive Stats for 2019. `https://www.backblaze.com/blog/hard-drive-stats-for-2019/`, 2020. Accessed: 2021-07-15. [see page 13]

[KM07] Jeff Kramer and Jeff Magee. Self-Managed Systems: An Architectural Challenge. In *Proceedings of the Future of Software Engineering (FOSE)*, pages 259–268. IEEE, 2007. [see pages 8, 20, and 300]

[KML+20] Christian Krupitzer, Sebastian Müller, Veronika Lesch, Marwin Züfle, Janick Edinger, Alexander Lemken, Dominik Schäfer, Samuel Kounev, and Christian Becker. A Survey on Human Machine Interaction in Industry 4.0. Technical report, University of Würzburg and University of Mannheim and ioxp GmbH and Syntax Systems GmbH, 2020. arXiv:2002.01025v1. [see page xxvii]

[KMN19] Dominique Knittel, Hamid Makich, and Mohammed Nouari. Milling Diagnosis using Artificial Intelligence Approaches. *Mechanics & Industry*, 20(8):809, 2019. [see page 81]

[KRV+15] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A Survey on Engineering Approaches for Self-adaptive Systems. *Pervasive and Mobile Computing*, 17:184–206, 2015. [see pages 2 and 141]

[KT99] Alexandros Kalousis and Theoharis Theoharis. Noemon: Design, Implementation and Performance Results of an Intelligent Assistant for Classifier Selection. *Intelligent Data Analysis*, 3(5):319–337, 1999. [see page 60]

[Ku17] Jin-Hee Ku. A Study on the Machine Learning Model for Product Faulty Prediction in Internet of Things Environment. *Journal of Convergence for Information Technology*, 7(1):55–60, 2017. [see page 4]

[KWZ+20] Christian Krupitzer, Tim Wagenhals, Marwin Züfle, Veronika Lesch, Dominik Schäfer, Amin Mozaffarin, Janick Edinger, Christian Becker, and Samuel Kounev. A Survey on Predictive Maintenance for Industry 4.0. Technical report, University of Würzburg and University of Mannheim and Syntax Systems GmbH and MOZYS Engineering GmbH, 2020. arXiv:2002.08224. [see page xxvii]

[KYO] KYOS Energy Consulting. Auto Scaling Production Services on Titus. `https://www.kyos.com/faq/what-is-power-scheduling/`. Accessed: 2021-07-15. [see page 1]

[LCTH00] Bo Li, M-Y Chow, Yodyium Tipsuwan, and James C Hung. Neural-network-based Motor Rolling Bearing Fault Diagnosis. *IEEE Transactions on Industrial Electronics*, 47(5):1060–1069, 2000. [see page 244]

[Les17] Veronika Lesch. Self-Aware Multidimensional Auto-Scaling. Master Thesis, Julius-Maximilians-Universität Würzburg, 2017. [see page 137]

[LFH+18] Zefang Li, Huajing Fang, Ming Huang, Ying Wei, and Linlan Zhang. Data-driven Bearing Fault Identification using Im-

proved Hidden Markov Model and Self-organizing Map. *Computers & Industrial Engineering*, 116:37–46, 2018. [see page 244]

[LG10] Christiane Lemke and Bogdan Gabrys. Meta-learning for Time Series Forecasting and Forecast Combination. *Neurocomputing*, 73(10-12):2006–2016, 2010. [see pages 21, 61, and 142]

[LHS+17] Eric Gordon Lamb, Caspar A Hallmann, Martin Sorg, Eelke Jongejans, Henk Siepel, Nick Hofland, Heinz Schwan, Werner Stenmans, Andreas Müller, Hubert Sumser, Thomas Hörren, Dave Goulson, and Hans de Kroon. More than 75 Percent Decline over 27 Years in Total Flying Insect Biomass in Protected Areas. *PLoS ONE*, 12(10):e0185809, 2017. [see page 14]

[LHY+20] Weichao Luo, Tianliang Hu, Yingxin Ye, Chengrui Zhang, and Yongli Wei. A Hybrid Predictive Maintenance Approach for CNC Machine Tool Driven by Digital Twin. *Robotics and Computer-Integrated Manufacturing*, 65:101974, 2020. [see page 84]

[LJ+18] Andrew Leung, Amit Joshi, et al. Auto Scaling Production Services on Titus. `https://netflixtechblog.com/auto-scaling-production-services-on-titus-1f3cd49f5cd7`, 2018. Accessed: 2021-07-15. [see page 2]

[LJJ+14] Jing Li, Xinpu Ji, Yuhan Jia, Bingpeng Zhu, Gang Wang, Zhongwei Li, and Xiaoguang Liu. Hard Drive Failure Prediction using Classification and Regression Trees. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 383–394. IEEE, 2014. [see page 264]

[LK77] J Richard Landis and Gary G Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1):159–174, 1977. [see pages 51, 52, and 260]

[LLG+16] Yaguo Lei, Naipeng Li, Szymon Gontarz, Jing Lin, Stanislaw Radkowski, and Jacek Dybala. A Model-based Method for Remaining Useful Life Prediction of Machinery. *IEEE Transactions on Reliability*, 65(3):1314–1326, 2016. [see page 77]

[Llo82] Stuart Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. [see pages 37 and 190]

[LLT+17] Shaobo Li, Guokai Liu, Xianghong Tang, Jianguang Lu, and Jianjun Hu. An Ensemble Deep Convolutional Neural Network Model with Improved DS Evidence Fusion for Bearing Fault Diagnosis. *Sensors*, 17(8):1729, 2017. [see page 244]

[LMC12] Imene Lahyani, Wafa Makki, and Christophe Chassot. Failure Prediction for Publish/subscribe System on MANET. In *IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 98–100. IEEE, 2012. [see page 70]

[LSG+10] Jie Liu, Abhinav Saxena, Kai Goebel, Bhaskar Saha, and Wilson Wang. An Adaptive Recurrent Neural Network for Remaining Useful Life Prediction of Lithium-ion Batteries. In *Annual Conference of the Prognostics and Health Management Society*, pages 1–9. PHM Society, 2010. [see page 244]

[LSW+17] Jing Li, Rebecca J Stones, Gang Wang, Xiaoguang Liu, Zhongwei Li, and Ming Xu. Hard Drive Failure Prediction using Decision Trees. *Reliability Engineering & System Safety*, 164:55–65, 2017. [see pages 78 and 271]

[LTZ+14] Nian Liu, Qingfeng Tang, Jianhua Zhang, Wei Fan, and Jie Liu. A Hybrid Forecasting Model with Parameter Optimization for Short-term Load Forecasting of Micro-grids. *Applied Energy*, 129:336–345, 2014. [see pages 66 and 142]

[LVT16] Asma Ladj, Christophe Varnier, and Fatima Benbouzid-Si Tayeb. IPro-GA: An Integrated Prognostic based GA for Scheduling Jobs and Predictive Maintenance in a Single Multifunctional Machine. *IFAC-PapersOnLine*, 49(12):1821–1826, 2016. [see page 244]

[LW13] Wenzhu Liao and Ying Wang. Data-driven Machinery Prognostics Approach using in a Predictive Maintenance Model. *Journal of Computers*, 8(1):225–231, 2013. [see page 244]

[LWP12] Wenzhu Liao, Ying Wang, and Ershun Pan. Single-machine-based Predictive Maintenance Model Considering Intelligent Machinery Prognostics. *The International Journal of Advanced Manufacturing Technology*, 63(1-4):51–63, 2012. [see pages 3 and 71]

[LWZ+14] Jay Lee, Fangji Wu, Wenyu Zhao, Masoud Ghaffari, Linxia Liao, and David Siegel. Prognostics and Health Management Design for Rotary Machinery Systems–Reviews, Methodology and Applications. *Mechanical Systems and Signal Processing*, 42(1-2):314–334, 2014. [see pages 4 and 9]

[LZXS07] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure Prediction in IBM Bluegene/L Event Logs. In *Seventh IEEE International Conference on Data Mining (ICDM)*, pages 583–588. IEEE, 2007. [see page 73]

[Mat75] Brian W Matthews. Comparison of the Predicted and Observed Secondary Structure of T4 Phage Lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975. [see page 51]

[McH12] Mary L McHugh. Interrater Reliability: The Kappa Statistic. *Biochemia Medica*, 22(3):276–282, 2012. [see page 257]

[MCJ+20] Jun Ma, Jack CP Cheng, Feifeng Jiang, Weiwei Chen, Mingzhu Wang, and Chong Zhai. A Bi-directional Missing Data Imputation Scheme based on LSTM and Transfer Learning for Building Energy Data. *Energy and Buildings*, 216:109941, 2020. [see page 298]

[MDH+19] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, Friedrich Leisch, Chih-Chung Chang, and Chih-Chen Lin. *Misc Functions of the Department of Statistics, Probability Theory Group*, 2019. R package: `https://cran.r-project.org/web//packages/e1071/e1071.pdf`. Accessed: 2021-07-16. [see page 205]

[Med21] Valerie Medleva. GameStop Stock Forecast: Will the Market Bonanza Last? `https://capital.com/gamestop-stock-forecast-will-the-market-bonanza-last`, 2021. Accessed: 2021-07-16. [see page 2]

[MH00] Spyros Makridakis and Michele Hibon. The M3-Competition: Results, Conclusions and Implications. *International Journal of Forecasting*, 16(4):451–476, 2000. [see pages 67 and 162]

[MH18] Michael Mulders and Mark Haarman. Predictive Maintenance 4.0 - Beyond the Hype: PdM 4.0 Delivers Results. Technical

report, PricewaterhouseCoopers and Mainnovation, 2018. [see pages 1, 243, and 244]

[MHKD05] Joseph F Murray, Gordon F Hughes, and Kenneth Kreutz-Delgado. Machine Learning Methods for Predicting Failures in Hard Drives: A Multiple-instance Application. *Journal of Machine Learning Research*, 6(5):783–816, 2005. [see pages 75, 225, and 269]

[MMAHT20] Pablo Montero-Manso, George Athanasopoulos, Rob J Hyndman, and Thiyanga S Talagala. FFORMA: Feature-based Forecast Model Averaging. *International Journal of Forecasting*, 36(1):86–92, 2020. [see page 59]

[Mob02] R Keith Mobley. *An Introduction to Predictive Maintenance*. Elsevier, 2002. [see page 243]

[MSR16] Ganapathy Mahalakshmi, S Sridevi, and Shyamsundar Rajaram. A Survey on Forecasting of Time Series Data. In *International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE)*, pages 1–8. IEEE, 2016. [see page 95]

[MSvdMW04] Christian Müller-Schloer, Christoph von der Malsburg, and Rolf P Würt. Organic computing. *Informatik-Spektrum*, 27(4):332–336, 2004. [see page 17]

[MXC+13] Qiang Miao, Lei Xie, Hengjuan Cui, Wei Liang, and Michael Pecht. Remaining Useful Life Prediction of Lithium-ion Battery with Unscented Particle Filter Technique. *Microelectronics Reliability*, 53(6):805–810, 2013. [see pages 9 and 75]

[NAL17] Ali Yadavar Nikravesh, Samuel A Ajila, and Chung-Horng Lung. An Autonomic Prediction Suite for Cloud Resource Provisioning. *Journal of Cloud Computing*, 6(1):1–20, 2017. [see page 142]

[NF20] Meenakshi Narayan and Ann Majewicz Fey. Developing a Novel Force Forecasting Technique for Early Prediction of Critical Events in Robotics. *PloS one*, 15(5):e0230009, 2020. [see page 72]

[NG74] Paul Newbold and Clive WJ Granger. Experience with Forecasting Univariate Time Series and the Combination of Forecasts. *Journal of the Royal Statistical Society. Series A (General)*, 137(2):131–146, 1974. [see page 142]

[NXT14] Selina SY Ng, Yinjiao Xing, and Kwok L Tsui. A Naive Bayes Model for Robust Remaining Useful Life Prediction of Lithium-ion Battery. *Applied Energy*, 118:114–123, 2014. [see page 244]

[OP95] Erik Ottem and Judy Plummer. Playing it SMART: The Emergence of Reliability Prediction Technology. Technical report, Seagate Technology Paper, 1995. [see page 216]

[OSN80] Shunichiro Oe, Takashi Soeda, and Takayoshi Nakamizo. A Method of Predicting Failure or Life for Stochastic Systems by using Autoregressive Models. *International Journal of Systems Science*, 11(10):1177–1188, 1980. [see page 68]

[PAAL13] Mahardhika Pratama, Sreenatha G Anavatti, Plamen P Angelov, and Edwin Lughofer. PANFIS: A Novel Incremental Learning Machine. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):55–68, 2013. [see page 88]

[PDSA15] Alessandro Pellegrini, Pierangelo Di Sanzo, and Dimiter R Avresky. A Machine Learning-based Framework for Building Application Failure Prediction Models. In *IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 1072–1081. IEEE, 2015. [see page 76]

[Peg69] C Carl Pegels. Exponential Forecasting: Some New Variations. *Management Science*, 15(5):311–315, 1969. [see page 33]

[PKWB17] Martin Pfannemüller, Christian Krupitzer, Markus Weckesser, and Christian Becker. A Dynamic Software Product Line Approach for Adaptation Planning in Autonomic Computing Systems. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, pages 247–254. IEEE, 2017. [see pages 8, 20, and 300]

[PL04] Ricardo BC Prudêncio and Teresa B Ludermir. Meta-learning Approaches to Selecting Time Series Models. *Neurocomputing*, 61:121–137, 2004. [see page 60]

[PL05] Ping-Feng Pai and Chih-Sheng Lin. A Hybrid ARIMA and Support Vector Machines Model in Stock Price Forecasting. *Omega*, 33(6):497–505, 2005. [see page 63]

[PL21] Matt Phillips and Taylor Lorenz. 'Dumb Money' Is on GameStop, and It's Beating Wall Street at Its Own Game. *New York Times*, 2021. `https://www.nytimes.com/2021/01/27/business/gamestop-wall-street-bets.html`. Accessed: 2021-07-16. [see page 2]

[PVHG13] Teerat Pitakrat, Andre Van Hoorn, and Lars Grunske. A Comparison of Machine Learning Algorithms for Proactive Hard Disk Drive Failure Detection. In *Proceedings of the 4th international ACM Sigsoft symposium on Architecting critical systems*, pages 1–10. ACM, 2013. [see pages 75, 217, and 226]

[PWB07] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure Trends in a Large Disk Drive Population. In *5th USENIX Conference on File and Storage Technologies (FAST)*, pages 17–29. USENIX, 2007. [see page 217]

[QLDL17] Xiwen Qin, Qiaoling Li, Xiaogang Dong, and Siqi Lv. The Fault Diagnosis of Rolling Bearing based on Ensemble Empirical Mode Decomposition and Random Forest. *Shock and Vibration*, 2017:1–9, 2017. [see page 244]

[QLLY03] Hai Qiu, Jay Lee, Jing Lin, and Gang Yu. Robust Performance Degradation Assessment Methods for Enhanced Rolling Element Bearing Prognostics. *Advanced Engineering Informatics*, 17(3-4):127–140, 2003. [see page 77]

[QLMF19] Jiantao Qu, Feng Liu, Yuxiang Ma, and Jiaming Fan. A Neural-network-based Method for RUL Prediction and SOH Monitoring of Lithium-ion Battery. *IEEE Access*, 7:87178–87191, 2019. [see page 244]

[R C18] R Core Team. *The R Datasets Package*, 2018. R package: `https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html`. Accessed: 2021-07-16. [see pages 122 and 123]

[Rei21] Andrej Reisin. Modelle mit Unsicherheiten. *tagesschau*, 2021. `https://www.tagesschau.de/faktenfinder/covid-prognose-rki-101.html`. Accessed 2021-07-16. [see page 2]

[REL20] RELEX Solutions. Planning for Every Future in Grocery Retail. Technical report, RELEX Solutions & EnsembleIQ, 2020. `https://hub.relexsolutions.com/grocery-retail-report-2020`. Accessed: 2021-07-16. [see page 2]

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986. [see page 44]

[RM05] Lior Rokach and Oded Maimon. Clustering Methods. In *Data Mining and Knowledge Discovery Handbook*, pages 321–352. Springer, 2005. [see pages 17, 37, and 192]

[Ros96] Oliver Rose. Estimation of the Hurst Parameter of Long-range Dependent Time Series. Technical report, University of Würzburg, Institute of Computer Science, 1996. [see page 23]

[RPDJ12] Leszek Rutkowski, Lena Pietruczuk, Piotr Duda, and Maciej Jaworski. Decision Trees for Mining Data Streams based on the McDiarmid's Bound. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1272–1279, 2012. [see page 276]

[RYC+20] Wei Rang, Donglin Yang, Dazhao Cheng, Kun Suo, and Wei Chen. Data Life Aware Model Updating Strategy for Stream-based Online Deep Learning. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 392–398. IEEE, 2020. [see page 91]

[Sch98] Arthur Schuster. On the Investigation of Hidden Periodicities with Application to a Supposed 26 Day Period of Meteorological Phenomena. *Terrestrial Magnetism*, 3(1):13–41, 1898. [see page 26]

[SCH+19] Xiaoyi Sun, Krishnendu Chakrabarty, Ruirui Huang, Yiquan Chen, Bing Zhao, Hai Cao, Yinhe Han, Xiaoyao Liang, and Li Jiang. System-level Hardware Failure Prediction using Deep Learning. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019. [see page 81]

[SD10] Stephan Schlüter and Carola Deuschle. Using Wavelets for Time Series Forecasting: Does it Pay Off? Technical report, IWQW discussion paper series, 2010. [see pages 65 and 142]

[Sea99] Seagate Product Marketing. Get S.M.A.R.T. for Reliability. Technical report, Technical report, Seagate Technology Paper, 1999. [see pages 216 and 231]

[SFN18] Cameron Sobie, Carina Freitas, and Mike Nicolai. Simulation-driven Machine Learning: Bearing Fault Classification. *Mechanical Systems and Signal Processing*, 99:403–419, 2018. [see page 80]

[SG86] Jeffrey C Schlimmer and Richard H Granger. Incremental Learning from Noisy Data. *Machine Learning*, 1(3):317–354, 1986. [see pages 85 and 86]

[SHM11] Joanna Z Sikorska, Melinda Hodkiewicz, and Lin Ma. Prognostic Modelling Options for Remaining Useful Life Estimation by Industry. *Mechanical systems and signal processing*, 25(5):1803–1836, 2011. [see page 74]

[SJ21] Daoming She and Minping Jia. A BiGRU Method for Remaining Useful Life Prediction of Machinery. *Measurement*, 167:108277, 2021. [see page 84]

[SLM10] Felix Salfner, Maren Lenk, and Miroslaw Malek. A Survey of Online Failure Prediction Methods. *ACM Computing Surveys*, 42(3):1–42, 2010. [see page 67]

[SLS99] Nadeem Ahmed Syed, Huan Liu, and Kah Kay Sung. Handling Concept Drifts in Incremental Learning with Support Vector Machines. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 317–321. ACM, 1999. [see page 86]

[SM07] Lu Shuang and Li Meng. Bearing Fault Diagnosis based on PCA and SVM. In *International Conference on Mechatronics and Automation*, pages 3503–3507. IEEE, 2007. [see page 244]

[Små21] Johanna Småros. Winning the Food Fight: Best Practices for Managing Grocery Retail Supply Chains. `https://www.relexsolutions.com/resources/managing-`

grocery-retail-supply-chains/, 2021. Accessed: 2021-07-16. [see page 2]

[Sol89] Ray J Solomonoff. A System for Incremental Learning based on Algorithmic Probability. In *Proceedings of the Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, pages 515–527. Citeseer, 1989. [see page 86]

[SOR+03] Ramendra K Sahoo, Adam J Oliner, Irina Rish, Manish Gupta, José E Moreira, Sheng Ma, Ricardo Vilalta, and Anand Sivasubramaniam. Critical Event Prediction for Proactive Management in Large-scale Computer Clusters. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 426–435. ACM, 2003. [see page 68]

[SSH16] Matthias Sommer, Anthony Stein, and Jörg Hähner. Local Ensemble Weighting in the Context of Time Series Forecasting using XCSF. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016. [see pages 57 and 142]

[SSP+14] Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820, 2014. [see page 76]

[Sta18] Statista. Wie Bewerten Sie Die Relevanz Von Vorhersage Von Wartungsaufgaben (Predictive Maintenance) Für Ihr Unternehmen? https://de.statista.com/prognosen/943287/expertenbefragung-zu-predictive-maintenance-in-der-logistikbranche, 2018. Accessed: 2021-07-16. [see pages 1 and 244]

[SWH20] Devarshi Shah, Jin Wang, and Q Peter He. Feature Engineering in Big Data Analytics for IoT-Enabled Smart Manufacturing–Comparison between Deep Learning and Statistical Learning. *Computers & Chemical Engineering*, 141:106970, 2020. [see page 84]

[SWLY18] Jing Shen, Jian Wan, Se-Jung Lim, and Lifeng Yu. Random-forest-based Failure Prediction for Hard Disk Drives. *International Journal of Distributed Sensor Networks*, 14(11):1550147718806480, 2018. [see page 80]

[Tay03] James W Taylor. Short-term Electricity Demand Forecasting using Double Seasonal Exponential Smoothing. *Journal of the Operational Research Society*, 54(8):799–805, 2003. [see page 1]

[THA18] Thiyanga S Talagala, Rob J Hyndman, and George Athanasopoulos. Meta-Learning How to Forecast Time Series. Technical report, Monash University, Department of Econometrics and Business Statistics, 2018. [see page 62]

[Tia12] Zhigang Tian. An Artificial Neural Network Method for Remaining Useful Life Prediction of Equipment Subject to Condition Monitoring. *Journal of Intelligent Manufacturing*, 23(2):227–237, 2012. [see page 74]

[TK92] Kar Yan Tam and Melody Y Kiang. Managerial Applications of Neural Networks: The Case of Bank Failure Predictions. *Management Science*, 38(7):926–947, 1992. [see page 4]

[TLG93] Timo Teräsvirta, Chien-Fu Lin, and Clive WJ Granger. Power of the Neural Network Linearity Test. *Journal of Time Series Analysis*, 14(2):209–220, 1993. [see page 23]

[TMMZT12] Diego Alejandro Tobon-Mejia, Kamal Medjaher, Noureddine Zerhouni, and Gerard Tripot. A data-driven failure prognostics method based on mixture of gaussians hidden markov models. *IEEE Transactions on reliability*, 61(2):491–503, 2012. [see page 244]

[UW99] Kai Uchtmann and Rainer Wirth. Maschinendiagnose an drehzahlveränderlichen Antrieben mittels Ordnungsanalyse. *Antriebstechnik*, 38(5):44–49, 1999. [Available in German only]. [see pages 194 and 202]

[VAH+02] Ricardo Vilalta, Chidanand V Apte, Joseph L Hellerstein, Sheng Ma, and Sholom M Weiss. Predictive Algorithms in the Management of Computer Systems. *IBM Systems Journal*, 41(3):461–474, 2002. [see page 68]

[VGCBS04] Ricardo Vilalta, Christophe G Giraud-Carrier, Pavel Brazdil, and Carlos Soares. Using Meta-learning to Support Data Mining. *International Journal of Computer Science & Applications*, 1(1):31–45, 2004. [see pages 144, 145, and 299]

[VHH16] Birgit Vogel-Heuser and Dieter Hess. Guest editorial industry 4.0–prerequisites and visions. *IEEE Transactions on Automation Science and Engineering*, 13(2):411–413, 2016. [see page 185]

[VSI21] Kevin Villalobos, Johan Suykens, and Arantza Illarramendi. A Flexible Alarm Prediction System for Smart Manufacturing Scenarios Following a Forecaster–Analyzer Approach. *Journal of Intelligent Manufacturing*, 32(5):1323–1344, 2021. [see page 72]

[VSSB13] Sebastian Vansyckel, Dominik Schäfer, Gregor Schiele, and Christian Becker. Configuration Management for Proactive Adaptation in Pervasive Environments. In *Proceedings of the IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*, pages 131–140. IEEE, 2013. [see pages 8, 20, 244, and 300]

[Wal] Wall Street Journal Custom Studios. How Manufacturers can Achieve Top Quartile Performance. `https://partners.wsj.com/emerson/unlocking-performance/how-manufacturers-can-achieve-top-quartile-performance/`. Accessed: 2021-07-16. [see page 243]

[Wel67] Peter Welch. The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, 1967. [see page 203]

[Wer88] Paul J Werbos. Generalization of Backpropagation with Application to a Recurrent Gas Market Model. *Neural Networks*, 1(4):339–356, 1988. [see page 46]

[Wey17] Danny Weyns. Software Engineering of Self-adaptive Systems: An Organised Tour and Future Challenges. *Chapter in Handbook of Software Engineering*, 2017. [see pages 2 and 141]

[WHZ07] Wei Wu, Jingtao Hu, and Jilong Zhang. Prognostics of Machine Health Condition using an Improved ARIMA-based Prediction Method. In *2nd IEEE Conference on Industrial Electronics and Applications*, pages 1062–1067. IEEE, 2007. [see page 69]

[Win60] Peter R Winters. Forecasting Sales by Exponentially Weighted Moving Averages. *Management science*, 6(3):324–342, 1960. [see page 33]

[Wir98] Rainer Wirth. Maschinendiagnose an Industriegetrieben Teil II: Signalidentifikation in der Praxis. *Antriebstechnik*, 37(11):77–81, 1998. [Available in German only]. [see page 203]

[WK96] Gerhard Widmer and Miroslav Kubat. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning*, 23(1):69–101, 1996. [see page 269]

[WM95] David H Wolpert and William G Macready. No Free Lunch Theorems for Search. Technical report, SFI-TR-95-02-010, The Santa Fe Institute, 1995. [see page 55]

[WM97] David H Wolpert and William G Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. [see pages 4, 32, 55, 95, 141, and 157]

[WM18] Cheng-Lung Wu and Stephen J Maher. Airline Capacity Planning and Management. In *The Routledge Companion to Air Transport Management*, pages 238–258. Routledge, 2018. [see page 2]

[Wor15] World Economic Forum. Industrial Internet of Things: Unleashing the Potential of Connected Products and Services. Technical report, World Economic Forum, 2015. [see page 244]

[WPT98] Walter Willinger, Vern Paxson, and Murad S Taqqu. Self-similarity and Heavy Tails: Structural Modeling of Network Traffic. *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, 23:27–53, 1998. [see page 23]

[WSMH09] Xiaozhe Wang, Kate Smith-Miles, and Rob J Hyndman. Rule Induction for Forecasting Method Selection: Meta-learning the Characteristics of Univariate Time Series. *Neurocomputing*, 72(10-12):2581–2594, 2009. [see pages 21, 60, 142, 143, 144, 145, 152, and 163]

[WWCL09] Aiping Wang, Guowei Wan, Zhiquan Cheng, and Sikun Li. An Incremental Extremely Random Forest Classifier for Online Learning and Tracking. In *16th IEEE International Conference on Image Processing (ICIP)*, pages 1449–1452. IEEE, 2009. [see page 87]

[WZT+18] Yi Wang, Ning Zhang, Yushi Tan, Tao Hong, Daniel S Kirschen, and Chongqing Kang. Combining Probabilistic Load Forecasts. *IEEE Transactions on Smart Grid*, 10(4):3664–3674, 2018. [see page 57]

[XCCP06] Jing-Xin Xie, Chun-Tian Cheng, Kwok-Wing Chau, and Yong-Zhen Pei. A Hybrid Adaptive Time-delay Neural Network Model for Multi-step-ahead Prediction of Sunspot Activity. *International Journal of Environment and Pollution*, 28(3-4):364–381, 2006. [see pages 65 and 142]

[XPL21] Yongyong Xiang, Baisong Pan, and Luping Luo. A New Model Updating Strategy with Physics-based and Data-driven Models. *Structural and Multidisciplinary Optimization*, 64:163–176, 2021. [see page 91]

[XW09] Xiaoyan Xu and Yu Wang. Financial Failure Prediction using Efficiency as a Predictor. *Expert Systems with Applications*, 36(1):366–373, 2009. [see page 4]

[XW16] Shuliang Xu and Junhong Wang. A Fast Incremental Extreme Learning Machine Algorithm for Data Streams Classification. *Expert Systems with Applications*, 65:332–344, 2016. [see pages 88 and 276]

[XWL+16] Chang Xu, Gang Wang, Xiaoguang Liu, Dongdong Guo, and Tie-Yan Liu. Health Status Assessment and Failure Prediction for Hard Drives with Recurrent Neural Networks. *IEEE Transactions on Computers*, 65(11):3502–3508, 2016. [see pages 76 and 269]

[XXW+18] Jiang Xiao, Zhuang Xiong, Song Wu, Yusheng Yi, Hai Jin, and Kan Hu. Disk Failure Prediction in Data Centers via Online Learning. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10. ACM, 2018. [see pages 90, 271, and 288]

[XZCM20] Zhiwei Xue, Yong Zhang, Cheng Cheng, and Guijun Ma. Remaining Useful Life Prediction of Lithium-ion Batteries with Adaptive Unscented Kalman Filter and Optimized Support Vector Regression. *Neurocomputing*, 376:95–102, 2020. [see page 244]

[YDN08] Zimin Max Yang, Dragan Djurdjanovic, and Jun Ni. Maintenance Scheduling in Manufacturing Systems based on Predicted Machine Degradation. *Journal of Intelligent Manufacturing*, 19(1):87–98, 2008. [see page 244]

[YHL+15] Wenjun Yang, Dianming Hu, Yuliang Liu, Shuhao Wang, and Tianming Jiang. Hard Drive Failure Prediction using Big Data. In *IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*, pages 13–18. IEEE, 2015. [see page 269]

[YM12] Ming-Yi You and Guang Meng. A Modularized Framework for Predictive Maintenance Scheduling. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 226(4):380–391, 2012. [see page 244]

[ZAG+21] Marwin Züfle, Joachim Agne, Johannes Grohmann, Ibrahim Dörtoluk, and Samuel Kounev. A Predictive Maintenance Methodology: Predicting the Time-to-Failure of Machines in Industry 4.0. In *Proceedings of the 21st IEEE IES International Conference on Industrial Informatics (INDIN)*. IEEE, 2021. (to appear). [see pages xxv, 245, and 255]

[ZBH+17] Marwin Züfle, André Bauer, Nikolas Herbst, Valentin Curtef, and Samuel Kounev. Telescope: A Hybrid Forecast Method for Univariate Time Series. In *International Work-Conference on Time Series Analysis (ITISE)*, 2017. [see pages xxvi and 96]

[ZBL+19] Marwin Züfle, André Bauer, Veronika Lesch, Christian Krupitzer, Nikolas Herbst, Samuel Kounev, and Valentin Curtef. Autonomic Forecasting Method Selection: Examination and Ways Ahead. In *Proceedings of the 16th IEEE International Conference on Autonomic Computing (ICAC)*, pages 167–176. IEEE, 2019. [see pages xxvi, 142, and 161]

[Zei16] Zeit Online. Wolfsburger Golf-Produktion steht für eine Woche still. *Zeit Online*, 2016. `https://www.zeit.de/mobilitaet/2016-08/volkswagen-vw-golf-wolfsburg-produktion-stillstand`. Accessed: 2021-07-16. [see page 185]

[ZEK21] Marwin Züfle, Florian Erhard, and Samuel Kounev. Machine Learning Model Update Strategies for Hard Disk Drive Failure Prediction. In *Proceedings of the 20th International Conference on*

Machine Learning and Applications (*ICMLA*). IEEE, 2021. (under review). [see pages xxv, 270, and 279]

[ZG15] Chunhui Zhao and Furong Gao. Online Fault Prognosis with Relative Deviation Analysis and Vector Autoregressive Modeling. *Chemical Engineering Science*, 138:531–543, 2015. [see pages 3 and 71]

[Zha03] Peter G Zhang. Time Series Forecasting using a Hybrid ARIMA and Neural Network Model. *Neurocomputing*, 50:159–175, 2003. [see pages 35 and 63]

[ZK20] Marwin Züfle and Samuel Kounev. A Framework for Time Series Preprocessing and History-based Forecasting Method Recommendation. In *Proceedings of the 2020 Federated Conference on Computer Science and Information Systems*, pages 141–144. IEEE, 2020. [see pages xxvi, 143, and 161]

[ZKE+20] Marwin Züfle, Christian Krupitzer, Florian Erhard, Johannes Grohmann, and Samuel Kounev. To Fail or Not to Fail: Predicting Hard Disk Drive Failure Time Windows. In *Proceeding of the International Conference on Measurement, Modelling and Evaluation of Computing Systems*, pages 19–36. Springer, 2020. [see pages xxvi, 215, and 225]

[ZLJ19] Rongtao Zhang, Binbin Li, and Bin Jiao. Application of XGBoost Algorithm in Bearing Fault Diagnosis. In *IOP Conference Series: Materials Science and Engineering*, page 072062. IOP Publishing, 2019. [see page 244]

[ZLZZ15] Xiaoyuan Zhang, Yitao Liang, Jianzhong Zhou, and Yi Zang. A Novel Bearing Fault Diagnosis Model Integrated Permutation Entropy, Ensemble Empirical Mode Decomposition and Optimized SVM. *Measurement*, 69:164–179, 2015. [see page 244]

[ZML+21] Marwin Züfle, Felix Moog, Veronika Lesch, Christian Krupitzer, and Samuel Kounev. A Machine Learning-based Workflow for Automatic Detection of Anomalies in Machine Tools. *ISA Transactions*, 2021. (in press). [see pages xxv, 186, and 199]

[ZQZF18] Qi Zhao, Xiaoli Qin, Hongbo Zhao, and Wenquan Feng. A Novel Prediction Method based on the Support Vector Regres-

sion for the Remaining Useful Life of Lithium-ion Batteries. *Microelectronics Reliability*, 85:99–108, 2018. [see page 244]

[Züf17] Marwin Züfle. Dynamic Hybrid Forecasting for Self-Aware Systems. Master Thesis, Julius-Maximilians-Universität Würzburg, 2017. [see page 96]

[Züf20] Marwin Züfle. Towards a Self-Aware Prediction of Critical States. In Sven Tomforde and Christian Krupitzer, editors, *Organic Computing: Doctoral Dissertation Colloquium 2020*, pages 139–155. Kassel University Press GmbH, 2020. [see page xxvii]

[ZWL+13] Bingpeng Zhu, Gang Wang, Xiaoguang Liu, Dianming Hu, Sheng Lin, and Jingwei Ma. Proactive Drive Failure Prediction for Large Scale Storage Systems. In *IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–5. IEEE, 2013. [see pages 75 and 79]

[ZXHP18] Yongzhi Zhang, Rui Xiong, Hongwen He, and Michael G Pecht. Long Short-term Memory Recurrent Neural Network for Remaining Useful Life Prediction of Lithium-ion Batteries. *IEEE Transactions on Vehicular Technology*, 67(7):5695–5705, 2018. [see pages 9, 80, and 244]

[ZXK+05] Xiaodong Zhang, Roger Xu, Chiman Kwan, Steven Y Liang, Qiulin Xie, and Leonard Haynes. An Integrated Approach to Bearing Fault Diagnostics and Prognostics. In *Proceedings of the 2005 American Control Conference*, pages 2750–2755. IEEE, 2005. [see page 244]

[ZXL07] Jie Zhao, Limei Xu, and Lin Liu. Equipment Fault Forecasting based on ARMA Model. In *International Conference on Mechatronics and Automation*, pages 3514–3518. IEEE, 2007. [see page 69]

[ZZC+21] Ying Zhang, Baohang Zhou, Xiangrui Cai, Wenya Guo, Xiaoke Ding, and Xiaojie Yuan. Missing Value Imputation in Multivariate Time Series with End-to-end Generative Adversarial Networks. *Information Sciences*, 551:67–82, 2021. [see page 298]

[ZZH+20] Ji Zhang, Ke Zhou, Ping Huang, Xubin He, Ming Xie, Bin Cheng, Yongguang Ji, and Yinhu Wang. Minority Disk Failure

Prediction based on Transfer Learning in Large Data Centers of Heterogeneous Disk Systems. *IEEE Transactions on Parallel and Distributed Systems*, 31(9):2155–2169, 2020. [see page 82]