Explainable and Interpretable Time Series Forecasting and Predictive Maintenance

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Matthias Jakobs

Dortmund

2025

Tag der mündlichen Prüfung:	09. 10. 2025
Dekan:	Prof. Dr. Jens Teubner
Gutachter:	Prof. Dr. Thomas Liebig
	Prof. Dr. João Gama

Abstract

Time-series data is ubiquitous in numerous application domains today, including safety-critical settings such as medical and industrial scenarios. However, real-world data can change its characteristics with time, which is referred to as drifts in concept. The phenomenon of concept drifts is particularly troubling for time-series forecasting tasks, where the goal is to predict future time-series values given known, past values. Traditional forecasting methods assume that these characteristics are fixed to achieve a high predicting performance. In this thesis, we will inspect methods how to adapt to changing concepts for time-series forecasting using both complex, deep learning methods and simple, interpretable models. To do that, we utilize the online model selection and online ensemble pruning frameworks for selecting between pools of different models in an online manner. Specifically, we will focus on aspects of explainability and interpretability throughout as these aspects are crucial for applying forecasting methods in practice. Additionally, we investigate a real-world scenario for predictive maintenance, one area of application where explainable and interpretable time-series analysis methods are crucial to gain insights for practitioners and technicians alike.

Acknowledgments

First and foremost I would like to thank my supervisor Prof. Dr. Thomas Liebig for his guidance, insights and suggestions throughout the process of writing this thesis. Additionally, I would like to thank Prof. Dr. Katharina Morik who, until her retirement, supervised me and allowed me to work at TU Dortmund University as part of first the ML2R project and later as part of the Lamarr Institute.

During my PhD I was fortunate to have done a research stay at *Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência* (INESC-TEC) in Porto, Portugal. Specifically, I would like to thank Prof. Dr. João Gama and Dr. Bruno Veloso for supervising and hosting me throughout my stay and additionally Prof. Gama for co-supervising this thesis! Moreover, I would like to thank all the colleagues of LIAAD / INESC-TEC I got to meet for making the time in Porto unforgettable.

I also was lucky enough to meet and collaborate with multiple colleagues throughout this year to whom I would like to extend my gratitude as well. In particular, I want to thank my dear colleagues Dr. Sascha Mücke, Dr. Raphael Fischer, Dr. Sebastian Buschjäger, Ann-Kathrin Oster and Dr. Amal Saadallah for fruitful discussions and an overall fun time!

Last but not least I want to thank my friends and family for their continued support and advice throughout the entire process of my PhD. In particular, I want to thank my wonderful fiancé Judith for always supporting me throughout the writing of this thesis. I could not have done this alone, thank you!

Contents

Part I.

Fundamentals

1. Introduction

Time-series data are ubiquitous throughout industry, academia and healthcare applications [MR22; Ull+20; Rai+21; Per+21]. This is due to the fact that almost all aspects of these domains are increasingly digitized. In astroparticle physics, for example, time-series are omnipresent since many observations of distant stars and galaxies can be seen as time-varying data [MR22]. In industrial applications, time-series occur prominently in the form of sensor readings. As an example, consider the case of monitoring different parts of a machine using analogue and digital sensors. With enough data, it is feasible to predict when these characteristics start to drift, indicating that there is (or soon will be) something wrong with the machine [SVG23; JVG25]. Moreover, patients are more and more diagnosed and treated based on data gathered over time, such as Electroencephalography (EEG), Electrocardiography (ECG) and others [Per+21]. Thus, devising methods to generate knowledge from time-series data is vital for many relevant real-world applications.

One way to utilize these large amounts of data is to try and *learn* from them by *training* parameterized functions, or *models*. The research field of Machine Learning (ML) has been tackling the tasks of modeling and predicting from data patterns for decades, with time-series analysis being a highly relevant subfield due the aforementioned ubiquity of time-series data applications. Of particular interest when dealing with time-series data is often the desire to predict, given some amount of historical data, how the time-series will likely behave in the future. This field within time-series analysis is called time-series *forecasting*. It is studied for traffic prediction, weather forecasts, stock market predictions and predictive maintenance, to name just a few [Kra+17; Ser+22]. By analyzing historic data the goal is to model the distribution of those time-series using ML approaches in order to confidently predict future values in advance. However, the characteristics of time-series data are complex, and often evolve over time, making time-series forecasting a particularly tough problem [SPM19]. The fact that properties of the data distribution, such as mean and variance, will change over time is often discussed under the name of *concept drift* in the literature [Gam+14] and requires the used ML methods to i) observe the data and detect possible changes and ii) adapt to those changes in a timely fashion.

Machine Learning research in general (and time-series analysis in particular) has experienced a sharp increase in interest in recent years due to two factors. First, the number of (very large) available datasets from all kinds of domains has increased due to cheap sensors, ubiquitous data sources and an increasing digitization. Second, the improvements in the field of Deep Learning (DL), a method of learning very large Neural Network (NN) models that scale well with larger and larger amounts of data. Together, these two factors

lead to ML models being adopted widely, for example in the form of Large Language Models (LLMs) such as ChatGPT[1] and Google Gemini[2]. Very large models are also increasingly adopted in time-series forecasting [LZ21; Zho+21; Wu+21; HH19; KC19; Shi+15]. Among the most widely adopted DL models are Convolutional Neural Networks (CNNs) and Long Short-Term Memorys (LSTMs), which are specific NN model architectures designed for structured, sequential data. While they are often highly accurate models they have one practical downside hindering their adoption in practice. Since they are able to learn highly non-linear feature representations using millions (and, recently, billions) of parameters there is no way to inspect or verify, let alone certify, how they come to their predictions. This makes their adoption to safety-critical fields such as financial and medical domains very hard. Questions such as "Why was the patient diagnosed with the particular disease?" or "Why did a part of the machine break?" cannot be answered directly, making DL models unsuitable for high-risk decision-making, even if their predictions are highly accurate. If it cannot be ruled out that decisions, even correct ones, are made due to noise in the features, spurious correlations or other erroneous factors, the trust in those models and their viability to be applied in practice diminishes. Society will not adopt this high-impact technology in critical sectors if they cannot trust in the predictions to be *right for the right reason*[RHD17; Sch+20].

To counter and address these problems researchers developed new methodology in parallel to DL advances. This research is often conducted under the names *Explainable AI*, *Trustworthy AI*, *Transparency* or *Interpretability*. Especially in the early stages of this novel research field, and still during the writing of this thesis, the nomenclature is not sharply defined by the community and often used interchangeably [Mol20; Rud19; Bec+23]. Thus, to avoid confusion from this point on, we will use the definition of Cynthia Rudin [Rud19], especially the distinction between *Explainability* (or Explainable Machine Learning (XML)) and *Interpretability* (or Interpretable Machine Learning (IML)). Explainability is concerned with taking models which have already been trained and evaluated and uncovering their inner workings through abstractions. This is mostly achieved through investigating the importance of certain features (and their explicit feature values for a particular decision) as well as finding subsets of data the model is very good at. Both allow researchers to come up with hypotheses about how the model works internally, which can subsequently be studied and verified. The other school of thought is what we will refer to as *Interpretability*, which is concerned with using inherently transparent and easy to understand models. This ranges from focusing on more classical, often small models, to deriving novel DL architectures that are more interpretable [Aga+21; BB19]. As postulated in [Rud19], it is often unnecessary to use complex DL models in the first place, since comparable performance on specific, real-world tasks can often be achieved with traditional, simpler methods [Zen+23], especially for safety-critical application settings with highly structured data and domain knowledge. Interpretability research can also utilize DL methods to come up with these small and interpretable models, making it possible to use the progress made in DL while still being able to deploy simple models

[1] chat.openai.com
[2] gemini.google.com

in practice [FH17]. In general, the advantages of combining high performing DL models and interpretable methods will be studied in later chapters of this thesis in depth.

As an example of the intersection of time-series analysis and the necessity for XML and IML methods consider the setting of a hospital sleep laboratory. In a sleep laboratory, patients are committed to sleep a full night while being under the full supervision of doctors, as well as being monitored using various sensors. For example, these measurements include an Electroencephalography (EEG) where brain functions are measured using multiple sensors placed over the patient's head. The doctors will manually annotate this data afterwards, taking note of whether the patient is awake or asleep, whether snoring occurred or whether the patient experienced obstructive apneas. Based on this data the doctors can form diagnoses and prescribe medications. It has been shown that ML approaches are (to a certain extent) able to take over this labor and time intensive annotation process [Per+21] as well as to identify anomalous slices of data [Xu+20; Che+20] for the doctors to inspect. However, these very successful methods are based on deep neural networks which, as discussed before, are opaque and uninterpretable. For the trust of the doctor in using ML as a tool as well as of the patient in the procedure itself it is important to understand model predictions on a fundamental level. Both XML and IML approaches offer a way to shed light into these methods.

Thus, in order to utilize the high predictive performance of modern DL methods for medical, scientific and industrial applications (where time-series data are prevalent) we need to study how we can make these models more trustworthy and safe. In this thesis we contribute towards this goal by making the following contributions:

- We adapt online model selection and ensembling for time-series forecasting, essential to tackling concept drifts, to be more transparent and more accurate by incorporating Grad-CAM [Sel+19] and Shapley values [LL17; Lun+20], two prominent XML methods used for computer vision and tabular data. We show that we can answer why a specific model is chosen to predict at each time instance and what the data looks like that each model is confident in predicting. That way we can build an understanding of the expertise that each model has, leading to better insights into the decision-making process.

- We show for the case of time-series forecasting that simple, interpretable, linear models can be successfully used for online time-series forecasting in combination with complex, opaque DL methods. Specifically, we can predict when the high performance DL methods are really necessary to achieve a low prediction error and find that this is rarely the case in practice. By omitting opaque models for the actual prediction we can reach maximum interpretability and trust as the models are simple enough that practitioners can understand them.

- We present a novel, straight-forward online rule-learning approach. We evaluate its effectiveness for the predictive maintenance use case where we are able to extract simple rules which explain the failure of real-world Metro trains from the

city of Porto in Portugal. Again, we achieve this by combining DL methods and interpretable models in the form of Decision Trees.

1.1. Outline

This thesis is separated into three parts.

In **Part I** we begin with the introduction to the thesis. Afterwards, Chapter 2 will introduce fundamental concepts which will be referenced throughout the thesis. Namely, we introduce the topic of ML formally, together with the ML approaches fundamental to this work, which are NNs and tree-based models. We will discuss the basics of time-series data, including time-series forecasting under concept drift and how one can effectively adapt to drifts using model selection and ensembling. Lastly, we will discuss the foundations of XML and IML and how they will be used for time-series analysis throughout this thesis.

In **Part II** we cover the proposed methods of using explainability methods to both improve our understanding and the prediction performance of CNNs and tree-based models for time-series forecasting. In Chapter 3, we present an online model selection approach which can adapt to concept drifts by adjusting each model's Region of Competence (RoC) [Pri19]. RoCs are constructed using explainability methods for DL methods to identify subsets of data that a model performs very well on. This allows the user to both inspect what these regions look like and allows the model selection algorithm to make use of them to calculate which model should be select for forecasting new data. Afterwards, we will extend this approach to online ensembling in Chapter 4. Our ensemble approach is able to adapt dynamically in terms of ensemble size to detected drifts in both the data and prediction performance of its members. Finally, we present another extension in Chapter 5 where the models available for selection are chosen to be tree-based models instead of DL models. Tree-based models can be inspected efficiently with Shapley values, a theoretically principled and popular XML method. Additionally, the characteristics of tree-based models can be used to further inspect the inner workings of our models, for example by generating rules w.r.t. individual predictions. Each of these three chapters contains an extensive experiment section where we compare the predictive performance to state-of-the-art methods for model selection and ensembling for time-series forecasting. Moreover, we provide extensive sections discussing the Explainability aspects of our proposed methods.

Part III focuses on aspects of Interpretability for time-series analysis. Specifically, we present a model selection approach in Chapter 6 which learns to choose between a simple, interpretable forecaster, and uninterpretable DL models, in an online manner. We are able to show that we only need to choose the large models for very few predictions to maintain the overall predictive performance of always choosing the large model. Moreover, we show that we are able to learn when to use which model from data, resulting in comparable predictive performance to state-of-the-art model selection approaches with

a high increase in interpretability. To further illustrate that interpretable models can be used to gain insight in real-world use cases, we present our approach of predicting failures of trains in the *Metro do Porto* in Portugal in Chapter 7. By using a hybrid approach of combining DL models with simple, interpretable decision trees we are able to gain insights into the failure characteristics of the train, alarming the train conductors of failures in time while also generating understandable rules on why the failure happened. Lastly, we will conclude the thesis in Chapter 8 by summarizing our findings, as well as discussing limitations and providing suggestions for future work.

1.1.1. Publications covered by this Thesis

The works used in this thesis were supported between 2020 and 2022 by the Federal Ministry of Education and Research of Germany as part of the Competence Center for Machine Learning ML2R and from 2022 to 2025 by the Federal Ministry of Education and Research of Germany and the state of North Rhine-Westphalia as part of the Lamarr Institute for Machine Learning and Artificial Intelligence.

- Matthias Jakobs, Bruno Veloso, and João Gama. "Interpretable Rules for Online Failure Prediction: A Case Study on the Metro Do Porto Dataset". In: *arXiv:2502.07394 [cs]* (2025). DOI: 10.48550/arXiv.2502.07394, (preprint)
- Matthias Jakobs and Thomas Liebig. "AALF: Almost Always Linear Forecasting". In: *arXiv:2409.10142 [cs]* (2025). DOI: 10.48550/arXiv.2409.10142, (preprint)
- Matthias Jakobs and Amal Saadallah. "Explainable Adaptive Tree-based Model Selection for Time-Series Forecasting". In: *Proceedings of the International Conference on Data Mining*. 2023. DOI: 10.1109/ICDM58522.2023.00027
- Amal Saadallah, Matthias Jakobs, and Katharina Morik. "Explainable Online Ensemble of Deep Neural Network Pruning for Time Series Forecasting". In: *Machine Learning* (2022). DOI: 10.1007/s10994-022-06218-4
- Amal Saadallah, Matthias Jakobs, and Katharina Morik. "Explainable Online Deep Neural Network Selection Using Adaptive Saliency Maps for Time Series Forecasting". In: *Machine Learning and Knowledge Discovery in Databases: Research Track*. 2021. DOI: 10.1007/978-3-030-86486-6

Note that the first two publications are available as preprints as they were under review during the writing of this thesis. Specifically, [JL25] was under review at the *Machine Learning* journal whereas [JL25] was under review at the *IEEE Access* journal.

1.1.2. Publications not covered by this Thesis

The following publications were part of collaborations with numerous colleagues involving several research fields and application settings. Their content will not be featured prominently in this thesis.

- Raoul Heese et al. “Explaining Quantum Circuits with Shapley Values: Towards Explainable Quantum Machine Learning”. In: *Quantum Machine Intelligence* 7 (2025). DOI: 10.1007/s42484-025-00254-8
- Bruno Casella et al. “Federated Time Series Classification with ROCKET Features”. In: *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2024. DOI: 10.14428/esann/2024.es2024-61
- Katharina Beckh et al. “Harnessing Prior Knowledge for Explainable Machine Learning: An Overview”. In: *Proceedings of the IEEE Conference on Secure and Trustworthy Machine Learning*. 2023. DOI: 10.1109/SaTML54575.2023.00038
- Amal Saadallah and Matthias Jakobs. “Online Deep Hybrid Ensemble Learning for Time Series Forecasting”. In: *Machine Learning and Knowledge Discovery in Databases: Research Track*. 2023, pp. 156–171. DOI: 10.1007/978-3-031-43424-2_10
- Raoul Heese et al. “Shapley Values with Uncertain Value Functions”. In: *Advances in Intelligent Data Analysis*. 2023. DOI: 10.1007/978-3-031-30047-9_13
- Raphael Fischer et al. “A Unified Framework for Assessing Energy Efficiency of Machine Learning”. In: *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. 2023, pp. 39–54. DOI: 10.1007/978-3-031-23618-1_3
- Sebastian Müller et al. “An Empirical Evaluation of the Rashomon Effect in Explainable Machine Learning”. In: *Machine Learning and Knowledge Discovery in Databases: Research Track*. 2023. DOI: 10.1007/978-3-031-43418-1_28
- Raphael Fischer, Matthias Jakobs, and Katharina Morik. “Energy Efficiency Considerations for Popular AI Benchmarks”. In: *arXiv:2304.08359 [cs]* (2023). DOI: 10.48550/arXiv.2304.08359, (preprint)
- Matthias Jakobs et al. “SancScreen: Towards a Real-world Dataset for Evaluating Explainability Methods”. In: *Proceedings of the Conference ”Lernen, Wissen, Daten, Analysen”*. 2022
- Rahel Wilking, Matthias Jakobs, and Katharina Morik. “Fooling Perturbation-Based Explainability Methods”. In: *Workshop on Trustworthy Artificial Intelligence as Part of the ECML-PKDD 22*. 2022
- Katharina Morik et al. “Yes We Care!-Certification for Machine Learning Methods through the Care Label Framework”. In: *Frontiers in Artificial Intelligence* 5 (2022). DOI: 10.3389/frai.2022.975029
- Raphael Fischer et al. “Solving Abstract Reasoning Tasks with Grammatical Evolution”. In: *Proceedings of the Conference ”Lernen, Wissen, Daten, Analysen”*. 2020

2. Background

This chapter will start by introducing some general notation which will be used throughout this thesis in Section 2.1. The fundamentals of ML, including supervised, unsupervised and self-supervised learning, will be discussed in Section 2.2. Afterwards, we will introduce time-series data formally in Section 2.3, with a focus on time-series forecasting in Section 2.3.2. We will also discuss different types of *stationarity* and *concept drifts* in Section 2.3.1. Since most of the thesis will contain discussions of DL and tree-based models in the context of time-series forecasting we will introduce these formally in Section 2.4. The chapter will conclude with an introduction to XML and IML in Section 2.5.

2.1. General Notation

First, let us introduce some basic notation that (if not otherwise mentioned) will be consistent throughout this thesis. We will denote with $\mathbb{B}, \mathbb{N}$ and $\mathbb{R}$ the set of binary, natural and real numbers, respectively. We will assume that the natural numbers start at one and will denote with $\mathbb{R}_0$ the natural numbers including zero. Scalar values are defined with non-bold letters, like $a \in \mathbb{R}$ or $B \in \mathbb{N}_0$. Vectors and matrices will be denoted with bold letters, such as $\boldsymbol{x} \in \mathbb{R}^n$ or $\boldsymbol{Z} \in \mathbb{R}^{n \times n}$, while sets will be denoted using upper case calligraphic letters such as $\mathcal{C} = \{C_1, \ldots, C_n\}$. When referring to an element of a vector or matrix we will use non-bold letters (when indexing a vector) or bold letters (when indexing a matrix). Additionally, we define $[n] := \{1, \ldots, n\}$ as well as the indicator function

$$\mathbb{I}[p] := \begin{cases} 1 & \text{if } p \\ 0 & \text{otherwise.} \end{cases}$$

2.2. Machine Learning

Traditionally, solving problems with computers has been done using algorithms: a fixed, ordered list of instructions, written by humans. While it is easy to reason over algorithms, understand each individual instruction and (maybe even) proof bounds on correctness and runtime, they are hard to apply to problems which are difficult to formalize. For example, consider the shortest path problem in graph theory, where the goal is to find the shortest path (in terms of number of edges or sum of edge weights) between two fixed nodes. Then, there are algorithms available that solve this problem (sometimes under certain graph constraints) for any graph with a known worst-case runtime [Dij59; Bel58; For56].

However, if we need to tackle a less formalized problem, for example to classify images based on their content or translate sentences between languages, we run into the problem that it is very difficult to find a list of ordered instructions which can solve these tasks perfectly.

Supervised Learning

Machine Learning tries to *learn* a connection between data and the desired output from *observations*, instead of following strict, predefined analysis steps. Let $p(x, y)$ be the joint probability over two spaces $\mathcal{X}$ and $\mathcal{Y}$ and let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be a sample (or observation) of *training examples* from $p(x, y)$. Specifically let $\boldsymbol{x}_i \in \mathcal{X}$ be the so-called *input* and $y_i \in \mathcal{Y}$ be the corresponding *label*. The label is an annotation that encodes the correct answer we want to associate with $\boldsymbol{x}_i$. For example, if $\boldsymbol{x}_i$ is an image then y_i could be the label encoding what is shown in it. As another example, $\boldsymbol{x}_i$ could be some entries from a table and y_i could be another column which we want to predict. Thus, ML aims to learn from examples how to predict y_i given $\boldsymbol{x}_i$.

We assume a given parameterized function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ which, given an input in $\mathcal{X}$ and its parameters $\boldsymbol{\theta}$, will output some value in $\mathcal{Y}$. By optimizing the parameters $\boldsymbol{\theta}$ we can change the mapping between the two spaces, making the model predictions more accurate. To quantify how accurate $f_{\boldsymbol{\theta}}$ is for a particular set of parameters we define a *loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ which calculates the error produced between model predictions $\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ and the true labels y. The task of *learning* or *training* now entails to adapt $\boldsymbol{\theta}$, based on the available training examples, to minimize this loss. More precisely, given the *risk* $R(\boldsymbol{\theta}) = \mathbb{E}[\ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y)]$ we want to find a set of parameters $\boldsymbol{\theta}^*$ which minimizes the risk. Since the expectation cannot be computed unless we have access to the full joint probability distribution, we aim to minimize the *empirical risk* (ER) in practice [Vap91], which gives rise to the following definition:

Definition 2.1 (Supervised Learning)**.** *Given a set of training examples* $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$*, a model* $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ *and a loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ *we aim to solve*

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathrm{ER}(\boldsymbol{\theta}, \mathcal{D}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i).$$

How this parameter set is optimized is highly dependent on the specific model that is being used. We will present specific learning methods for two model families, namely (Deep) Neural Networks and Tree-based models, in the next section.

The two most common application settings for supervised learning are *classification* and *regression* problems. Regression is generally concerned with predicting a real-valued target, i.e., $\mathcal{Y} = \mathbb{R}$. Since the prediction $\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ and the target y are both real-valued, a reasonable loss function is the *squared error* and, for multiple data points, the *mean squared error*, given by

$$\ell_{\text{MSE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N}\sum_{i=1}^{N} \ell_{\text{SE}}(\hat{y}_i, y_i) = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2.$$

One of the simplest, yet often used models for regression is *Linear Regression*, which models the relationship between input and output as a linear combination. Let $\boldsymbol{x}$ be a vector of features and y the corresponding regression target. Then, a Linear Regression model is given by

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w} + b, \quad \boldsymbol{\theta} = \{\boldsymbol{w}, b\} \tag{2.1}$$

where $\boldsymbol{w} = (w_1, w_2, \ldots, w_k)$ is the weight vector and w_i is associated with feature x_i. In addition, a constant offset (or *bias*) b is also part of the learnable parameters. Often the bias is included in the weight matrix by setting $\boldsymbol{w} = (b, w_1, w_2, \ldots, w_k)$ and adding a constant feature to $\boldsymbol{x}$. Using ordinary least squares [HTF17] we can compute a closed-form solution to finding $\hat{\boldsymbol{w}}$ that minimizes Definition 2.1 with ℓ_{MSE} for a sample $(\boldsymbol{X}, \boldsymbol{y})$ as

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \frac{1}{N}\sum_{i=1}^{N} \ell_{\text{SE}}(f_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i) = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

In classification the image of f is a finite, discrete set, i.e., $\mathcal{Y} = \mathcal{C} = \{C_1, C_2, \ldots, C_n\}$. Consider for example the task of categorizing plants into three species based on measurements such as petal lengths [Fis36], i.e., $|\mathcal{C}| = 3$. However, instead of returning a fixed label, most models opt to model the conditional probability

$$f_{\boldsymbol{\theta}} = p(y|\boldsymbol{x}, \boldsymbol{\theta}).$$

Then, given a prediction $\hat{y} \in [0, 1]$ and the true label $y \in \{0, 1\}$ we can use *cross entropy loss* to measure the quality of a classification model:

$$\ell_{\text{CE}}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}). \tag{2.2}$$

We can adapt the Linear Regression model from earlier to work for classification. Specifically, we assume a binary classification task, meaning that there are two possible classes, i.e., $|\mathcal{C}| = 2$. Then, we can write the *Logistic Regression* as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{x}^T\boldsymbol{w} + b) = \frac{1}{1 + \exp(-(\boldsymbol{x}^T\boldsymbol{w} + b))}, \quad \boldsymbol{\theta} = \{\boldsymbol{w}, b\}. \tag{2.3}$$

Notice that the Logistic Regression can be interpreted as a Linear Regression with an additional non-linear *sigmoid* or *logistic* function $\sigma : \mathbb{R} \to [0, 1]$. The output of Logistic Regression is then the probability of class 1, i.e., $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = p(y = 1|\boldsymbol{x}, \boldsymbol{\theta})$. As there is no closed-form solution to fitting the Logistic Regression to the training data we can use the

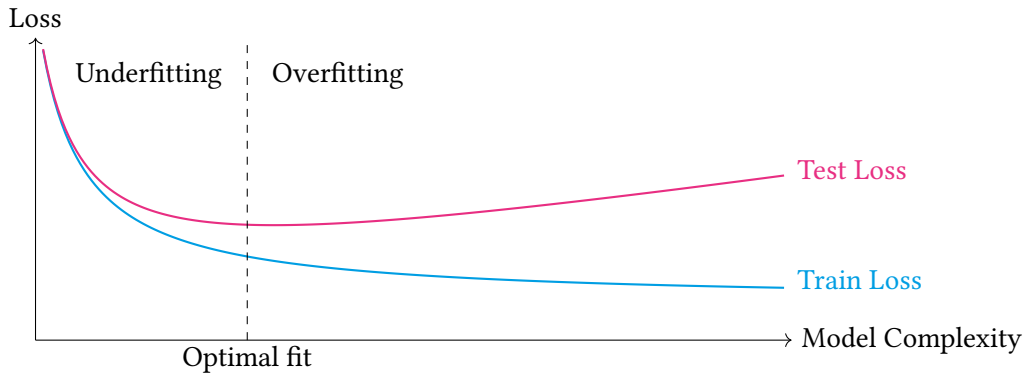

Figure 2.1.: Illustration of underfitting and overfitting as shown via the train and test loss. While the train loss keeps shrinking as the Model Complexity increases the loss on unseen data (Test loss) not only stops decreasing but actually increases again.

approach of *Maximum Likelihood Estimation*. The likelihood $\mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$ of the model with a specific parameter set $\boldsymbol{\theta}$ for a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ of size m is given via

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \prod_{(\boldsymbol{x},y=1)\in\mathcal{D}} f_{\boldsymbol{\theta}}(\boldsymbol{x}) \prod_{(\boldsymbol{x},y=0)\in\mathcal{D}} (1 - f_{\boldsymbol{\theta}}(\boldsymbol{x})).$$

Thus, the higher the likelihood is that the model parameters agree with the training the better our model fits the data. Equivalently, we can consider minimizing the negative log-likelihood, leading to the following minimization problem:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \; -\log(\mathcal{L}(\boldsymbol{\theta}|\mathcal{D})).$$

Notice that minimizing the negative log-likelihood is identical to minimizing the empirical risk when using the cross entropy loss function from Eq. (2.2):

$$\begin{aligned} -\log(\mathcal{L}(\boldsymbol{\theta}|\mathcal{D})) &= -\left(\sum_{(\boldsymbol{x},y=1)\in\mathcal{D}} \log(f_{\boldsymbol{\theta}}(\boldsymbol{x})) + \sum_{(\boldsymbol{x},y=0)\in\mathcal{D}} \log(1 - f_{\boldsymbol{\theta}}(\boldsymbol{x})) \right) \\ &= -\sum_{i=1}^{m} y_i \log(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) + (1 - y_i) \log(1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) \\ &= \sum_{i=1}^{m} \ell_{\text{CE}}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i). \end{aligned}$$

At this point, we can utilize multiple methods from literature to solve the optimization problem, such as the *Newton-Raphson* algorithm [HTF17].

Generalization and Overfitting

One downside of the empirical risk minimization approach is that we want to find a set of parameters $\boldsymbol{\theta}$ that truly learns the connection between $\mathcal{X}$ and $\mathcal{Y}$, i.e., we want to find

a set of parameters which *generalizes* well to unseen data and does not *overfit* on the training data. If we use all available data for training there is no notion of how well the trained model will perform on unseen data that we will encounter in deployment. Thus, we are at risk of *overfitting* to the training data [HTF17]. Assume we now have two distinct sets $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$. Then we can use only $\mathcal{D}^{\text{train}}$ for training the model and use unseen examples from $\mathcal{D}^{\text{test}}$ for evaluation. It has been observed in practice that the more trainable parameters a model has the easier it is to overfit to training data, essentially remembering each data point instead of finding a general mapping between input and output domain. This behavior is visualized in Fig. 2.1. One way to try and get a good estimate of the true risk *cross validation*, where all available data samples are repeatedly partitioned into two parts. Then, one partition is used to train the model and the other to evaluate. By averaging the loss over all splits we get a good estimate of the loss. Naturally, this is very time-intensive for large datasets and longer model training times.

Self-Supervised Learning

In the previous supervised setting there was always a label given as an external signal, such as a corresponding regression value or a discrete class. Consider the example given earlier of classifying plant types. To do this, an expert had to note the correct type of each plant associated with the corresponding feature vector. While human annotation, especially by experts, can be considered the gold standard in many applications, it is not without two major flaws:

1. Two (or more) people annotating the same data point could come to different conclusions about the correct label. In some research fields this is known as Interrater (dis)agreement [GBC13] and can be highly problematic. Consider as an example the task of curating a dataset where patients are categorized as having a disease or being healthy. If the agreement between the doctors labeling the dataset is low then the noise added will likely hinder the quality of learned models, leading to faulty predictions.

2. Annotating data can take a long time. Labeling hundreds or thousands of data points is not only tedious but expensive, especially if it is the done by high-paid experts such as doctors or other domain experts.

Self-supervised learning is an approach to learn from unlabeled data, where the target (i.e., the label) is constructed out of the data itself. Naturally, this leads to larger quantities of available data as the costly labeling process is not needed. For self-supervised learning the same optimization strategies apply as in the supervised learning task. We will use the example of learning data compression to illustrate this approach.

Let's consider a matrix $\boldsymbol{X} \in \mathbb{R}^{M \times K}$, consisting of M rows of K-dimensional vectors. In addition, we will define two parameterized functions $E_{\boldsymbol{\theta}'} : \mathbb{R}^k \to \mathbb{R}^l$ and $D_{\boldsymbol{\theta}''} : \mathbb{R}^l \to \mathbb{R}^K$ which we will refer to as the *encoder* and *decoder*, respectively. Collectively, this architecture is often referred to as an Autoencoder (AE). The encoders task is to project

each K-dimensional input vector $\boldsymbol{x}_i$ onto a representation of dimensionality $l < K$. The decoder, on the other hand, is trained to project the low dimensional representation back into the K-dimensional input space. As a quality measure we can define *reconstruction error*, which is the MSE between input data and the output of the decoder. We can denote the MSE equivalently using the squared L_2 norm:

$$\hat{\boldsymbol{\theta}}', \hat{\boldsymbol{\theta}}'' = \arg\min_{\boldsymbol{\theta}', \boldsymbol{\theta}''} \frac{1}{M} \sum_{i=1}^{M} \| D_{\boldsymbol{\theta}''}(E_{\boldsymbol{\theta}'}(\boldsymbol{x}_i)) - \boldsymbol{x}_i \|_2^2.$$

By enforcing that $l < K$ the model has to learn an efficient representation that *compresses* the input in a way that preserves as much information as possible. Thus, the learned representation (also called *latent representation* or *encoding*) is a lossy compression of the input space. Notice that there is no need to assign a label to each data point since the data itself already provides the label. Notably, time-series forecasting can be seen as a self-supervised learning task (as will be discussed in Section 2.3.2).

Unsupervised learning

Similar to self-supervised learning there are not labels given in *unsupervised learning*. However, in comparison to self-supervised learning where the label is constructed from the data itself there is no clear target in unsupervised learning. As an example, we will discuss one very popular application of unsupervised learning, namely *clustering*. Specifically, we will discuss the k-means approach [Llo82].

We assume a set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$ of m-dimensional data points, each of which we want to assign to one of k clusters. Thus, we aim to find a partition $\mathcal{S} := \{\mathcal{S}_j : j \in [k]\}$ of $\mathcal{D}$ where the *cluster center* $\boldsymbol{\mu}_j$ of each cluster $\mathcal{S}_j$ is given by the average of the points in the cluster, i.e.,

$$\boldsymbol{\mu}_j = \frac{1}{|\mathcal{S}_j|} \sum_{\boldsymbol{x}_i \in \mathcal{S}_j} \boldsymbol{x}_i.$$

If some data point $\boldsymbol{z}_i$ is closest to a cluster center μ_j it will be added to $\mathcal{S}_j$. To measure closeness we can use the *euclidean distance*, given by the L_2 norm of differences:

$$\|\boldsymbol{x} - \boldsymbol{z}\|_2 = \sqrt{\sum_{i=1}^{m} (x_i - z_i)^2}.$$

Putting it all together, we can define k-means clustering as finding a solution to the following optimization problem:

Definition 2.2 (k-means clustering)**.** *Given a set of data points $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, the euclidean distance function $\|\cdot\|_2$ and a number of desired clusters k the objective of k-means clustering is to solve*

$$\arg\min_{\boldsymbol{\mathcal{S}}} \sum_{j=1}^{k} \sum_{\boldsymbol{x}_i \in \mathcal{S}_j} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2.$$

Since the algorithm from [Llo82] starts by assigning random cluster centers and works in an iterative fashion it is common to restart the clustering algorithm multiple times.

2.3. Time-Series Data

In many areas of research and industry we find data that has a temporal component [LZ21; Ull+20; Ser+22]. That is, we have T consecutive samples which are measurements of some quantity at equidistant moments in time. For example, consider the temperature of a room measured by a sensor every day at 9 am for T days. Alternatively, we can think of a video, recorded at 30 frames per second for T seconds, as a time-series of images of length $30T$.

Machine Learning approaches specifically developed for time-series data cover (among others) classification, regression and clustering approaches. In time-series classification [Ism+19; Bag+17] the goal is to classify entire time-series into discrete classes. Consider for example the classification of EEG signals to help in making medical diagnoses [BKM15]. Also, the field of time-series regression is concerned with predicting a specific numeric value for each time-series. Examples include the prediction of crop yield from satellite images or predicting heart rate from time-series corresponding to sensors attached to patients [Tan+21]. Also, the notion of clustering time-series is an important practice when, for example, analyzing patient recordings or in physics [ASW15]. Most importantly for this work, time-series forecasting is used in many application settings to predict (in advance) how the time-series will evolve, such as in weather prediction or financial applications. We will introduce forecasting more formally (with concrete examples) in Section 2.3.2.

For the purposes of this thesis we will restrict ourselves to time-series that consist of vectors of real numbers, which we refer to as univariate and multivariate time-series.

Definition 2.3 (Multivariate and Univariate time-series)**.** *A multivariate time-series*

$$\boldsymbol{X} = (\boldsymbol{X}_t)_{t=1}^{T} = (\boldsymbol{X}_1, \ldots, \boldsymbol{X}_T)$$

of length T is a sequence of vectors $\boldsymbol{X}_t \in \mathbb{R}^M$. If $M = 1$ we say that the time-series is univariate.

These time-series consist of one (in the case of univariate time-series) or multiple measurements of real values over time. In the case of multivariate time-series we will refer to the number of values M measured at each time-step as the number of *channels*. We will denote with $\boldsymbol{X}_i$ the i-th entry (with $i \in [T]$). Also, we will denote with $\boldsymbol{X}_{i:j}$ the subseries $(\boldsymbol{X}_i, \boldsymbol{X}_{i+1}, \ldots, \boldsymbol{X}_j)$. Each entry is time-aligned, meaning that the M measurements are assumed to have taken place at the exact same time. Through the use of superscript we notate the use of a specific channel, i.e., X_i^c is the value of channel c at time t. Analogous to the time-index notation, we will denote with $\boldsymbol{X}^{\mathcal{S}}$ the time-series of some channel subset $\mathcal{S} \subseteq \mathcal{M}$ with $|\mathcal{M}| = M$.

When analyzing time-series it can be useful to view them as realizations from an underlying discrete-time *stochastic processes* [Bor17].

Definition 2.4 (Stochastic process)**.** *A discrete-time stochastic process is defined as*

$$\boldsymbol{Y} = (\boldsymbol{Y}_t)_{t=1}^{T} = (\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_T)$$

where each $\boldsymbol{Y}_i$ with $i \in [T]$ is a random variable from some probability space $(\Omega, \mathcal{F}, \mathcal{P})$.

Analogous to defining a time-series as a realization of a stochastic process we also sometimes refer to the stochastic process as the *underlying process* of a time-series [Ham94]. Note that we will focus on discussions of realizations for most of the thesis and merely use the underlying process for definitions in this chapter.

Many machine learning methods that we will use in this thesis require a fixed-length input to process. This can be a problem in the domain of time-series data as each individual time-series might have different lengths. The number of parameters of methods such as Neural Networks also scales with the dimensionality of the input (which, in the context of time-series data is the length of the series). Thus, it is a popular method to convert time-series data into smaller, fixed-size *windows*, each of which can be processed individually. We assume that we want to compute the windows of size L for some time-series $\boldsymbol{X}$ of length T. Then, the first vector is given by the first L values of $\boldsymbol{X}$. Next, we start with the d-th value of $\boldsymbol{X}$ and, again, take the following L values for the second window. The parameter d is called the *stride* or *step-size* and controls the distance between two consecutive windows. This approach continues until the last value of $\boldsymbol{X}$ was used to form a window or the amount of values left is smaller than L. Notice that for $d < L$ consecutive windows are overlapping.

Definition 2.5 (Sliding window)**.** *Let $\boldsymbol{X} \in \mathbb{R}^{M \times T}$ be some multivariate time-series of length T. Further, let L be the desired length of each window and let d be the desired stride. Then, the vector of windows is given by*

$$W_{d,L}(\boldsymbol{X}) := \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix}, \quad \boldsymbol{x}_j := \left(\boldsymbol{X}_{(j-1)d+1}, \ldots, \boldsymbol{X}_{(j-1)d+L}\right).$$

In total, the sliding window approach will produce $m = \lceil \frac{T-L+1}{d} \rceil$ windows.

The sliding window approach is illustrated in Fig. 2.2 for some univariate time-series of length $T = 8$ and for a stride $d = 2$ and window size $L = 4$. The resulting matrix (on the right of the figure) has m rows and L columns where m is the number of windows.

2.3.1. Stationarity and Concept drift

Stationarity is a useful concept, especially when forecasting time-series data, since it requires that the characteristics the underlying stochastic process, such as mean and vari-

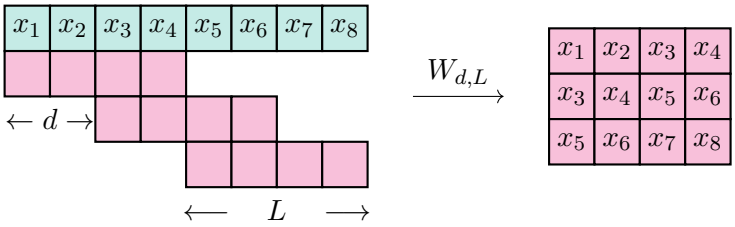

Figure 2.2.: Illustration of the sliding window approach for time-series of length 8 (in green) with window size $L = 4$ and a stride of $d = 2$. The resulting windows are shown in red.

ance, are not changing with time. Specifically, let $(\boldsymbol{X}_t)_{t=1}^T$ be a time-series which is a realization (i.e., sample) from its underlying stochastic process $(\boldsymbol{Y}_t)_{t=1}^T$. We will denote with $F_{\boldsymbol{Y}_t}$ the cumulative density function (CDF) of $\boldsymbol{Y}_t$. For notational convenience we will omit the subscript from F if the random variables are clear from context.

Definition 2.6 (Strong-stationarity [Bor17])**.** *Let $\boldsymbol{Y} = (\boldsymbol{Y}_i)_{i=1}^T$ be a stochastic process and let $\boldsymbol{X} = (\boldsymbol{X}_i)_{i=1}^T$ be the corresponding time-series sample. Then, the process is said to be strictly stationary if and only if*

$$F(\boldsymbol{X}_{i+\tau:i+\tau+t}) = F(\boldsymbol{X}_{i:i+t}) \quad \forall i, t, \tau \in \mathbb{N} : i + \tau + t \leq T.$$

That is, a time-series is said to be strong-stationary if its distribution characteristics are time-invariant. Naturally, the fact that all these characteristics are independent of time is a fairly strong assumption in reality. It is often more reasonable to assume that a process is *weak-stationary*, given by the following, more relaxed, definition:

Definition 2.7 (Weak-stationarity [Ham94])**.** *Let $\boldsymbol{Y} = (\boldsymbol{Y}_i)_{i=1}^T$ be a stochastic process. Let $\mathbb{E}[\boldsymbol{Y}_t]$ be the expected value of $\boldsymbol{Y}_t$. Then, the process is said to be weakly stationary if all of the following are satisfied:*

$$\begin{aligned} \mathbb{E}[\boldsymbol{Y}_t] &= \mu \quad \forall t \in [T] \\ \mathbb{E}[(\boldsymbol{Y}_t - \mu)(\boldsymbol{Y}_{t-j} - \mu)] &= \gamma_j \quad \forall t \in [T], j \in t+1, \ldots, T \\ \mathbb{E}[(\boldsymbol{Y}_t)^2] &< \infty \quad \forall t \in [T]. \end{aligned}$$

Weak-stationary thus only assumes three things about the characteristics of the time-series: That the expected value and autocovariance functions (given by $\mathbb{E}[(\boldsymbol{Y}_t - \mu)(\boldsymbol{Y}_{t-j} - \mu)]$) do not vary with time, and that the second moment of each random variable is finite. Naturally, if a strong-stationary time-series has finite second moments this will also mean that the time-series is weak-stationary.

While both stationarity assumptions help in modeling and analyzing the time-series, they are often not valid in real-world scenarios. For example, consider the number of customers of a bookshop, recorded each day, as a time-series. One might find a strong

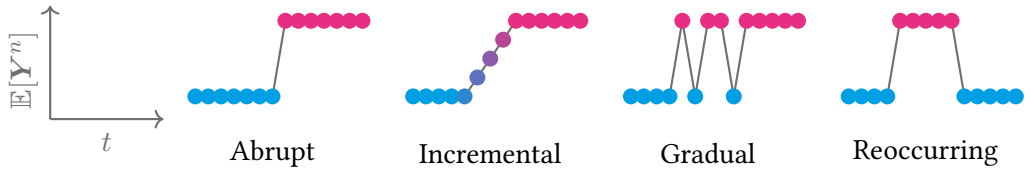

Figure 2.3.: Illustration of four types of concept drift. Time is shown on the x axis, while some moment of the underlying process such as mean or variance is shown on the y axis. One concept is shown in blue while another concept is shown in red. Adapted from [Gam+14].

seasonal component around the Christmas holidays when more customers might shop for presents. It thus cannot hold that the expectation is independent of time. On the other hand, consider a sensor placed on top of a machine in a factory, which monitors the vibrations of that machine. If the sensor accidentally gets touched it might be placed slightly differently compared to before. This can drastically change the characteristics of the recorded vibration patterns. Lastly, consider a time-series of temperature of a particular place over time. Due to climate change, the underlying characteristics will change over time, leading, among other things, to a slow but steady drift towards higher temperatures.

A useful framework to understand and classify these kinds of changes, be they rapid or slow, is that of *concept drift* [SG86; WK96; Gam+14]. Since we are concerned with drifts in the context of time-series forecasting (and thus are only interested in changing of the data distribution) we will define concept drift as what Gama et al.[Gam+14] defines as *virtual drift*. That is:

Definition 2.8 (Concept drift)**.** *Let $(\boldsymbol{Y}_i)_{i=1}^T$ be a stochastic process, $(\boldsymbol{X}_i)_{i=1}^T$ be the corresponding time-series sample, $t_0, t_1 \in [T]$ be two indices of the process and let $F_{\boldsymbol{Y}}$ be the CDF of $\boldsymbol{Y}$. Then, there is a concept drift between t_0 and t_1 if*

$$F_{\boldsymbol{Y}_{t_0}}(\boldsymbol{X}_{t_0}) \neq F_{\boldsymbol{Y}_{t_1}}(\boldsymbol{X}_{t_1}).$$

The authors in [Gam+14] define four general types of concept drift which we will outline and are visualized in Fig. 2.3 :

- **Abrupt**: The characteristics of the time-series change suddenly at some point.
- **Incremental**: Over a certain period of time the characteristics of the time-series change slowly from one concept to another, eventually settling. During this intermediate phase a mixture of two concepts is observed.
- **Gradual**: Over time, more and more of a new concept is observed until the new concept is entirely present.
- **Reoccuring**: After switching from some concept A to another concept B, the time-series returns to the initial concept A.

Notice that we, in accordance with Gama et al. [Gam+14], do not consider outliers as concept drifts, even though they technically fit the definition in Definition 2.8. This is due to the fact that outliers are short, erroneous changes to the data distribution which do not last for a long time. Concepts, however, are defined as different states which have some longevity to them.

Finally, we want to quickly discuss two important time-series characteristics which will be important when talking about forecasts in the upcoming section. These two components are *trend* and *seasonality*. As the expected value of time-series with a trend and time-series with seasonality is a function of time in both cases time-series with either (or both) characteristics can not be stationary. The discussion of whether there are concept drifts present between two points in a time-series with trend and/or seasonality is more complicated. For seasonal data, we could classify each distinct season as a particular concept which will **reoccur** over time (see right-most example in Fig. 2.3). Additionally, trend can be an indicator of a transition between two stable concepts, which could indicate an **incremental** drift (see second from the left example in Fig. 2.3).

2.3.2. Time-series Forecasting

Given that time-series data is present in many industry applications and fields of research it is often desired to try and anticipate future, unobserved values, given historical data. In the context of time-series research this field is called time-series forecasting and has been a common field of research for decades [Hol04; Win60; HA21].

In time-series forecasting the goal is to predict, at some time t, the following H values of a time-series, given the past L values. We will refer to the H predicted values as the *forecasting horizon*, and to the known L past values as the *context* or *lagged values*. For notational convenience, we will extend the sliding-window notation from Definition 2.5 to include the future values we want to model.

Definition 2.9 (Sliding window for forecasting)**.** *Let $\boldsymbol{X} \in \mathbb{R}^{M \times T}$ be some multivariate time-series of length T. Further, let L be the desired length of each window, let H be the desired forecasting horizon and let d be the desired stride. Then, the vector of windows is given by*

$$W_{d,L,H}(\boldsymbol{X}) := \begin{pmatrix} (\boldsymbol{x}_1, \boldsymbol{y}_1) \\ (\boldsymbol{x}_2, \boldsymbol{y}_2) \\ \vdots \\ (\boldsymbol{x}_m, \boldsymbol{y}_m) \end{pmatrix}, \qquad \begin{aligned} \boldsymbol{x}_j &:= (\boldsymbol{X}_{(j-1)d+1}, \quad \dots, \boldsymbol{X}_{(j-1)d+L}) \\ \boldsymbol{y}_j &:= (\boldsymbol{X}_{(j-1)d+L+1}, \quad \dots, \boldsymbol{X}_{(j-1)d+L+H}). \end{aligned}$$

In total, the sliding window approach will produce $m = \lceil \frac{T-L+H+1}{d} \rceil$ windows.

We also illustrate an example sliding window result in Fig. 2.4. Notice that, while the number of time steps of the original time-series is identical to that of Fig. 2.2 the resulting number of windows m is different since the horizon H has to be accounted for.

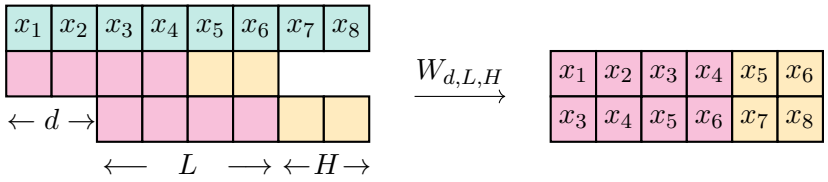

Figure 2.4.: Illustration of the sliding window approach for forecasting, where the time-series of length 8 (in green) is separated into windows (shown in red) of L lagged values ($L = 4$ here) followed by the next $H = 2$ values which make up the forecasting horizon (shown in yellow). Also, note that the stride d is equal to 2.

After introducing basic notation for the time-series forecasting setting we can proceed in defining forecasting formally.

Definition 2.10 (Time-series forecasting)**.** *Let $\boldsymbol{X}$ be a (multivariate) time-series of length T and let $H, L \in \mathbb{N}^+$ be the forecasting horizon and number of lagged values, respectively. Then, for some window index $t \in [m]$, the forecast of a forecaster f is given by*

$$\hat{\boldsymbol{y}}_t = f(\boldsymbol{x}_t), \quad (\boldsymbol{x}_t, \cdot) \in W_{d,L,H}(\boldsymbol{X}).$$

Essentially, we can frame time-series forecasting as a learning task that requires to learn a predictor from past data given in the form of a fixed-length representation. While we want to note that not all forecasters require a fixed window size and can in theory process a variable length time-series we will mostly talk about models that require a fixed-length representation in this thesis.

Statistical Methods

As mentioned above, time-series forecasting has been a research field for decades. Today, the more traditional approaches are often labeled *statistical methods* in the literature, specifically to separate them from modern, DL-based approaches. In this subsection we will discuss some important models of this family, as they will be used as important baselines during later experiments. For the rest of this chapter we will assume a univariate time-series $\boldsymbol{X} = (X_1, X_2, \ldots, X_t)$ of length t and $H = 1$ unless otherwise stated, since that is the context in which most of these methods were originally derived.

Baselines Before discussing any method in detail we would like to highlight some baselines against which one can compare newly developed forecasting methods. They can serve as simple sanity checks to test whether more elaborate methods actually perform better, which is not always easy to tell in forecasting. This is due to the fact that we often discuss forecast performance in terms of absolute or squared error measures, which can be hard to interpret.

First, the most simple baseline to predict a future value is to predict the last known value. Assume $\boldsymbol{X}$ where X_t is the last observed value of the time-series. Then, for any $H \in \mathbb{N}$ we could predict

$$X_{t+h} = X_t.$$

When we reference this method later in the thesis during benchmarking we will refer to it last the *last value* baseline. Next, instead of predicting the last known value one could decide to predict based on the average of the last p known value, i.e.,

$$X_{t+H} = \frac{1}{p} \sum_{i=0}^{p-1} X_{t-i}.$$

This baseline will be referred to as *mean value*, accordingly. When a certain seasonality is known about the time-series, we could compute the *seasonal naive* forecast. Consider, for example, a time-series of sales in a store. It might be reasonable to assume that the predicted number of sales for next Friday is fairly similar to the sales of previous Friday. If we assume a seasonality of s steps we can write the seasonal-naive forecaster as

$$X_{t+1} = X_{t-s+1}.$$

ARIMA A still widely used method to forecast univariate time-series is the *ARIMA* model [HA21]. It consists of three parts, namely an *autoregressive* (AR) process, an *integration* step (I) and a *moving average* (MA) process. The autoregressive AR(p) process models X_{t+1} as a linear function of past p values, i.e.,

$$X_{t+1} = \vartheta_0 + \epsilon_t + \sum_{i=1}^{p} \vartheta_i X_{t-i+1} = \vartheta_0 + \epsilon_t + \boldsymbol{\vartheta}^T \boldsymbol{X}_{t-p+1:t}, \quad \epsilon_t \sim \mathcal{N}(0, 1). \tag{2.4}$$

Notice that the AR(p) method is a linear function in its input. Additionally, the process is assumed to have a Gaussian noise component, denoted with ϵ_t.

The moving average MA(q) process models X_{t+1} as a linear function of the past q forecasting errors, i.e.,

$$X_{t+1} = \phi_0 + \sum_{j=1}^{q} \phi_j \epsilon_{t-j+1} = \phi_0 + \boldsymbol{\phi}^T \boldsymbol{\epsilon}_{t-q+1:t}. \tag{2.5}$$

By combining both models to a single model, we can define the ARMA($\boldsymbol{X}$) model of order (p, q), given by

$$\text{ARMA}_{\boldsymbol{\theta}}(\boldsymbol{X}) = \vartheta_0 + \boldsymbol{\vartheta}^T \boldsymbol{X}_{t-p+1:t} + \phi_0 + \boldsymbol{\phi}^T \boldsymbol{\epsilon}_{t-q+1:t}, \quad \boldsymbol{\theta} = \{\boldsymbol{\vartheta}, \boldsymbol{\phi}\}.$$

In order to overcome possible non-weak-stationary time-series, such as time-series with a trend, the ARIMA model can also utilize *integration* or *differencing*. For example, first order differencing is defined as

$$X_t^{(1)} := X_t - X_{t-1}.$$

In general, we define d-th order differencing as

$$X_t^{(d)} := X_t^{(d-1)} - X_{t-1}^{(d-1)}$$

where $X^{(0)} := X$. By combining all three components, we can define the full ARIMA(p, d, q) model:

Definition 2.11 (ARIMA)**.** *Given a time-series* $\boldsymbol{X} = (X_1, X_2, \ldots, X_t)$ *and* $p, d, q \in \mathbb{N}$ *the ARIMA model is defined as*

$$\text{ARIMA}_{\boldsymbol{\theta}}(\boldsymbol{X}) = \vartheta_0 + \boldsymbol{\vartheta}^T \boldsymbol{X}_{t-p+1:t}^{(d)} + \phi_0 + \boldsymbol{\phi}^T \boldsymbol{\epsilon}_{t-q+1:t}, \quad \boldsymbol{\theta} = \{\boldsymbol{\vartheta}, \boldsymbol{\phi}\}.$$

After setting the desired hyperparameters p, d, q the parameters of the model can be fitted using Maximum Likelihood Estimation (MLE) on historic data. In order to generate forecasts using the model we can produce the next value forecast X_{t+1} using all available historical data necessary. Since the prediction errors $\boldsymbol{\epsilon}$ are not known, we will ignore them for the first forecast. Once we obtain the forecast $\hat{X}_{t+1}$ we can utilize it to predict X_{t+1} by plugging it in to the autoregressive part as well as computing the known residual $\epsilon_{t+1} = X_{t+1} - \hat{X}_{t+1}$. There are also extensions of the ARIMA model incorporating information about seasonality [HA21] and adapting it to the multivariate case [APV12].

Exponential Smoothing Exponential smoothing [Win60; Hol04; Hyn+02][3] is similar to the AR(p) process discussed earlier since it also computes the forecast of X_{t+1} given the previous p known values $X_t, X_{t-1}, \ldots, X_{t-p}$. However, it computes a weighted average, assuming that the importance of time-steps decreases exponentially with time. This means that more recent values are exponentially more important than older ones.

Definition 2.12 (Simple Exponential Smoothing)**.** *Let* $a \in [0, 1]$ *and* $k \in \mathbb{N}$*. Then, simple exponential smoothing is given by*

$$X_{t+1} = \sum_{k=0}^{K} a(1-a)^k X_{t-k}.$$

Exponential Smoothing can also be easily extended to incorporate seasonal components [Hol04; Win60] as well as forecast time-series with trend [Hol04].

[3] Note that the citation of Charles C. Holt's work is of a reprint of the original 1957 report to the Office of Naval Research (ONR 52).

2.4. Machine Learning models for Time-series data

ARIMA and ETS models, discussed in the previous section, are good choices for capturing linear patterns and short-term dependencies for stationary time-series data. However, they often struggle to perform well when faced with complex, non-linear temporal relationships, seasonality variations, or long-term dependencies. As a result, there is a growing interest in more advanced techniques, such as DL and tree-based methods, which offer high flexibility and robustness in handling the forecasting of more complex time-series.

Deep learning models, with their ability to automatically learn complex patterns from large datasets, have shown great promise in capturing long-term dependencies and non-linear relationships useful for time-series forecasting [LZ21]. Convolutional Neural Networks (CNNs) and Long Short-Term Memorys (LSTMs) excel at handling high-dimensional sequential data but require substantial computational resources and large datasets for effective training [FS24].

On the other hand, tree-based methods, including Decision Trees (DTs), Random Forests (RFs) and Gradient Boosting Trees (GBTs), offer a more interpretable and efficient alternative. While they might not capture temporal dependencies as effectively as DL models, they provide a balance between accuracy and computational efficiency, especially for smaller datasets (as we will discuss in Chapter 5).

In this section, we will discuss the basics of Deep Learning with a focus on architectural choices useful in analyzing time-series data. Afterwards, we outline the training of Decision Trees for regression and classification, as well as the ensembled models RFs and GBTs.

2.4.1. Deep Learning

Deep learning is often used to describe large, non-linear models being trained on a large amount of data. This is achieved with the help of Neural Networks, a family of Machine Learning models characterized by being able to fit arbitrary functions by minimizing a loss function using Backpropagation and Gradient Descent. Due to their modular and flexible architectures they are easily adaptable to different tasks. Next, we will introduce and discuss the most widely adopted and simplest kind of Neural Networks fist, namely the Fully-connected Network (FCN), and how to fit its parameters.

Fully-connected Networks

As outlined previously, we consider models as parameterized functions $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ from some input domain $\mathcal{X}$ to some output domain $\mathcal{Y}$. The same holds for Neural Networks. For simplicity, we will consider both input and output domains to be vector spaces of real numbers, i.e., let $\mathcal{X} := \mathbb{R}^M$ and $\mathcal{Y} := \mathbb{R}^K$.

Before considering neural networks, we start by introducing the most elemental building block, which is a *neuron*. Recall from Eq. (2.1) that a linear model for some M-dimensional input $\boldsymbol{x} \in \mathbb{R}^M$ is given by

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b, \quad \boldsymbol{\theta} = \{\boldsymbol{w}, b\}.$$

Neurons work similarly in that they consist of learnable weights $\boldsymbol{w}$ and b that are used to compute the weighted sum of some given input. However, to introduce the ability to learn non-linear relations between input and output a neuron also consist of a differentiable, non-linear *activation function* $\varphi : \mathbb{R} \to \mathbb{R}$. Common activation functions found the literature are the sigmoid and Rectified Linear Unit (ReLU) [NH10], given by

$$\sigma(x) = (1 + \exp(-x))^{-1}, \quad \text{ReLU}(x) = \max(0, x).$$

Note that, from here on, we will use a slight abuse of notation and also use φ as an element-wise operation in case of vectors and matrices. Additionally, note that (when using the sigmoid activation function) a neuron is identical to a Logistic Regression (see Eq. (2.3)).

Neural Networks are typically defined as having *layers* of operation that are applied sequentially on its input. In a FCN each layer consists of a fixed number of the neurons outlined above. Each neuron processes the input from the previous layer (or the input to the FCN itself in case of the first layer) and computes its activation. Then, the output of the layer consists of all neuron activations, which itself serve as the input for the following layer. More formally, let $\boldsymbol{x} \in \mathbb{R}^M$ be some M-dimensional feature vector which will serve as input to a FCN with L layers. Then, the l-th layer is defined as

$$f_{\boldsymbol{\theta_l}}(\boldsymbol{x}) = \varphi\left(\boldsymbol{W}_l \boldsymbol{x} + \boldsymbol{b}_l\right), \quad \boldsymbol{\theta}_l = \{\boldsymbol{W}_l, \boldsymbol{b}_l\}.$$

We utilized vector notation for simplicity, meaning that $\boldsymbol{W}_l \in \mathbb{R}^{M \times N}$ and $\boldsymbol{b}_l \in \mathbb{R}^N$. In other words, a layer of neurons computes N different logistic regression outputs on the same input. If we denote with $\circ$ the function composition operator, we can define the architecture of a FCN as follows:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = (f_{\boldsymbol{\theta}_1} \circ f_{\boldsymbol{\theta}_2} \circ \cdots \circ f_{\boldsymbol{\theta}_L})(\boldsymbol{x}), \quad \boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L\}. \tag{2.6}$$

Note that while we assume the NN architecture to be purely sequential for notational convenience this is not necessarily the case in practice as more complex architectures are possible (as long as the model remains fully differentiable).

Depending on the task that the FCN is used for the last layer is designed in a specific way. In classification settings the number of neurons in the last layers corresponds to the number of classes that can be predicted and the element-wise activation function is substituted with the *softmax* S activation, given by

$$S_i(\boldsymbol{x}) = \frac{\exp(x_i)}{\sum_{j=1}^{M} \exp(x_j)}.$$

The softmax's function is to project the output of the last layer onto the probability simplex, allowing for each output to be interpreted as a probability for that class. The actual prediction is then chosen as the class with the highest probability. In a regression setting, on the other hand, the activation function on the last layer is often omitted in cases where the regression target is not known to be in some specific domain.

Learning the model parameters

Neural Network models are commonly trained using an approach called *gradient descent*. Gradient descent minimizes a differentiable function by iteratively adapting each of the functions parameters in negative gradient direction which minimizes the function's value in the limit. Formally, let $(\boldsymbol{X}, \boldsymbol{y})$ be a matrix of feature vectors and corresponding labels, let ℓ be some differentiable loss function and let $\boldsymbol{\theta}$ be the model weights. We will refer to $(\boldsymbol{X}, \boldsymbol{y})$ as the *train set*. Then, we can define the gradient $\nabla\ell(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\theta})$ of our loss function ℓ w.r.t. the model parameters as

$$\nabla\ell(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\theta}) = \left(\frac{\partial\ell(f_{\boldsymbol{\theta}}(\boldsymbol{X}), \boldsymbol{y})}{\partial\theta_i}\right)^T_{i=1,\dots,|\boldsymbol{\theta}|}.$$

Given the gradient we can deduce how to update each θ_i to decrease the loss. For example, if the partial derivative w.r.t. θ_i at time t is positive we know that increasing θ_i would increase the loss function as well. Since we are interested in reducing the loss, we can thus change θ_i in negative gradient direction. If we start with some random model parameters $\boldsymbol{\theta}^{(0)}$ at time $t = 0$, the iterative update (for some time t) is given by

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \lambda\nabla\ell(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\theta}^{(t)})$$

where λ is the so-called *step-size*. The step size is a hyperparameter that indicates how strong these updates should be and which can have strong impact on training performance and convergence properties. To calculate the gradient efficiently, *backpropagation* is utilized, which allows to calculate the gradient for any arbitrary NN. This is done by calculating the gradient layer-wise, starting at the end of the network and propagating the gradient back-to-front until the desired parameter is reached via the chain rule of derivatives.

Computing the gradient w.r.t. the entire training set, as shown before, can be computationally limiting since the training dataset has to be loaded into memory in is entirety in order to utilize full parallelism for efficient training. Due to the prevalence of increasingly larger dataset most DL approaches opt to use *mini-batches* for training instead. Consider an independent sample of the training set of size B. Then, *stochastic* gradient descent computes the gradient w.r.t. this mini-batch, updates the parameter vector and moves on to the next batch. The batch size is usually taken as a hyperparameter which can be tuned separately. It can be shown that, under some assumptions, stochastic gradient descent approximates regular gradient descent.

Batch normalization To stabilize the training and speeding up convergence for stochastic gradient descent many neural network architectures utilize *batch normalization* [IS15]. Batch normalization is a layer which learns empirical mean and variance of the input batches, resulting in normalized batches for which gradients will saturate or vanish less frequently. Formally, let $\mathcal{B} := \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_B\}$ be a mini-batch of size B. Then, we define the empirical mean and variance as

$$\hat{\boldsymbol{\mu}} = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{x}_i, \quad \hat{\boldsymbol{\sigma}}^2 = \frac{1}{B}\sum_{i=1}^{B} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})^2.$$

Finally, the batch normalization layer normalizes the input data point $\boldsymbol{x}$ (which is part of batch $\mathcal{B}$) and scales it using learned parameters via

$$f_{\boldsymbol{\theta}_l}(\boldsymbol{x}) = \boldsymbol{\alpha}\frac{\boldsymbol{x} - \hat{\boldsymbol{\mu}}}{\sqrt{\hat{\boldsymbol{\sigma}} + \epsilon}} + \boldsymbol{\beta}, \quad \boldsymbol{\theta}_l = \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}.$$

Dropout One common downside of using larger and larger DL architectures is that they can easily *overfit* on the training data (see Fig. 2.1). To combat overfitting NN architectures often utilize *dropout* during training [Hin+12]. Dropout randomly shuts of neurons during the forward pass of a batch, thus forcing the model to adapt and optimize different paths through the network. Empirically, it has been shown that dropout can reduce the tendency to overfit and make the process of learning more stable. More formally, a dropout layer can be implemented via

$$f(\boldsymbol{x}) = \boldsymbol{M} \odot \boldsymbol{x}, \quad M_i \sim \text{Bin}(p)$$

where $\odot$ is the element-wise product and $\text{Bin}(p)$ is a Binomial distribution. Essentially, dropout samples (for each batch or sample) a binary mask which "drops out" part of the input. Notice that a Dropout layer does not have learnable parameters but requires the hyperparameter p to be set.

Convolutional Neural Networks

While FCNs are a popular and important type of Neural Network, their fixed arrangement of neurons is limiting on certain data types, such as on time-series data. Since FCNs are defined as matrix products of input and weight matrices they treat each input independently of the others. For example, concepts such as "three consecutive increasing values" require deeper networks to learn due to a lack of considering local dependencies. On the other hand, a *Convolutional Layer* [LeC+89] does not consist of a fixed number of neurons but of several *filters*. A filter is a matrix of learnable weights which gets convolved with the input along the temporal dimension.

Formally, let $\boldsymbol{x} \in \mathbb{R}^T$ be a univariate time-series of length T and $\boldsymbol{w} \in \mathbb{R}^L$ be a filter of length L. Then, we can define a discrete convolution of $\boldsymbol{x}$ with $\boldsymbol{w}$ as the sliding dot-product given by

$$\boldsymbol{x} * \boldsymbol{w} = \left(\boldsymbol{w}^T \boldsymbol{x}_{i:(i+L)}\right)_{i \in [T-L]}. \tag{2.7}$$

In general, convolutional layers consist of multiple filters, each of which gets convolved with the input sequence separately, resulting in one new output sequence per filter. Assume a convolutional layer with K filters of length L, leading to the weight matrix $\boldsymbol{W}_l \in \mathbb{R}^{K \times L}$. Additionally, each layer consists of a bias term, resulting in a bias vector $\boldsymbol{b}_l \in \mathbb{R}^K$. The operation that a convolution layer computes is thus given by

$$f_{\boldsymbol{\theta}_l}(\boldsymbol{x}) = \left(\varphi(\boldsymbol{W}_l \boldsymbol{x}_{i:(i+L)} + \boldsymbol{b}_l)\right)_{i \in [T-L]}, \quad \boldsymbol{\theta}_l = \{\boldsymbol{W}_l, \boldsymbol{b}_l\}. \tag{2.8}$$

In practice, there are slight adaptions to the operation of a convolutional layer, depending on the application context. For example, it is often desired that the output of a convolution has the same dimensionality as the input. To achieve this, the sequence is filled at the edges with so-called *padding elements*, often the real value 0 or copies of the last known values.

Pooling As stated above, Convolution layers preserve the time dimension (if padding is used) or slightly shrink the number of time steps with each operation. To aggregate the output over the time dimension and learn time-aggregated features it is common to use *pooling* operations. Pooling operations are defined as a function that takes L consecutive values and aggregates them to a single value. The most often used pooling operations are *max pooling* and *average* pooling. We can formalize both operations over windows (similar to Eq. (2.8)) as

$$f^{\text{Max}}(\boldsymbol{x}) = \left(\max(\boldsymbol{x}_{i:(i+L)})\right)_{i \in [T-L]}, \quad f^{\text{Avg}}(\boldsymbol{x}) = \left(L^{-1} \mathbf{1}^T \boldsymbol{x}_{i:(i+L)}\right)_{i \in [T-L]}.$$

Notice that pooling does not involve learning parameters. In fact, pooling can be interpreted as a fixed convolution with a stride equal to the window size, i.e., there is no overlap between windows.

Recurrent Neural Networks

An alternative approach to process sequential data is to process them using a *recurrent* approach. Recurrent layers process the input sequence one item at a time, producing a corresponding output token. Crucially, the layer contains a *memory* which is used to compute each output token and updated with new data. That way, temporal information from earlier tokens can be preserved and used during the computation of future tokens, making the layer a *network with memory* [Elm90]. We proceed by defining the general framework and give two concrete examples of recurrent layers.

Definition 2.13 (Recurrent Neural Network)**.** *A recurrent neural network layer is a differentiable, parameterized function which, given some input $\boldsymbol{x}$ and state $\boldsymbol{h}$, computes*

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{h}_t) = (\hat{\boldsymbol{y}}_t, \boldsymbol{h}_{t+1}).$$

The temporal subscript t could, in the case of input vector $\boldsymbol{x} \in \mathbb{R}^{C \times T}$, be defined as $\boldsymbol{x}_t \in \mathbb{R}^C$, i.e., the value of all channels at time t. Then, the output for time t will be computed, together with an update of the current hidden state $\boldsymbol{h}_t$. By recursively plugging in $\boldsymbol{x}$ and $\boldsymbol{h}$ the layer processes each input consecutively.

Let $\boldsymbol{x} \in \mathbb{R}^{C \times T}$ be some C-dimensional sequence of length T. As a simple example of a recurrent network, consider the network proposed by Elman [Elm90] which consists of three weight matrices $\boldsymbol{W}_h \in \mathbb{R}^{C \times H}, \boldsymbol{W}_y \in \mathbb{R}^{H \times C}, \boldsymbol{U}_h \in \mathbb{R}^{H \times H}$ and two bias vectors $\boldsymbol{b}_h \in \mathbb{R}^H, \boldsymbol{b}_y \in \mathbb{R}^C$. Additionally, the hidden output is activated with a non-linear activation function φ. The hyperparameter H defines the size of the hidden vector. The first hidden vector $\boldsymbol{h}_0$ needs to be initialized. The original paper proposes to initialize it to 0.5 for each entry [Elm90]. The function of the layer l is then given by

$$f_{\boldsymbol{\theta}_l}(\boldsymbol{x}_t, \boldsymbol{h}_t) = (\underbrace{\boldsymbol{W}_y \boldsymbol{h}_{t+1} + \boldsymbol{b}_y}_{=:\boldsymbol{y}_t}, \underbrace{\varphi(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{U}_h \boldsymbol{h}_t + \boldsymbol{b}_h)}_{=:\boldsymbol{h}_{t+1}}), \quad \boldsymbol{\theta}_l = \{\boldsymbol{W}_y, \boldsymbol{W}_h, \boldsymbol{U}_h, \boldsymbol{b}_h, \boldsymbol{b}_y\}.$$

Notice that the new hidden vector update $\boldsymbol{h}_{t+1}$ is calculated first. Then, it is used in the computation of the new state.

One of the problems encountered in training with recurrent layers is the tendency to face the *vanishing gradient problem* [Hoc91]. This phenomenon means that during the backpropagation process the gradient shrinks further and further, resulting in fewer and less information reaching the beginning of the network. This limits the depth of the network and thus the adoption of deep architectures. In fact, the vanishing gradient problem was one motivation to propose the ReLU activation over sigmoid activation in deeper NNs. This is due to the fact that the derivative of the sigmoid function is given by

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x)) \leq \frac{1}{4}.$$

As mentioned when discussing backpropagation earlier (Section 2.4.1) the chain rule is applied to calculate the gradients during training, meaning that repeatedly taking the derivative of sigmoid activations can only decrease the gradient magnitudes. One way to mitigate this phenomenon is thus to use a different activation function, such as ReLU [NH10], for which it holds that

$$\frac{\partial \mathrm{ReLU}(x)}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Long Short-Term Memory (LSTM) networks [HS97] tackle the vanishing gradient problem by incorporating three *gates* into the recurrent network function. These allow the

gradients to potentially flow uninhibited and thus omit the problem entirely. The LSTM computes three weight vectors which control the flow through the cell and can determine which information from previous states should be kept or not. Specifically, for the current input $\boldsymbol{x}_t$ and hidden state $\boldsymbol{h}_t$ it calculates

$$\begin{aligned}\boldsymbol{w}_f &:= f_{\boldsymbol{\theta}_f}(\boldsymbol{x}_t, \boldsymbol{h}_t) = \sigma(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_t + \boldsymbol{b}_f) \\ \boldsymbol{w}_i &:= f_{\boldsymbol{\theta}_i}(\boldsymbol{x}_t, \boldsymbol{h}_t) = \sigma(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_t + \boldsymbol{b}_i) \\ \boldsymbol{w}_o &:= f_{\boldsymbol{\theta}_o}(\boldsymbol{x}_t, \boldsymbol{h}_t) = \sigma(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_t + \boldsymbol{b}_o).\end{aligned}$$

Additionally, a new cell state $\tilde{\boldsymbol{c}}_t$ is calculated as

$$\tilde{\boldsymbol{c}}_t := g_{\boldsymbol{\theta}_c}(\boldsymbol{x}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{h}_t + \boldsymbol{b}_c).$$

Finally, the output is calculated using these quantities as

$$f(\boldsymbol{x}_t, \boldsymbol{h}_t, \boldsymbol{c}_t) = (\underbrace{\boldsymbol{w}_f \odot \boldsymbol{c}_t + \boldsymbol{w}_i \odot \tilde{\boldsymbol{c}}_t}_{=:\boldsymbol{c}_{t+1}}, \underbrace{\boldsymbol{w}_o \cdot \tanh(\boldsymbol{c}_t)}_{=:\boldsymbol{h}_{t+1}}).$$

Intuitively, the output of the gates $(\boldsymbol{w}_f, \boldsymbol{w}_i, \boldsymbol{w_o})$ can control (on different levels) which parts of the information to keep (when updating cell state and hidden state) and which parts to forget.

2.4.2. Tree-based Models

After a brief overview over Deep Learning methods in the previous subchapter we will now consider a different approach to supervised learning tasks, namely tree-based models. More specifically, we will start with Decision Trees (DTs), which have been a staple of ML for decades. DTs (and ensembles of DTs) are still popular methods today, achieving state-of-the-art performance in various application settings [MSA22; GOV22]. We will focus on the CART algorithm to construct DTs by Breiman et al. [Bre+84] in our description. Additionally, we will talk about techniques implemented in the popular implementation by `scikit-learn` [Ped+11] throughout this section.

Decision trees

DTs are directed, acyclic graphs which operate on the principle of learning axis-aligned splits in feature space. That is, a DT partitions the feature space into disjunct regions, each having a fixed prediction. Let $T = (\mathcal{V}, \mathcal{E})$ be a tree with nodes $\mathcal{V}$ and directed edges $\mathcal{E}$. We will denote with $r \in \mathcal{V}$ the root-node, i.e., the node of the tree for which traversal begins. Given some data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{x}_i \in \mathbb{R}^k$ and $|\mathcal{D}| = n$ the goal of each node $v \in \mathcal{V}$ is to split $\mathcal{D}$ into two groups. This splitting is done by iterating over all features of $\boldsymbol{x}$ and, for each feature i, determine the threshold τ which minimizes the (weighted) sum of *node impurities* $I(\mathcal{D}_{x_i \leq \tau}) + I(\mathcal{D}_{x_i > \tau})$. The definition of node impurity is task-dependent, i.e., there are different impurity measures for classification and regression

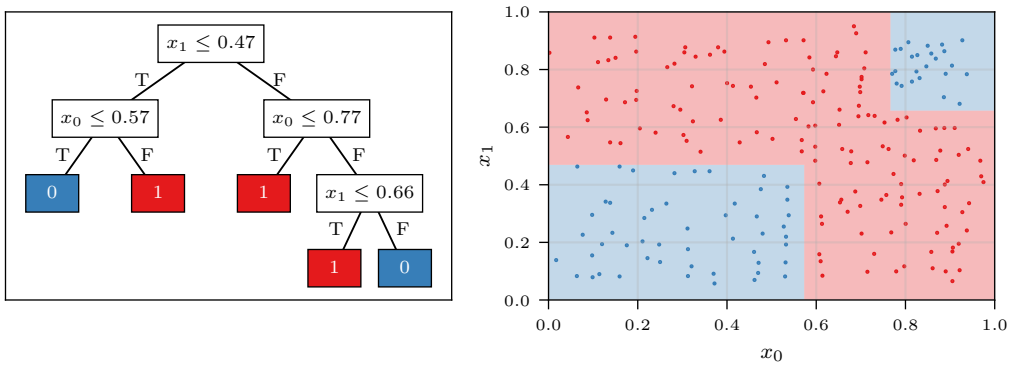

Figure 2.5.: Left: A Decision Tree trained on a two-dimensional binary classification dataset. **Right**: Visualization of the training samples (shown as dots) and the corresponding classes (shown in color). The background color corresponds to the learned DTs decision regions.

tasks. For classification, most impurity measures utilize the label proportion of data points in the node. For simplicity, we consider the *Gini index* $I_{\text{Gini}}(\mathcal{D})$ for binary classification tasks which is given as

$$I_{\text{Gini}}(\mathcal{D}) = 2p(1-p), \quad p = |\mathcal{D}|^{-1} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} \mathbb{I}[y_i = 1].$$

For regression tasks, *squared error* $I_{\text{SE}}(\mathcal{D})$ is an often used criterion. Here, the impurity is given by the sum of squared difference between each target y_i and the average over all y_i in the node, i.e.,

$$I_{\text{SE}}(\mathcal{D}) = \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} (y_i - \hat{y})^2, \quad \hat{y} = |\mathcal{D}|^{-1} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} y_i.$$

In order to choose the threshold all feature values found in the training set are first sorted (for example in ascending order). Afterwards, each feature value is taken as a threshold candidate and the corresponding impurities are calculated, choosing the optimal threshold. Thus, for each node the runtime to determine the optimal split is given by $\mathcal{O}(kn \log(n))$. Since there can only be a maximum of $\mathcal{O}(n)$ nodes (as no splitting is possible if a node consist of only one data point) the overall runtime is given by $\mathcal{O}(kn^2 \log(n))$.

After determining the optimal split and separating the dataset into two sets, each set is again recursively split by minimizing the impurity scores. This procedure continues until some stopping criterion is met. Common stopping criteria include the maximum depth of the tree or some minimal number of data points in a node. If a node is no longer split it is converted into a *leaf node*, with which the average label (in the regression case) or the majority class (in the classification case) is associated. For the latter case, there is also the option to aggregate class probabilities, i.e., associate with each leaf the fraction of positive

classes. Each leaf node thus corresponds to one disjunct region with fixed prediction in feature space. We will denote the set of leaf nodes with $\mathcal{L} \subset \mathcal{V}$. For illustration, consider the DT classifier shown in Fig. 2.5. On the left side, we visualized the tree, starting with the root node. For a new, unseen data point, we start traversal at the root node. For each node, if the criterion of the node is met, the data point traverses to the left, otherwise to the right. The leaf nodes are shown as colored boxes, with the corresponding majority class. On the right side of the figure, we visualized the corresponding partition of the feature space together with the training data used to fit the DT. Notice that, while we only visualize a limited view of the feature space, the splitting rules are open-ended, meaning that the entire feature space is associated with exactly one label.

Trees as rules

Depending on the size of a DT and the semantic meaning of the features, it is fairly easy to interpret a DT. In fact, DTs can be reformulated as lists of rules, one for each path in the tree, which corresponds to one region of the feature space. Let $\mathcal{R} = \{\mathcal{R}_1, \dots, \mathcal{R}_n\}$ be a partition of the feature space into n disjunct regions. Then, we can reformulate the tree model as calculating

$$f(\boldsymbol{x}) = \sum_{\mathcal{R}_i \in \mathcal{R}} c_i \mathbb{I}[\boldsymbol{x} \in \mathcal{R}_i]$$

where c_i is the corresponding label for the region $\mathcal{R}_i$. In the classification setting, where each region corresponds to one class $c \in \mathcal{C}$, we can even combine all regions with the same class into one rule, resulting in a total of $|\mathcal{C}|$ rules. For the DT shown in Fig. 2.5 we can thus construct the rules as

$$f(\boldsymbol{x}) = \begin{cases} 0 & \text{if } (x_1 \leq 0.47 \wedge x_0 \leq 0.57) \vee (x_1 > 0.47 \wedge x_0 > 0.77 \wedge x_1 > 0.66) \\ 1 & \text{otherwise.} \end{cases}$$

Ensembling of trees

While fitting a single DT can be a good starting point (or desired to find simple rules for a given problem) they have the downside of being able to overfit fairly easily [Bre96]. Consider the case where each point in the training data is assigned its own region. This will result in zero training error but bad generalization to unseen data. This can be overcome by limiting the maximum depth or requiring that a leaf cannot contain single elements. Alternatively, it is common in practice to use ensembles of trees for better generalization performance [Bre01; Fri01].

Bagging We will now present the Random Forest (RF) model [Bre01], which uses *bagging*, or *bootstrap aggregation*, to train m trees and combine their predictions [Bre96]. Consider again some dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{x}_i \in \mathbb{R}^k$. Then, we can draw m random subsets (with replacement), notated as $\{\widetilde{\mathcal{D}}_i\}_{i \in [m]}$ with $\widetilde{\mathcal{D}}_i \sim \mathcal{D}$. Next, we can fit

Algorithm 2.1 Gradient Boosting Tree (GBT)

1: $f_0 = \arg\min_\gamma \sum_{i=1}^{n} \ell(\gamma, y_i)$ ▷ Base prediction
2: **for** $m \in [M]$ **do**
3: $\quad \boldsymbol{r}_m = \left(-\frac{\partial \ell(f_{m-1}(\boldsymbol{x}_i), y_i)}{\partial f_{m-1}(\boldsymbol{x}_i)} \right)_{i \in [n]}^{T}$ ▷ Compute residuals
4: $\quad$ Fit Decision Tree Regressor h on $(\boldsymbol{x}_i, r_{m_i})_{i \in [n]}$
5: $\quad f_m(\boldsymbol{x}) := f_{m-1}(\boldsymbol{x}) + \beta h(\boldsymbol{x})$ ▷ Add new tree to ensemble
6: **end for**
7: return f_M

one DT h per subset, resulting in $\{h_i\}_{i \in [m]}$. Instead of exhausting all possible features while training the individual trees it is common in practice to draw a subset of features at random and only finding the best split for this subset per node. Finally, to predict a new data point $\boldsymbol{x}$ all trees are evaluated and their predictions aggregated. For classification tasks, there are the options to use soft-voting (computing the average class probabilities) or hard-voting (i.e., computing the majority class) for prediction. In regression tasks the leaf values are averaged and returned. Random Forests have the advantage of few hyperparameters which need to be optimized, a high generalization ability [Bre01] and a high parallelizability since the trees can be fitted independent of each other.

Boosting Another method to ensemble models which is often used in practice is Boosting, specifically GBTs for ensembling trees [Fri01]. The idea of boosting is to fit consecutive learners on the residuals of the previous learner. The generic algorithm to compute one such ensemble for a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$, adapted from Chapter 10 of [HTF17], is given in Algorithm 2.1. Given some loss function ℓ, we start with optimizing a input-independent scalar value γ which will serve as the base prediction (Line 1). Next, for each tree in the ensemble (the number is a hyperparameter M) the negative gradient of the loss function w.r.t. the last model's prediction $f_{m-1}(\boldsymbol{x})$ is calculated (Line 3). This negative gradient is often called the *residual* of f_{m-1} Then, a DT Regressor learns to predict these residuals from data, essentially correcting for the error of the previous ensemble (Line 4). Afterwards we add the learned DT to the ensemble (Line 5), optionally with a fixed learning rate $\beta \in [0, 1]$. This process is repeated until the budget of M trees is exhausted and the final ensemble f_M is returned (Line 7).

When fitting a GBT to a regression task a common choice for ℓ is the squared error $\ell = \frac{1}{2}\ell_{\text{SE}}$. In fact, by substituting ℓ for the squared error in Line 3 of Algorithm 2.1 we can write $\boldsymbol{r}_m$ in closed-form as

$$\boldsymbol{r}_m = \left(-\frac{1}{2} \frac{\partial \ell_{\text{SE}}(\hat{y}_i, y_i)}{\partial \hat{y}_i} \right)_{i \in [n]}^{T} = \left((y_i - \hat{y}_i)^2 \right)_{i \in [n]}^{T}$$

where $\hat{y}_i := f_{m-1}(\boldsymbol{x}_i)$. Additionally, the optimal base prediction (Line 1 in Algorithm 2.1) is given as the average label, i.e., $f_0 = \frac{1}{n}\sum_{i=1}^{N} y_i$. For binary classification tasks, we can use $\ell = \ell_{\text{CE}}$ and derive the residual closed-form as

$$\begin{aligned}
\boldsymbol{r}_m &= \left(-\frac{\partial \ell_{\text{CE}}(\hat{y}_i, y_i)}{\partial \hat{y}_i}\right)^T_{i\in[n]} \\
&= \left(\frac{\partial y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)}{\partial \hat{y}_i}\right)^T_{i\in[n]} \\
&= \left(-\frac{\partial \log(1+\exp(\hat{y}_i) - y_i\hat{y}_i)}{\partial \hat{y}_i}\right)^T_{i\in[n]} \\
&= \left(\frac{\exp(\hat{y}_i)}{1+\exp(\hat{y}_i)} - y_i\right)^T_{i\in[n]}.
\end{aligned}$$

Additionally, the base prediction (Line 1) is set to a constant $f_0 = 0$ [HTF17].

2.4.3. Metrics and Evaluation

Earlier, we talked about loss functions and how they can be used to fit models to data. As an example, we saw that taking the derivative of a loss function w.r.t. a model's parameter gives us information on how to change the parameter to better fit the training data distribution. We also discussed how we can measure generalization performance of models by inspecting the loss on unseen data in Section 2.2. Next, we want to discuss how to compare multiple models over multiple datasets, which we will do extensively in the upcoming chapters. Thus, we also assume the time-series forecasting scenario throughout this section.

Assume that we have a trained forecasting model f, as well as some test data $\mathcal{D} = \{(\boldsymbol{x}_t, y_t)\}_{t\in[T]}$ of size T, where we want to predict (at time t) the value y_t from vector $\boldsymbol{x}_t$. Moreover, let $\boldsymbol{y}$ be the vector of test data labels and $\hat{\boldsymbol{y}} := (f(\boldsymbol{x_t}))_{t=1}^T$ be the prediction of f. For forecasting, we can use the (mean) squared error (Section 2.2) as a suitable metric:

$$\text{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{T}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2$$

Naturally, a prediction error of 0 would indicate a perfect fit. One downside of using the (mean) squared error when evaluating forecasters is that the unit of measurement is squared as well. Suppose that the domain $\mathcal{D}$ is the temperature of some room which we want to forecast, measured in degrees Celsius ($°C$). Then, the error is given in degrees Celsius squared, which might be unintuitive and unsuitable for applications where a minimum or maximum expected error is desired. One method to make the error more interpretable is to use the Root Mean Squared Error (RMSE), defined as

$$\text{RMSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(y_i - \hat{y}_i)^2} \tag{2.9}$$

Forecaster	$ts1$	$ts2$	$ts3$	$ts4$	Avg. Loss	Avg. Rank
f_1	0.5	0.4	0.3	0.8	0.5	**1.25**
f_2	0.6	0.5	0.4	0.1	**0.4**	1.75

Table 2.1.: Example results when comparing two forecasters f_1 and f_2 on 4 time-series. Observe that the average loss of f_2 is smaller but the average rank of f_2 is larger than that of f_2.

where the unit (due to taking the square root) is preserved. When comparing performances across time-series with different units it is common to compute the symmetric mean absolute percentage error, given by

$$\text{SMAPE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{T} \sum_{i=1}^{T} \frac{|y_i - \hat{y}_i|}{|y_i| + |\hat{y}_i|}. \tag{2.10}$$

SMAPE results in a value in $[0, 1]$, where 0 is again a perfect fit. Thus, we can compare forecasters across time-series of different datasets and domains.

Suppose now that we have fixed some performance metric and that we want to evaluate multiple forecasting models in terms of predictive performance. In time-series forecasting, we are often given a dataset consisting of multiple time-series to forecast. Thus, for each time-series we can compute one loss value per forecaster. Suppose that we have a dataset of n time-series and M forecasters. To find out which forecaster performs best, we could compute the average loss over all n time-series and compare the value between the forecasters. However, this approach has two issues: First, it is prone to outliers of individual time-series. As an illustration, consider the example in Table 2.1 of a dataset with $n = 4$ time-series and the losses for $M = 2$ forecasters. We can see that the average error of forecaster f_2 is lower, suggesting that it performed better on this dataset. However, notice that this is due to a large difference in prediction error on the last time-series out of the dataset. This leads to the average loss being smaller while f_1 outperformed f_2 on all other time-series. Second, by merely averaging losses it is hard to decide whether a difference in performance is statistically significant. While a difference between models might seem stark this can also be due to the scaling of the individual time-series.

Thus, a common approach to gain a more robust idea of model performance is to calculate rankings based on the losses. That is: If we calculated M losses, one for each forecaster, on the same time-series we can compute a ranking where the smallest value gets rank 1, the second-smallest rank 2 etc. Afterwards, to compare across n time-series we can compute the average ranking. This approach is also useful if we not only compare forecasters between time-series from the same dataset but over multiple datasets with possibly different domains (and thus, different scales). Also, we do not have to make any specific assumption about the distribution of the losses since only there relative rankings are considered. We also computed the average rank for the earlier example in Table 2.1

in the right-most column, showing that f_1 performed consistently better in comparison to f_2.

We can further compare forecasters pairwise using statistical hypothesis testing, namely rank tests, in order to assess how significant the differences in performance are. Formally, let $\boldsymbol{L}$ be a matrix of performance metrics (such as RMSE) for M forecasters on n time-series. Additionally, we define with $\boldsymbol{R}$ the matrix of rankings where each column is given by ranking the corresponding column of $\boldsymbol{L}$:

$$\boldsymbol{L} = \begin{bmatrix} l_{1,1} & \cdots & l_{1,n} \\ \vdots & \ddots & \vdots \\ l_{M,1} & \cdots & l_{M,n} \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} r_{1,1} & \cdots & r_{1,n} \\ \vdots & \ddots & \vdots \\ r_{M,1} & \cdots & r_{M,n} \end{bmatrix}.$$

Additionally, let the average rank for forecaster j be given by $r_j := \frac{1}{n}\sum_{i=1}^{n} r_{j,i}$. First, we have to test if there is enough difference between the forecasters at all. To do that, we utilize the Friedman test [Fri37]. After computing the average rankings it formulates the null hypothesis that the following statistic z_F is χ^2-distributed (with $M-1$ degrees of freedom) [BCM16]:

$$z_F = \frac{12}{nM(M-1)}\left(\sum_{j=1}^{M} r_j^2 - \frac{n(M-1)^2}{4}\right).$$

The test thus measures the difference between the assumption that all average rankings are given by $r_j = \frac{n(M-1)^2}{2}$. If the null hypothesis is rejected we can assume that there are indeed significant differences in average rank. Thus, we can now proceed to evaluate in a pair-wise manner how different the average ranks are. To do that we can use another non-parameteric rank test, namely the Wilcoxon signed-rank test [Wil45; Dem06; BCM16]. Let $i, j \in [M]$ be two models which we want to compare. For every dataset k let $d_k := l_{i,k} - l_{j,k}$ be the difference of losses between i and j. After computing the differences for all datasets we can again compute the rankings based on (absolute values) of these differences. We denote these rankings with r'_k for the ranking of $|d_k|$. Finally, the test statistic is given as

$$T = \min(R^+, R^-), \quad R^+ := \sum_{d_k>0} r'_k + \frac{1}{2}\sum_{d_k=0} r'_k, \quad R^- := \sum_{d_k<0} r'_k + \frac{1}{2}\sum_{d_k=0} r'_k.$$

For the Wilcoxon signed-rank test, the null hypothesis states that the difference in rankings is not significant. This can be accomplished by computing the following test statistic [Dem06]:

$$z_W = \frac{T - 0.25n(n+1)}{\sqrt{(1/24)n(n+1)(2n+1)}}$$

which is approximately normal distributed for large T. Thus, if we can reject the null hypothesis we can confidently say that forecasters i and j have a different performance (and also which forecaster performed better, based on the average rank).

Forecaster	Losses	Wins	Avg. Rank
`Baseline 1`	2 (2)	13(13)	3.4
`Baseline 2`	3(3)	12(10)	2.7
`Baseline 3`	4 (2)	11(7)	1.9
Our method	-	-	1.3

Table 2.2.: Example results for a comparison of four methods on 15 time-series. The Losses column indicates the number of times that the baseline method performed better (in terms of ranking of loss values) in comparison to "Our method" (analogous wins). The number significant wins/losses is given in parentheses.

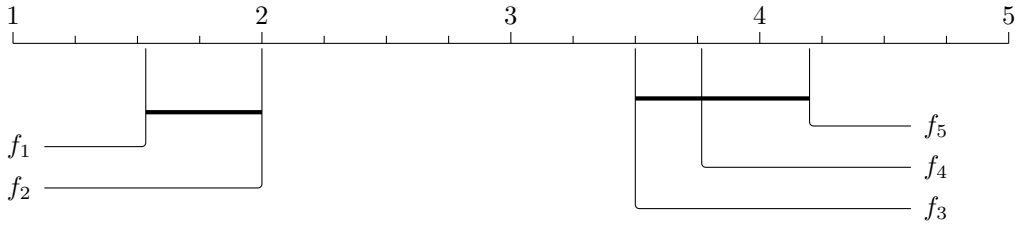

Figure 2.6.: Example CDD comparing 5 forecasters. The methods f_1 performed best, but not significantly better than f_2. Methods f_3 to f_5 also performed not significantly different from each other, but significantly worse compared to f_1 and f_2.

In order to communicate the results of the previously mentioned significance tests we will resort to two methods for the remaining chapters. First, we will present the average ranks, wins and losses for each forecaster over multiple datasets in a table. This is useful if we investigate the efficacy of one method (i.e., our presented methods) in comparison to baseline methods. Consider the table in Table 2.2 as an example. We consider a *win* of our method if the rank is lower than the comparison method. Once a win is established we will denote in parentheses if that difference is significant, i.e., if the null hypothesis of the corresponding Wilcoxon signed-rank test was rejected. Losses are reported in the same way. A different way of visualizing these results (which we will also utilize later) are so-called Critical Difference Diagrams (CDDs) [Ism+19] (see Fig. 2.6 for an example). A CDD plots each method with its average rank along a horizontal access (where further left indicates lower rank). Thus, we can get a first visual comparison between the methods by inspecting the closeness of two methods. Additionally, if we could not reject the null hypothesis of the Wilcoxon test we can denote that by a horizontal bar between the two methods. In the example, we can see that f_1 outperforms the other forecasters. However, it does not perform significantly better than f_2 due to the Wilcoxon test not rejecting the null hypothesis. CDDs are thus a powerful visualization tool when inspecting the prediction performance of multiple models on multiple datasets.

2.5. Explainable and Interpretable Machine Learning

After discussing the basics of Machine Learning and Time Series Forecasting we will proceed by formally introducing both Explainable Machine Learning (XML) and Interpretable Machine Learning (IML) in the following two subsections. These will serve as a basis to more application-focused discussions on XML in Part II as well as to applications of IML in Part III.

2.5.1. Explainable Machine Learning

In the introduction of this thesis we briefly discussed what we mean when we refer to Explainable Machine Learning. XML is way to investigate the inner workings of complex ML models. For example, we assume some sort of trained, deep forecasting model. While they tend to achieve high predictive performance, are straight-forward to implement using modern programming libraries and scale very well with large amounts of data, they are inherently unsuitable for a lot of use cases where the *why* behind decisions is crucial. These use cases are mainly those where peoples health, money and dignity are at stake. Healthcare and financial applications are often cited application scenarios where DL models perform well but are hard to operationalize due to regulatory demands [Gui+18]. If even professionals can not tell why a model predicted, for example, that a patient will likely suffer from some disease in the future, then the model cannot be used in practice. XML methods that aim to shed light into why models predict the way they do aim to bridge this gap and try to make high performance models ready for the real world.

Similar to other authors, we will discuss XML approaches along three categories [Gui+18; Mol20; LL17]. The first category that differentiates XML methods is if they aim to give a *local* or *global* explanation. Local explanations are explanations that refer to a specific model prediction. For example, they might try to answer the questions of how important a certain feature was for a particular prediction. On the other hand, global explanations try to investigate the model independent of individual predictions. They might, for example, be concerned with finding which features are never utilized by the model.

The second category is concerned with the type of model that is investigated. What are often referred to as *model-agnostic* methods do not assume anything about the model that is evaluated [RSG16; LL17]. In fact, they consider the model a *black-box*, which one can query with data and observe the output but which one cannot investigate or use further. Naturally, these XML methods are applicable to all ML models. On the other hand, there are XML methods that assume either certain properties of the model or specific model families [Lun+20; Sel+19]. These are referred to as *model-specific* XML methods. For example, they generate explanations based on gradients of the output w.r.t. the features, which means that the model has to be a fully differentiable [STY17; Sel+19]. Alternatively, they might generate their explanations in a faster way by assuming a certain structure of the model, such as depending on the model being a tree [LL17; Lun+20].

The third category is concerned with the type of explanation that a XML method generates. For the purpose of this thesis, we will discuss the most common type found in the literature, namely *attribution-based* explanations. We want to note, however, that there are numerous methods for other types of explanations proposed in the literature, such as rules generated by surrogate models [FH17] or generating counterfactuals [Ate+20; DGK20]. For more examples and an in-depth discussion we refer to relevant surveys such as [AB18; Gui+18].

Attribution-based methods are explainability methods that try to quantify how important each feature was for a specific prediction (in case of local methods) or to the model in general (in case of global methods). This information can be used to form hypotheses about the models inner workings, such as which features are never or rarely used. Formally, we define local *attributions* in the following way:

Definition 2.14 (Local attribution)**.** *Given a data point to explain $\boldsymbol{x} \in \mathcal{X}$ and some prediction function $f : \mathcal{X} \to \mathcal{Y}$, a local attribution $\boldsymbol{a}$ generated by attribution method $g : \mathcal{X} \times (\mathcal{X} \to \mathcal{Y}) \to \mathcal{A}$ is defined as*

$$\boldsymbol{a} = g(\boldsymbol{x}, f)$$

where the explanation space $\mathcal{A}$ has the same dimensionality as $\mathcal{X}$.

Semantically, the attribution a_i for feature i reflects the *importance* (whose definition varies for different attribution methods g) of feature x_i for the prediction $f(\boldsymbol{x})$. Methods of this kind are discussed under various names in the literature. In the context of computer vision they are often called *saliency maps* [Ade+18] as they highlight which part of an image is most important (or *salient*). The term *feature importance* is also often used to measure how much a decision of a model (in general or on a particular data point) is depending on a feature [Kön+21].

We will conclude by outlining a fairly simple attribution approach, namely to use the gradient of the output w.r.t. the input. This, of course, requires that the model is fully differentiable, which is the case for all neural network architectures. Thus, in the context of explaining neural network decisions *gradient-based* methods are very popular. Consider some differentiable model $f : \mathbb{R}^n \to \mathbb{R}$, together with some input $\boldsymbol{x} \in \mathbb{R}^n$. We can compute the gradient of the models output w.r.t. the input as

$$\boldsymbol{a} = g_{\text{Grad}}(\boldsymbol{x}, f) := \nabla f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_i} \right)_{i \in [n]}.$$

As mentioned in Section 2.4.1, the partial derivative of a function w.r.t. some x_i indicates the direction of the steepest ascent. If the output of f is, for example, a class probability, then a positive gradient means that increasing the value of x_i slightly will increase the output probability, giving a notion of importance to the feature x_i. While this approach is very simple and naive it serves as the basis for many attribution-based methods [STY17; LL17; Smi+17] and can be used as a simple baseline explainability method by itself. One downside to the simple approach is that the explanations tend to be fairly noisy, i.e., most

features are deemed to be somewhat relevant [Smi+17]. This means that interpreting the explanations can be fairly hard due to the high level of noise.

To summarize, we note that our used definition of XML methods for the upcoming sections specifically refers to local attribution methods, which can be both model-agnostic or model-specific. We will present model-specific approaches for both CNNs and tree-based models in Part II which we will use to determine the most salient subsequences in time-series data.

2.5.2. Interpretable Machine Learning

Interpretable Machine Learning is defined as using and developing understandable-by-design methods for the problems at hand [Rud19]. Rather than trying to explain the complex behavior of large, opaque models (as was the case in the previous subsection) the models referred to as "interpretable" are simple and / or small enough so that a practitioner can understand them. The classical example of interpretable models are rule-based methods, such as Decision Trees and other rule-learning approaches [Gui+18], as rules tend to be intuitive for humans to understand. For example, consider a trained Decision Tree for some task of depth 10. As mentioned earlier in Section 2.4.2, this tree can be interpreted as a list of rules, one for each leaf node. Since the tree has depth 10 we know that we have at most $2^{10} = 1024$ leaf nodes. In comparison to XML approaches what constitutes as "interpretable" is not clearly defined, as it depends on the individual user. Naturally, a single user is unlikely to grasp and investigate the entire tree due to its size. However, we argue that the degree of interpretability is still higher than that of a (deep) Neural Network for two reasons: First, each prediction can be understood exactly. Notice that (for our example) each data point can only travel along one path towards one leaf, i.e., there are at most 10 nodes before a prediction is reached. This makes it easy to investigate exactly how this prediction was computed in a understandable manner, which is not possible for the large matrix multiplications necessary in NNs. Second, the fact that each stage of the prediction is a thresholding of the feature values makes it possible to determine regions of equal prediction, which we can use to verify whether specific inputs will lead to correct predictions. The last point is part of what is discussed in the literature as *model verification*, which is an open problem for generic NN architectures [Liu+21]. We will discuss this aspect of interpretability for tree-based models in more details in Chapter 5.

Which models are simple enough to understand is also heavily depending on the problem domain. In terms of time-series forecasting, we claim that both simple, autoregressive AR(p) models and Exponential Smoothing (ETS) (see Section 2.3.2) are well studied and easy to interpret. This is mostly due to the fact that they are linear, as well as due to their small amount of learnable weights. Experts can easily reason about the importance of different lagged values and could even derive new models based on that investigation, for example if they find that certain lagged values are barely used for prediction.

The downside of relying solely on these simple, interpretable methods is their often perceived poor predictive performance, compared to state-of-the-art DL methods. For complex problems which rely on complex modeling strategies this is a logical limitation, especially of methods modeling a linear relationship between input and output. However, we want to note that it is highly application specific (and also a matter of how much engineering work goes into deriving these methods), often leading to comparable performance between interpretable and non-interpretable methods [Rud19]. DL methods perform well on many real-world tasks due to the fact that they effectively learn a complicated, non-linear feature extractor, followed by a linear discriminator. On the other hand, interpretable methods often require domain knowledge, in addition to high efforts engineering to achieve high predictive performance.

We will discuss in Part III approaches that combine interpretable models with opaque, black box Neural Networks. First, we present an approach to learn when an interpretable forecaster (in our case an AR(p) process) will be good enough (and when to rather resort to larger NN models) in Chapter 6. Moreover, we present an online rule-learning approach which uses the uninterpretable NN prediction as a signal for an anomaly in a predictive maintenance setting. This approach effectively combines the predictive power of NN models with the interpretability of tree-based models.

Part II.

Explainable and Adaptable Model Selection and Ensembling

3. Performance Gradient-based Saliency Maps

In the last chapter we discussed some simple methods to forecast time-series data such as ARIMA and ETS (see Section 2.3.2), both of which have parameters which are learned from data. Additionally, we discussed the basics of Machine Learning and discussed Neural Network architectures in the context of modern DL on large data sets. Unsurprisingly, DL methods have been successfully applied in time-series forecasting [Rom+13; Tai+12; LPP20; DIA17; KC19], achieving high levels of performance due to their ability to learn complex, non-linear feature representations from large amounts of data [US02]. This includes specific time-series forecasting areas such as energy forecasting [Hon+20], forecasting financial time-series [SGO20] and forecasting in particle accelerators in physics [LA23]. Recurrent Neural Network architectures such as LSTM or Convolutional Neural Networks (see Section 2.4.1) have proven to be especially effective architectures for time-series forecasting due to their inductive bias towards processing sequential data. Many improvements over these architectures have been proposed in literature, ranging from optimizations of the model structure to combining networks together into a single forecasting model [Gam17; LPP20; KC19]. However, DL methods can also suffer from the same problem as statistical methods, namely that their predictive performance is can be time-dependent. This is due to the complex and evolving nature of time-series data, which we have motivated under the name of concept drift in Definition 2.8. This makes intuitive sense from the *no free lunch theorem* [WM97] which states that no single model can be optimal for all problems. Hence, different forecasters have different expertise and a varying prediction performance [Cer+17a; Cer+19; Pri19]. While forecasters could be retrained on more recent data containing the changed concept this approach is often infeasible for DL methods as training time and resource consumption are very large [FS24]. One possible solution to overcome this problem of varying performance is to have multiple forecasters and choose between them for each new data point based on their expected performance. This framework is referred to as *Online Model Selection*. The online aspect means that the selection between models can be changed between each prediction, making the framework flexible in adapting to new concepts.

This chapter is based on the publication [SJM21]. The authors of this thesis contributed to the conceptualization of the paper idea, the general methodology of the approach and implemented most of the code. He also contributed text to the Experiments section and revised the remaining chapters.

This chapter will start with a formal introduction to Online Model Selection in Section 3.1, followed by an introduction to Regions of Competence in Section 3.2, a recent method of Online Model Selection which we will adapt for our proposed method. Specifically, we will present Performance Gradient-based Saliency Maps (PGSMs), where we modify the explainability method Grad-CAM [Sel+19; AS19] to find the most salient subsequences of a time-series based on a forecasters prediction error. We show how to compute each forecasters RoC via PGSMs in Section 3.5. We will also outline how the pool of forecasters was designed (Section 3.3), as well as our proposed solution to adapting to concept drifts by adapting the RoCs after concept drift detection (Section 3.6). Afterwards we will conduct extensive experiments in Section 3.7 where we will compare ourselves both to different variants of the OS-PGSM selector and to state-of-the-art online model selection methods. In addition, we will showcase the explainability aspects that are possible with our approach. Lastly, we will end the chapter with some concluding remarks.

3.1. Online Model Selection

Assume a given set $\mathcal{P} = \{f_1, \cdots, f_M\}$ of M *trained* forecasters. We will refer to this set as the *pool* of models available for selection. Also assume that, at some time t, the task is to forecast from the last L known values $\boldsymbol{x}_t \in \mathbb{R}^L$. Then, we aim to find the model in the pool which will result in the smallest forecasting error.

Definition 3.1 (Online Model Selection)**.** *Let $\boldsymbol{x}_t \in \mathbb{R}^L$ be the last L known values, let $y_t \in \mathbb{R}$ be the true next value and $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$ be a suitable loss function. Then, the online model selection problem is given by finding the best model f_t^* so that*

$$f_t^* = \arg\min_{f \in \mathcal{P}} \ell(f(\boldsymbol{x}_t), y_t).$$

This framework allows an adaption to concept drifts as discussed in Definition 2.8 by switching to models with better prediction performance for new concepts. Additionally, if one model in the pool consistently outperforms the others, the framework is able to chose that model every time. Thus, assuming a perfect selection, online model selection will always lead to a better or equal prediction performance compared to choosing one fixed model, making it a powerful framework in adapting to novel data characteristics.

Since the true value y_t is not known for new data the approaches for solving the minimization problem proposed in the literature can be divided into three categories. The first category of methods is based on approximating a posterior over the expected error of the different forecasters using either parametric [BM01] or non-parametric estimation methods [AGP10]. These methods are often not practical in the context of forecasting since continuous composite densities for the error function of the target and estimated time-series values have to be approximated. Thus, the results depend largely on the quality of approximation. The second category consists of using empirical estimation of the unseen error of a given model using unseen data in the form of a validation or calibration dataset.

Models with lowest estimated error are selected subsequently [RP99]. These methods are quite ineffective in practice since the estimated empirical error is usually lower than the true error. The third category is based on the meta-learning paradigm, where the selection of the adequate method is viewed as a prediction task. Another ML model called the *selector* predicts which model to choose based on meta-features and previous selections [Cer+17a; SPM19; Wol92; Jac+91]. For example, ADE [Cer+19] evaluates all forecasters from the pool on unseen validation data. Afterwards, the authors fit one regression model per forecaster (such as a Random Forest) on the absolute error values of that forecaster. Then, during inference, all regression models predict the absolute error for the new data point and the model with the lowest error is selected. In general, these methods consist of three steps:

1. **Fit base forecasters**: Select a set of base learners and fit them to some training data. While some methods allow for a diverse set of forecasters and are thus model agnostic [Cer+17b; Cer+19] others are restricted to certain model families [Jac+91].

2. **Fit the selector**: The selector mechanism (which is usually some sort of ML approach) learns (on unseen validation data) which model to chose when. For example, some approaches compute time-series statistics as features for the selector [Wol92] while others use the time-series data directly [Pri19]. As the label of the selector it is a popular approach to choose some performance metric of the individual forecaster, such as squared or absolute error [Cer+17b; Cer+19; Wol92].

3. **Select on new data**: The trained selector is used to decide which forecaster to select for a new data point. For example, the selector could predict the model directly in the form of a classification task [Wol92] or the predictions of the selector can be further processed to come to a final selection [Cer+17a].

We will focus our attention on this last category of selection methods as the flexibility of varying the selector by utilizing different learners allows us to address the two major downside of previously proposed methods. First, many proposed methods assume the selector to be static [Cer+17b; Cer+19; Pri19]. This means that they fit the selector on held-out validation data and do not update it afterwards. Consequently, the selector itself can be affected by drift, impacting the performance of the selection and consequently of the overall prediction performance. Second, the question of why a specific forecaster was chosen is often not addressed in previous approaches [Cer+17b; Cer+19]. Questions such as "Why was forecaster A chosen instead of B?" cannot be addressed adequately if the selection strategy is as intransparent as the forecasters themselves. While online model selection can improve the performance on real-world tasks it cannot be used in practice if the selection and forecasting processes cannot be trusted by the users. The only approach that allows to investigate this question of why a specific forecaster was selected was presented in [Pri19]. However, the authors do not directly address the questions of explainability in their work. Moreover, they use a static selection mechanism, making it problematic for real-world drifting data as discussed above.

3.2. Regions of Competence

Our approach of *explainable* model selection, which we will present in this chapter, extends the recently proposed work of Online Model Selection with Region of Competence (RoC) [Pri19], for which we use the following definition:

Definition 3.2 (Region of Competence [Pri19])**.** *Let $\mathcal{P} = \{f_1, \dots, f_M\}$ be set of forecasting models and let $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ be a set of windows $\boldsymbol{x}_i$ and the correct forecast y. Then, the Region of Competence of model $f_j \in \mathcal{P}$ is given as*

$$\mathcal{R}_j := \{\boldsymbol{x}_i \mid (f_j(\boldsymbol{x}_i) - y_i)^2 \leq (f_k(\boldsymbol{x}_i) - y_i)^2 \quad \forall f_k \in \mathcal{P}, i \in [N]\}.$$

While the RoCs can be visually inspected to gain some insight into the expertise of the forecasters they give no insight into which parts of the sequence were most important. During inference these RoCs are also used to determine the model to forecast for some new window $\boldsymbol{x}_t$ by returning the model with the RoC entry that is closest to $\boldsymbol{x}_t$, which is very susceptible to noise. Our approach determines the most salient subsequences for each window and adds those to the RoCs instead, which has two advantages: First, we can show that the model selection performance increases if we use salient subsequences instead of entire window for selection. Second, we are able to provide better insight into each forecasters' expertise by visualizing the fine-grained subsequences inside the RoCs. Since the subsequences are found using the prediction error they visually indicate the most important parts of the original window. In addition, we also present an approach to continuously update each forecasters RoC based on a concept drift detection mechanism which, as we will show, makes our method robust to time-variying time-series characteristics. Throughout this chapter we will refer to our method as *Online Selection with Performance Gradient-based Saliency Maps (OS-PGSM).*

3.3. Pool of forecasters

Throughout this chapter we will assume that we are given some univariate time-series $\boldsymbol{X}$ which we will partition into three parts along the time dimension. The first part, $\boldsymbol{X}^{\text{train}}$ will be used to fit the forecasters described in this section. The following part, $\boldsymbol{X}^{\text{val}}$ of validation data is used to compute each forecasters RoC. The last section, $\boldsymbol{X}^{\text{test}}$ is used to test the selection strategy during our experiment section.

As will become clear in Section 3.4 we need to use NN architectures which utilize convolutional layers for feature extraction in order to be able to compute PGSMs. The architecture of each model is thus similar, as each model consists of a sequence of 1D convolutional layers where the number of filters and layers are variable. Each convolution layer is followed by a ReLU activation and batch normalization layer (see Section 2.4.1). Some base models will have a fully connected network following the last convolution layer while the others will have a LSTM network. After parameterizing each forecaster they are trained

on windows of L past values over $\boldsymbol{X}^{\text{train}}$. We will further discuss the model pool make up in Section 3.7.

3.4. Computing PGSMs

The Performance Gradient-based Saliency Maps (PGSMs) we propose are based on Class activation maps (CAMs) [Zho+16], specifically an extension called Grad-CAM [Sel+19]. Grad-CAM has been proven to successfully pass commonly used sanity checks, which are devised to check whether the saliency map is truly providing insights into what the model is doing or not [Ade+18]. In contrast to previous approaches of applying Grad-CAM to time-series data [AS19] we are interested in explaining the performance of a given forecaster in the form of its empirical error. Let $\boldsymbol{x}_i \in \mathbb{R}^L$ be some window of a univariate time-series we want to forecast and let $y_i \in \mathbb{R}$ be the corresponding true value of the forecast. Then, assuming some trained forecaster f_j we denote the squared error of the prediction as

$$\epsilon_i^j = (f_j(\boldsymbol{x}_i) - y_i)^2.$$

Our goal is to attribute to each lagged value in $\boldsymbol{x}_i$ the respective contribution to ϵ_i^j. Since we are extending Grad-CAM we also need to restrict our forecaster f_j to be a CNN, meaning that it contains (at least one) convolutional layer for feature extraction. Grad-CAM utilizes the feature maps of the last convolution layer of the network in order to compute the explanation. Specifically, let $\boldsymbol{A}^{(u)}$ be the output of the u-th filter of the last convolutional layer for $\boldsymbol{x}_i$. We assume that there are U filters in the last convolutional layer, i.e., the set of activations is given as $\boldsymbol{A} = \{\boldsymbol{A}^{(u)}\}_{u=1}^{U}$. Moreover, we assume that the CNN architecture utilizes padding and does not contain layers that change the size of the input time-series, resulting in $\boldsymbol{A}^{(u)} \in \mathbb{R}^L$ for all $u \in [U]$. Next, assume some differentiable subnetwork computing the forecast based on the filter outputs, such as a FCN or LSTM network. We compute the empirical error ϵ_i^j with this forecast. For each filter $u \in [U]$ an importance weight $w^{(u)}$ associated with ϵ_i^j is computed by averaging over the gradients of the error w.r.t. each entry in the feature map, i.e.,

$$w^{(u)} = \frac{1}{L} \sum_{l=1}^{L} \frac{\partial \epsilon_i^j}{\partial A_l^{(u)}}.$$

Finally, we use $w^{(u)}$ to weight the contribution of each feature map to the final explanation, given by

$$\boldsymbol{a}_i^j = \text{ReLU}\left(\sum_{u \in [U]} w^{(u)} \boldsymbol{A}^{(u)} \right).$$

As we are only interested in highlighting time steps contributing most to ϵ_i^j we apply a ReLU application on this weighted combination, resulting in the final performance-based explanation.

3.5. Computing refined Regions of Competence

Next, we present our approach for finding the most salient subsequences using the PGSM information. The pseudocode for our approach is also shown in Algorithm 3.1. First, assume some set of P pretrained forecaster $\mathcal{P} = \{f_1, \ldots, f_P\}$ from which the selector will choose. Assume that we are given some validation time-series $\boldsymbol{X}^{\text{val}}$ on which we want to determine the RoCs for each forecaster. To do that, we utilize the windowing approach to create n_ω chunks of length ω:

$$(\boldsymbol{X}_i^{\text{val}})_{i=1}^{n_\omega} = W_{z,\omega}(\boldsymbol{X}^{\text{val}}). \tag{3.1}$$

For each of these chunks we will further apply windowing as defined in Definition 2.9 to split it into windows suitable to be processed by the forecasters. Specifically, we assume that the forecasters can process a window of L time-steps, i.e., we split the j-th window $\boldsymbol{X}_j^{\text{val}}$ as

$$\{(\boldsymbol{x}_i^{(j)}, y_i^{(j)})\}_{i=1}^{n_L} = W_{1,L,1}(\boldsymbol{X}_j^{\text{val}}), \quad \boldsymbol{x}_i^{(j)} \in \mathbb{R}^L, y_i^{(j)} \in \mathbb{R}. \tag{3.2}$$

Then, we determine the best performing forecaster for each chunk based the corresponding window, i.e., the best performing model for chunk j is given by solving

$$f_k^* = \underset{k \in [M]}{\arg\min} \sum_{i=1}^{M} (f_k(\boldsymbol{x}_i^{(j)}) - y_i^{(j)})^2. \tag{3.3}$$

Since forecaster f_k^* is the best performing forecaster on chunk j we will proceed by adding all windows of chunk j to its RoC. Specifically, we will compute the explanations $\boldsymbol{a}_i^{(j)}$ as shown in the previous section corresponding to the loss on each window $(\boldsymbol{x}_i^{(j)}, y_i^{(j)})$ and use these to find the most salient subsequences. To do that, normalize each attribution to be between 0 and 1 and define a threshold $\tau = 0.5$ which we will use to set all entries below τ to zero. Thus, we will only retain the highly informative parts of the window. Using the thresholded attribution we will extract all subsequences from $\boldsymbol{x}_i^{(j)}$ for which the corresponding attribution is larger than 0 and longer than 3 in order to not extract very short sequences. We visualize this approach in Fig. 3.1.

As mentioned in the beginning of the chapter the selection of the best forecaster for new data is done over the RoCs. Assume, during inference, some new window $\boldsymbol{x}_t$ at time t for which the best forecaster is to be selected. Then, we compute the distance between each forecasters RoC entries and $\boldsymbol{x}_t$ in terms of the Dynamic Time Warping (DTW) [BC94] distance function and select the forecaster with the smallest distance, i.e., Afterwards, the forecaster $\hat{f}$ with the smallest DTW distance is chosen to forecast $\boldsymbol{x}_t$:

$$\hat{f} := \underset{k \in [M]}{\arg\min} \ \min\{D_{\text{DTW}}(\boldsymbol{r}, \boldsymbol{x}_t) \mid \boldsymbol{r} \in \mathcal{R}_k\}. \tag{3.4}$$

We chose DTW because the RoC entries are of variable length and DTW is able to compare time-series of different lengths by matching the same entry in the shorter time-series to multiple entries in the longer time-series via dynamic programming.

Figure 3.1.: Illustration of how to extract the most salient subsequences from the window $\boldsymbol{x}_i^{(j)}$ (top) given the PGSM explanation $\boldsymbol{a}_i^{(j)}$ (bottom). First, notice that we use $\tau = 0.5$ to threshold the explanation, only retaining values which are highly important. Second, we find subsequences in the thresholded explanation that are longer or equal than 3 entries and extract the corresponding values from $\boldsymbol{x}_i^{(j)}$ to add to the forecasters RoC $\mathcal{R}_k$.

3.6. Detecting Concept Drifts and Adapting RoCs

As explained above, the RoCs are computed using the validation set $\boldsymbol{X}^{\text{val}}$. However, due to the dynamic behavior of time-series data it might happen that concept drifts (see Definition 2.8) appear during the inference phase. As a result, the RoCs have to be updated to take into account the possible presence of new patterns after the occurrence of such drifts and also to gain knowledge of which models are more adequate to handle these patterns if they reoccur again. Thus, we want our method to enrich the RoCs by adding new entries using newly available data whenever concept drifts are detected. The detection of concept drifts is performed by monitoring the deviation in the mean of the time-series [SPM19] using the well-known Hoeffding bound [Hoe63]. To do that, let $\mu_0 = \mathbb{E}[\boldsymbol{X}_{t_0:t_1}]$ be the initial mean over the validation data partition of $\boldsymbol{X}$, i.e., t_0 and t_1 are the first and last time-steps of $\boldsymbol{X}^{\text{val}}$. Then, for each new observation time $t > t_1$ to forecast we compute $\Delta_t = |\mu_t - \mu_0|$ with $\mu_t = \mathbb{E}[\boldsymbol{X}_{t_1:t}]$. The Hoeffding bound states that after W observations of a random variable with range r the true mean has diverged from 0 with probability σ if

$$\Delta_t > \sqrt{\frac{r^2 \log(2/\sigma)}{2W}}. \tag{3.5}$$

We estimate the range of data r empirically and set $W = t - t_1 + 1$. The confidence σ is a user-defined hyperparameter. Thus, we assume a drift whenever Δ_t diverges. All data up until $t - 1$ is assumed to be known, and thus we can use the data to enrich each forecasters RoC as described earlier in Section 3.5. Also, we set $\mu_0 = \mu_t$ and reset t_0 and t_1 to the range of μ_t.

Algorithm 3.1 OS-PGSM

1: Train each $f_k \in \mathcal{P}$ on X^{train}
2: Initialize RoCs: $\mathcal{R}_k = \{\}\ \forall j \in [P]$
3: **for** $\boldsymbol{X}_i^{\text{val}} \in W_{z,\omega}(\boldsymbol{X}^{\text{val}})$ **do** ▷ Eq. (3.1)
4: Determine the best performing f_k ▷ Eq. (3.3)
5: **for** $(\boldsymbol{x}_i^{(j)}, y_i^{(j)}) \in W_{1,L,1}(\boldsymbol{X}_i^{\text{val}})$ **do** ▷ Eq. (3.2)
6: Compute PGSM explanations $\boldsymbol{a}_i^{(j)}$ ▷ Section 3.4
7: Compute salient subsequences using $\boldsymbol{a}_i^{(j)}$ ▷ Fig. 3.1
8: Add subsequences to the corresponding RoC $\mathcal{R}_k$ ▷ Fig. 3.1
9: **end for**
10: **end for**
11: Initialize μ_0, t_0, t_1 ▷ Section 3.6
12: **for** $\boldsymbol{x}_t \in W_{1,L,1}(\boldsymbol{X}^{\text{test}})$ **do**
13: Forecast $\boldsymbol{x}_t$ with model $\hat{f}_k$ ▷ Eq. (3.4)
14: $\mu_t = \mathbb{E}[\boldsymbol{X}_{t_1:t}]$
15: **if** concept drift detected **then** ▷ Eq. (3.5)
16: Set $\boldsymbol{X}^{\text{val}} = \boldsymbol{X}_{t_1:t-1}$
17: $\mu_0 = \mu_t,\ t_0 = t_1,\ t_1 = t-1$
18: Recompute and add new RoCs ▷ Lines 3-9
19: **end if**
20: **end for**

3.7. Experiments

We present the experiments carried out to validate OS-PGSM and to answer these research questions:

- **Q1:** How does OS-PGSM perform compared to the state-of-the-art and existing online model selection methods for time-series forecasting?
- **Q2:** What is the advantage of reducing the step size z for sliding the window of size n_ω over $\boldsymbol{X}^{\text{val}}$ on the performance of OS-PGSM?
- **Q3:** What is the advantage of updating the RoCs in an informed fashion (i.e. following drift detection)?
- **Q4:** How scalable is OS-PGSM in terms of computational resources compared to the most competitive online model selection methods? What is the computational overhead of the **drift-aware** adaption of the models' RoCs?
- **Q5:** How can OS-PGSM be used to provide suitable explanations for the reason behind selecting a specific model at a certain time interval or instant?

3.7.1. Experimental Setup

As mentioned earlier we assume that each time-series is split into three parts used for training the forecasters, estimating the RoCs and evaluating the approach, respectively. Specifically, we use the first 50% for training ($\boldsymbol{X}^{\text{train}}$), the next 25% for validation ($\boldsymbol{X}^{\text{val}}$) and the last 25% for testing ($\boldsymbol{X}^{\text{test}}$). In total, we evaluate our approach and the baselines on a total of 102 real-world time-series. A full list of the used time-series, together with a description of their characteristics, is given in Table 3.1. After predicting the test portion of all time-series for all methods we compute the error via the RMSE (see Section 2.4.3). To compare the performance of the same method over multiple datasets we compute a ranking between the methods' on a per-time-series basis, assigning rank 1 if the method achieved the lowest error. By computing the average ranking over all datasets we can measure the expected performance and can compare between methods more easily. The code to reproduce all experiments is available online[4].

3.7.2. OS-PGSM Setup, Variants and Baselines

The pool $\mathcal{P}$ of CNN-based forecasting models is created by using different architecture hyperparameter settings. Specifically, we vary the number of filters used in the convolutional layers from $\{32, 64, 128\}$ and the filters' kernel size from $\{1, 3\}$. By combining these different hyperparameter configurations, as well as adding or removing LSTM layers, we create a total of $M = 12$ different forecasting models. In addition, our approach OS-PGSM has also a number hyperparameters which we summarize in Table 3.2. These hyperparameters were determined in preliminary experiments on other datasets.

We compare OS-PGSM against the following approaches which include state-of-the-art methods for forecasting and model selection methods devised in the context of forecasting. First, we compare against `ARIMA` and Exponential Smoothing (`ETS`) (see Section 2.3.2). We utilize the `forecast`[5] package which automatically estimates the (p, d, q) hyperparameters of `ARIMA` as well as the K hyperparameter from Exponential Smoothing. Next, we add the best performing forecasters from $\mathcal{P}$, denoted as `CNN` and `CNN-LSTM`, depending on whether the forecasting network is fully connected (first case) or an LSTM layer (later case). Additionally, a simple `LSTM` is added for comparison [GES01]. `KNN-RoC` [Pri19] computes static RoCs using the complete windows of the validation set as input and the rank of the individual forecasters on each interval as labels for a k-nearest neighbor classifier, using DTW distance and $K = 3$. Next, we also compare ourselves against a simple validation procedure where the CNNs are evaluated offline and the best model is selected to forecast all the upcoming data points [RP99], which we denote as `Valid`. `ADE` [Cer+17a; Cer+19] was recently developed for online dynamic ensemble of forecasters construction. However, instead of ensembling multiple models together we select the best performing model to forecast. A Random Forest is used for estimating each candidate error and select the best model based on the lowest predicted error at test time. We denote s `Stacking`

[4] https://github.com/MatthiasJakobs/os-pgsm/tree/ecml2021
[5] https://github.com/robjhyndman/forecast

Name	Nr.	Source	Characteristics
Total rents	1	[Cer+17a]	Hourly, Jan. to Mar. 2011
Temperature	1	[Cer+17a]	Hourly, Jan. to Mar. 2011
Amount registered	1	[Cer+17a]	Hourly, Jan. to Mar. 2011
AbnormalHeartbeat	1	[Bag+17]	3053 measurements (4kHz)
CatsDogs	1	[Bag+17]	14773 audio samples (16kHz)
Cricket	1	[Bag+17]	1197 accelerometer readings (184Hz)
EOGHorizontalSignal	1	[Bag+17]	1250 measurements (1kHz)
EthanolConcentration	1	[Bag+17]	1 second spectrum measurement
Mallat	1	[Bag+17]	1024 measurements (simulated)
Phoneme	1	[Bag+17]	1024 samples of audio
PigAirwayPressure	1	[Bag+17]	2000 pressure measurements
Rock	1	[Bag+17]	2844 samples of spectrum analysis
SNP500	1	[DG17; HH19]	Daily closing, 2010 to 2017
NASDAQ	1	[DG17; HH19]	Daily closing, 2010 to 2017
DJI	1	[DG17; HH19]	Daily closing, 2010 to 2017
NYSE	1	[DG17; HH19]	Daily closing, 2010 to 2017
RUSSELL	1	[DG17; HH19]	Daily closing, 2010 to 2017
Humidity RH1	1	[DG17]	10 minute steps, Jan. to May 2016
Humidity RH2	1	[DG17]	10 minute steps, Jan. to May 2016
Temperature T4	1	[DG17]	10 minute steps, Jan. to May 2016
Temperature T5	1	[DG17]	10 minute steps, Jan. to May 2016
Avg. Cloud Cover	1	[SA81]	Hourly, Apr. 25 to Aug. 25, 2016
M4 competition	80	[MSA20]	Subset of 20 hourly, monthly, weekly and daily time-series each

Table 3.1.: All used datasets for our experiments. In total, 102 datasets were used.

Parameter	Description	Value
L	Number of lagged values (Eq. (3.2))	5
ω	Size of chunks within the validation set (Eq. (3.1))	25
z	Step size for windowing (Eq. (3.1))	1
τ	Threshold for subsequence extraction (Fig. 3.1)	0.5
σ	Hoeffding-Bound confidence (Eq. (3.5))	0.95

Table 3.2.: Hyperparameters of OS-PGSM and their values for the experiments.

a method where a meta-learner (Random Forest) is trained to predict which model to select using a set of meta-features consisting of input time sequence statistical characteristics and performance-based features [Wol92]. Finally, we compare ourselves against `Adaptive Mixture` [Jac+91], which consists of shallow CNNs and a gating network. The gating network acts as a selector by performing a single output stochastic switch to select a given expert with the estimated switch probability.

We also evaluate different variants of OS-PGSM in order to evaluate the importance of each part of our method.

- OS-PGSM: The version of our method presented in the previous sections, which detects concept drifts and adapts the RoCs accordingly.
- OS-PGSM-Int: Same as OS-PGSM, but the step size in Eq. (3.1) is set to $z = \omega$. That way, the RoCs will be more sparse and the RoC update after a concept drift requires less computation.
- OS-PGSM-Euc: Instead of using DTW as a distance measure to select the forecaster we use Euclidean distance. To account for time-series of different lengths the subsequences are padded (and those padding entries are not taken into account in determining the distance).
- OS-PGSM-Int-Euc: A combination of OS-PGSM-Int and OS-PGSM-Euc.
- OS-PGSM-St: Same as OS-PGSM, but the RoCs are not updated using the drift detection mechanism. The RoCs are computed and stored offline and only the selection takes place online.
- OS-PGSM-Per: Same as OS-PGSM, but the RoCs are update periodically in a blind manner (i.e. without detecting the occurrence of concept drifts) with a periodicity of 10% of the length of $\boldsymbol{X}^{\text{test}}$.

3.7.3. Results

Table 3.3 presents the average ranks and their standard deviation for all methods. For the paired comparison, we compare our method OS-PGSM against each of the other methods. Specifically, we counted the number of wins and losses for each time-series using the RMSE error. To see if the pairwise difference between methods is significantly high we use the non-parametric Wilcoxon Signed Rank test to compute significant wins and losses, which are presented in parentheses (assuming a significance level $\alpha = 0.05$) (see Eq. (5.1) for more information).

As we can see in Table 3.3, OS-PGSM outperforms the baseline methods in terms of wins and loses in pairwise comparison. The online model selection approaches, e.g., `KNN-RoC`, `ADE-Single` and `Adaptive Mixture`, show inferior performance in comparison to OS-PGSM. `ARIMA`, `ETS`, `LSTM`, and `CNN` are also considerably worse in average rank compared to OS-PGSM. `CNN-LSTM` shows slightly better performance, but still cannot outperform

Method name	Losses	Wins	Average Rank	±
ETS	5(4)	97(96)	13.04	4.26
ARIMA	7(6)	95(93)	12.90	4.35
Stacking	11(10)	91(80)	12.11	4.12
Adaptive Mixture	17(8)	85(84)	10.93	5.78
LSTM	17(8)	85(83)	9.97	3.60
CNN	18(6)	84(80)	9.00	3.60
Valid.	20(12)	82(72)	7.84	4.18
KNN-RoC	22(16)	80(73)	7.41	3.95
CNN-LSTM	23(6)	79(71)	7.30	3.59
ADE-Single	30(19)	72(61)	4.28	2.90
OS-PGSM-Int-Euc	33(9)	69(66)	3.95	3.13
OS-PGSM-Euc	35(10)	66(56)	3.86	3.10
OS-PGSM-Int	40(7)	62(54)	3.62	2.80
OS-PGSM-Per	45(7)	57(46)	3.09	3.05
OS-PGSM-St	49(10)	53(44)	2.93	2.62
OS-PGSM	-	-	**2.78**	2.63

Table 3.3.: Comparison of OS-PGSM and its variants to different state-of-the-art approaches for all 102 time-series. The two right most columns give the average rank in terms of mean squared error and its standard deviation across different time-series. A rank of 1 means the model was the best performing on all time-series.

OS-PGSM. The most competitive state-of-the-art approach to OS-PGSM is ADE-Single. Nevertheless, it has a higher average rank and a lower performance than all the variants of our method. These results address the research question **Q1**.

Comparing OS-PGSM to different variants itself, we see a clear advantage in using all the choices in our method. First, Dynamic Time Warping (DTW) is better in computing the similarities between the input sequences and the RoCs, especially when both have different lengths as explained above. This explains why the variants using Euclidean distance have worse performance. In addition, by setting $z = 1$, higher number of windows of size ω are created and, as a result, a higher number of RoCs are computed (see Section 3.5). This contributes to creating richer information about RoCs of different forecasters, compared to setting $z = \omega$ in OS-PGSM-Int and OS-PGSM-Int-Euc. The downside is the increased processing time to create these RoCs (as we will see next). Finally, OS-PGSM-St is better than OS-PGSM-Per, which shows that unnecessary updates of the RoCs can be more detrimental than not adapting to changes at all. On the other hand, the informed, concept drift detection based approach of OS-PGSM performs better than both OS-PGSM-Per and OS-PGSM-St. This can be explained by the fact that the update of the RoCs is only beneficial for datasets where concept drifts actually occur. Moreover, taking into account these new appearing concepts is helpful for the selection of models due to newly observed patterns. Figure 3.2 show an illustrative example of the RoCs of f_4 before and after drift

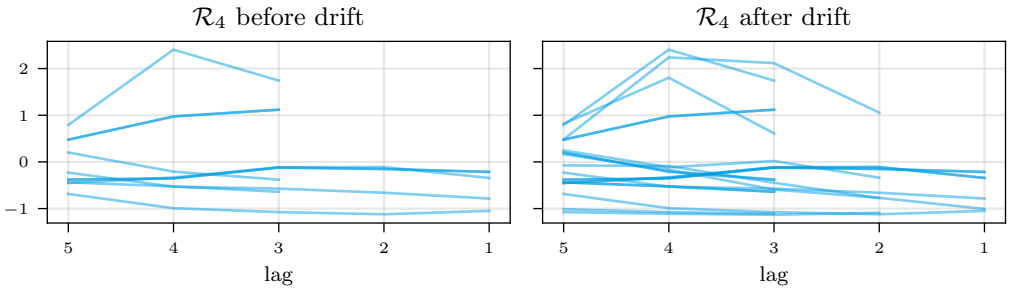

Figure 3.2.: RoCs for the forecaster model f_4 before and after enrichment, i.e., before and after a detected concept drift.

Method	ADE-Single	OS-PGSM	OS-PGSM-St	OS-PGSM-Int	OS-PGSM-Per
Avg. Runtime	167.87	8.42	2.33	0.90	154.11
±	56.40	18.30	4.80	1.65	204.22

Table 3.4.: Empirical runtime comparison between different methods (in seconds).

detection. Notice the additional patterns added which make the two broad categories of RoC entries more diverse. These answers research questions **Q2** and **Q3**.

In the next experiment, we compare the runtime of OS-PGSM and its variants against the most competitive state-of-the-art method, `ADE-Single`, in Table 3.4. The reported runtime for OS-PGSM and OS-PGSM-Per takes into account the computation of the new RoCs. `ADE-Single` [Cer+19] relies on periodic update of the meta-learning strategy behind the selection (same periodicity as OS-PGSM-Per). All the reported runtimes concern only the online predictions and any operation computed offline is not taken into account. The results demonstrate that OS-PGSM and its variants have lower average runtime than `ADE-Single`. OS-PGSM-Int is faster than OS-PGSM-St since fewer evaluation windows of size ω are created and as a result, a lower number of RoCs is generated for distance comparisons. OS-PGSM has lower runtime than OS-PGSM-Per. This is due to using drift detection to update the RoCs only when necessary, leading to fewer updates. This results in faster predictions and less computational overhead of generating PGSMs. The high deviation of the runtime of OS-PGSM is due to the different numbers of drifts per time-series. While some time-series have high variations of the mean over time, and thus will result in a lot of drifts being detected, other time-series are much more static. Due to our approach of detecting these changes we can achieve a fast runtime for more static time-series without suffering prediction performance penalties. This answers question **Q4**.

Lastly, we provide some insights into how OS-PGSM can be used to provide suitable explanations for the reason behind model selection. Figure 3.3 shows a comparison between the current input time-series window (left part in black) with the closest RoC entry of the

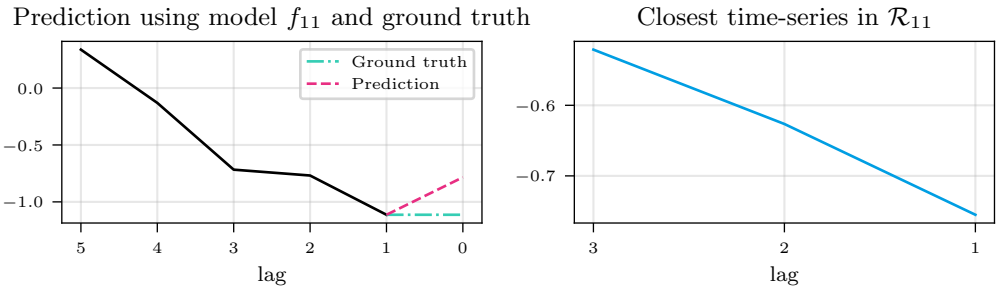

Figure 3.3.: Comparison of the current input pattern to the closest RoC ($\mathcal{R}_{11}$).

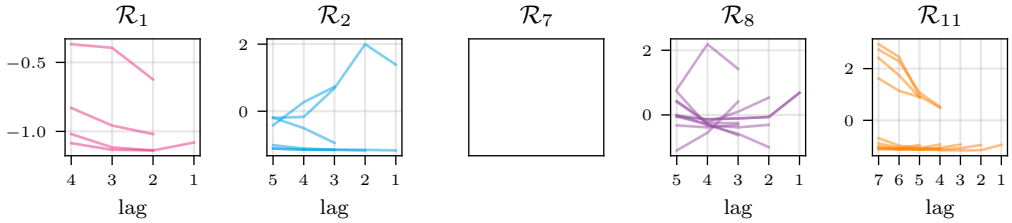

Figure 3.4.: Visualization of RoCs for the Total Rents data using OS-PGSM. Notice that $\mathcal{R}_7$ is empty, meaning that f_7 was never the best model during the validation phase.

selected model to perform the forecast. A clear similarity between both patterns can be observed which justifies the choice of this model since it has been proven to show some degree of competence in forecasting using similar patterns as input. While the amount of lags differ, i.e., the time-series windows are of different lengths, the use of DTW is able to match the shape of both time-series. The selection of, in this case f_{11}, is further validated when comparing between the true time-series value (ground truth, green) and the predicted value (red). While these two values differ slightly, an evaluation of all the candidates showed that f_{11} had, in fact, the smallest prediction error. A more general overview over the RoCs for the Total Rents dataset is shown in Fig. 3.4. Observe that some models have quite similar RoC patterns, while others are fairly distinct. For example, forecaster f_1 was proficient at slightly decreasing linear trends, as well as those trends followed by a slight uptick (see $\mathcal{R}_1$ in Fig. 3.4). On the other hand, forecaster f_{11} was best at predicting from sharply decreasing trends (see $\mathcal{R}_{11}$ in Fig. 3.4). Notice that f_7 has an empty RoC. This means that during the RoC creating process f_7 never outperformed any other model and thus did not get to enrich its RoC. At this point, during the selection process, f_7 is effectively ignored. This also leads to better explainability in the sense of sparseness, since not every model is forced to contribute to the forecasting. Hence, models that are poorly designed or not well-trained, get ignored during the selection. However, note that after a concept drift detection all forecasters from the pool are again evaluated on the newly acquired data. At this point, f_7 could be outperforming other models and enrich its RoC. As it can be seen with the varying amount of transparency of lines of the RoCs, many

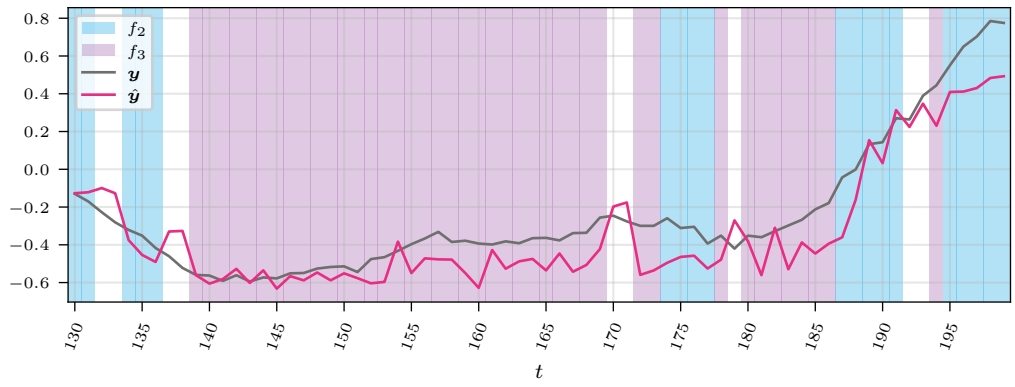

Figure 3.5.: A visualization of model selection one the AbnormalHeartbeat dataset.

very similar RoCs are collected for each forecaster, confirming the assumption that a certain model is an expert on specific input regions of the time-series. Another visualization aspect of the forecasting is shown in Figure 3.5. We focus for visualization clarity on the two models (f_2 and f_3). We highlight regions where they are selected by OS-PGSM. Notice that the parts of the time-series that exhibit mostly stationary behavior or small increasing trends are predicted by f_3. Conversely, strongly increasing or decreasing patterns are predicted by f_2. This shows that models can be associated with interpretable features of the time-series, such as trend changes, making the process of understanding each models expertise easier. All the shown aspects address clearly research question **Q5**.

3.8. Concluding Remarks

In this chapter we have introduced OS-PGSM, which is an explainable Online Model Selection method for time-series forecasting. Specifically, it uses gradient-based saliency maps to derive RoCs which can be inspected to understand the expertise of each forecaster. Using a simple drift detection mechanism the RoCs are updated whenever new concepts occur. This allows for an adaption to novel concepts, leading to increased predictive performance compared to other state-of-the-art model selection methods. We could also show that OS-PGSM s can be used successfully for providing useful explanations behind model selection, which provides a way to use CNNs in more safety-critical application domains for forecasting.

4. Online Ensemble Pruning using PGSM

We have mentioned in Section 3.1 that one way to react to the time-varying characteristics inherent in time-series data is to approach the forecasting as an Online Model Selection task. We proposed a method in the previous chapter which used the concept of Regions of Competence (RoCs) to select, for each new input pattern, which model to choose. In this chapter, we will discuss a second method to deal with concept drifts in an adaptive manner. Specifically, we will generalize the notion of Online Model Selection to *Online Ensemble Pruning*. That is: Instead of choosing one forecaster to predict the pattern we aim to select multiple forecasters and combining their predictions to achieve a high prediction accuracy. Our focus is on finding the smallest possible subset of forecasters necessary for high performance in order to not suffer from high computational costs. Formally, we will define Online Ensemble pruning over the pool of forecasters $\mathcal{P} = \{f_1, \ldots, f_N\}$ in the following way:

Definition 4.1 (Online Ensemble Pruning)**.** *Let $\boldsymbol{x}_t \in \mathbb{R}^L$ be the last L known values, let $y_t \in \mathbb{R}$ be the true next value and let $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$ be a suitable loss function. Then, the online ensemble pruning problem is given by*

$$\mathcal{G}^* = \underset{\mathcal{G} \subseteq \mathcal{P}}{\arg\min}\, \ell(\bar{f}_{\mathcal{G}}(\boldsymbol{x}_t), y_t)$$

where $\bar{f}_{\mathcal{G}}(\boldsymbol{x}_t) = \frac{1}{G} \sum_{j=1}^{G} f_j(\boldsymbol{x}_t)$ and $|\mathcal{G}| = G$.

Notice that we defined the ensemble as equally weighted, i.e., each forecasters weight in the convex combination is equal. Using non-uniform weights it is possible to include more of the prediction of certain forecasters or reduce their contribution. By constraining the weights to be zero everywhere except for one entry we can derive the model selection problem from the previous chapter. This means that online ensemble pruning can be seen as a generalization of online model selection by constraining $G = 1$.

Ensemble pruning is a popular method to overcome the high computational costs of traditional ensemble learning techniques and to improve the ensemble predictive performance. An ensemble with a very large number of models may add a lot of computational overhead due to the large memory requirements of some of its base models [MD97]. The optimization of run-time overhead is imperative for certain applications where real-time requirements have to be met, such as in online forecasting. Furthermore, both theoretical and empirical studies have shown that ensemble predictive performance depends on the predictive performance of its individual models and how diverse the ensemble is [LYZ12; BWT05; MHS08; Car+04]. All the previous works agree that encouraging diversity is

beneficial to the performance of ensemble, but it is hard to tell the theoretical properties of diversity in ensemble [LYZ12]. Therefore, understanding and promoting ensemble diversity remains an important research question in ensemble learning. The generalization performance of an ensemble depends on its empirical error and diversity [LYZ12]. Thus, it seems reasonable to design an ensemble pruning strategy with both the empirical error and diversity in mind. However, finding the optimal subset is a combinatorial search problem with exponential computational complexity. It is thus infeasible to compute the exact solution by exhaustive search and approximations are necessary. Several methods have been proposed in the literature to solve this issue by directly using global search [ZWT02; Zha02; CTY09] or iteratively using greedy search [MHS08; PTV12]. Greedy search methods can be further divided into greedy *forward* pruning which start with an empty set and iteratively add the learners optimizing a certain criterion, and greedy *backward* pruning which starts with the complete ensemble and iteratively eliminates learners. It has been shown that greedy pruning methods are able to achieve comparative performance and robustness with global search methods but at much smaller computational costs [LYZ12]. A comprehensive review of state-of-the-art ensemble pruning methods can be found in [TPV09].

In the time-series domain, pruning methods need to be dynamic in order to cope with the time-evolving nature of time-series in the form of concept drift as mentioned before (see Section 2.3.1). Dynamic methods can adapt the pruning strategy either in a blind manner over time, i.e. at each time instant during inference, periodically [KK15; ZDY21], or in an informed fashion following concept-drift detection [SPM19]. A drift-aware ensemble pruning for time-series forecasting using correlation-based measures for similarity and dynamic clustering of the top-similar models has been proposed in [SPM19]. In addition, several methods for making the ensemble weights dynamic in forecasting have been suggested in the literature [Cer+19; STM21; GG16; Cer+17a; SM21]. However, not all of them guarantee that some weights will be set to zero [STM21; GG16; Cer+17a; SM21]. In [Cer+19], the authors frame their aggregation as a ranking task, in which the models are ranked decreasing by their weight. Correlation among the output of the base learners is used to quantify their redundancy. A given learner is penalized for its correlation to each learner already ranked. If it is fully correlated with other learners already ranked, its weight becomes zero. On the other hand, if it is completely uncorrelated with its ranked peers, it gets ranked with its original weight. The pruned ensemble can then be generated by choosing a fixed number of methods based on the final weights. However, this number has to be a hyperparameter, making the approach inflexible as the amount of possible pruning also varies over time.

In the following sections we will present our approach for online ensemble pruning which can adapt to concept drifts. Specifically, we will use the PGSMs from the previous chapter to find the most salient subsequences to build each forecasters Region of Competence (RoC). During inference, we conduct a two-step approach to find the smallest sub-ensemble while minimizing both the expected prediction error as well as maximizing the diversity. The size of the sub-ensemble is derived via a bound on the distance of RoC members, leading to a dynamic ensemble size which can adapt to changes in the time-

series characteristics. Using both the concept drift detection approach from Section 3.6 and monitoring the ensembles predictive performance over time we are able to update the RoCs and the pruned ensemble for high flexibility. By utilizing the PGSMs we are also gaining insight into the ensembling process and can generate explanations of why models were ensembled together.

This chapter is based on the publication [SJM22]. The original paper is an equal-contribution publication shared with the first author and the author of this thesis. Specifically, the author of this thesis contributed to the derivation of the bound in Proposition 4.1, implemented most of the code for the experiments and contributed text to the Methodology and Experiments sections of the original publication.

The chapter is structured as follows. We will start discussing the changes to the PGSM computation presented earlier in Section 4.1. Next, we present our two-stage online ensemble pruning method in Section 4.2, where we discuss the advantage of clustering the RoCs members as well as derive a bound which balances the ensembles predictive performance and ambiguity. Moreover, we present an additional way to detect concept drifts by monitoring the ensembles predictive performance in Section 4.3. We will then conduct extensive experiments in Section 4.4 to investigate different research questions, including a discussion on different explainability aspects. Finally, we will finish the chapter with some concluding remarks.

4.1. Adapting PGSMs to ensembling

In order to use the approach of computing PGSMs from Section 3.4 we resort to using the same kind of model pool as described in Section 3.3. Namely, the pool needs to contain of CNNs, specifically containing 1D convolutional layers which are used to extract features from the time-series windows. Different architectures are again constructed by varying the number of filters and number of layers, as we will detail in Section 4.4. In the previous chapter we measured the performance of each forecaster on chunks of validation data and let the best model create new entries in its RoC. In this chapter, the performance (i.e. error) of all forecasters are measured and each forecaster gets to enrich its RoC. After thresholding and extracting the subsequences using the PGSM explanation as mentioned in Section 3.5, we discard every sequence which is of a different size to the number of lags L. Doing that we are able to utilize the euclidean distance for computing distances to the RoC entries since all entries have equal length. In preliminary experiments this approach lead to higher predictive performance compared to ensembling with DTW.

4.2. Online Ensemble Pruning

Assume that we want to forecast the true value X_{t+h} given the last L known (i.e., lagged) values $\boldsymbol{x}_t = (X_{t-L}, \ldots, X_t)$ which we will also refer to as the current *input pattern*. We

will refer to the i-th entry of $\boldsymbol{x}_t$ as $x_t^{(i)}$ throughout this chapter. At test time, in order to forecast some pattern $\boldsymbol{x}_t$ the similarity to all RoC entries of all forecasters is computed. Thus, for each forecaster f_j with $j \in [N]$ the closest RoC entry $\boldsymbol{r}_j^*$ satisfies

$$\hat{\boldsymbol{r}}_j = \underset{\boldsymbol{r}_j \in \mathcal{R}_j}{\arg\min} \|\boldsymbol{r}_j - \boldsymbol{x}_t\|_2 \tag{4.1}$$

and is selected to represent f_j for the following online ensemble pruning approach.

The pruning decision is performed in a step-wise online manner for each new input pattern. The expected error of the ensemble $\mathcal{P}$, denoted by $e_{\mathcal{P}}$, at time $t+h$ can be expressed as [KV95; BWT05]:

$$\begin{aligned} e_{\mathcal{P}}(X_{t+h}) &= (X_{t+h} - \bar{f}_{\mathcal{P}}(\boldsymbol{x}_t))^2 \\ &= (X_{t+h} - \frac{1}{N}\sum_{j=1}^{N} f_j(\boldsymbol{x}_t))^2 \\ &= (\frac{1}{N}\sum_{j=1}^{N} X_{t+h} - f_j(\boldsymbol{x}_t))^2 \\ &= \underbrace{\frac{1}{N}\sum_{j=1}^{N}(X_{t+h} - f_j(\boldsymbol{x}_t))^2}_{=:\bar{e}_{\mathcal{P}}(X_{t+h})} - \underbrace{\frac{1}{N}\sum_{j=1}^{N}(f_j(\boldsymbol{x}_t) - \bar{f}_{\mathcal{P}}(\boldsymbol{x}_t))^2}_{=:\bar{a}_{\mathcal{P}}(X_{t+h})}. \end{aligned} \tag{4.2}$$

The left term in Equation 4.2 refers to the weighted average error of the base models $\bar{e}_{\mathcal{P}}$ and the right term to the ensemble ambiguity $\bar{a}_{\mathcal{P}}$ which is simply the variance of the ensemble around the weighted mean and measures the disagreement between networks on X_{t+h}. The above relations can be averaged over several time steps H and the ensemble generalization error can be written as:

$$\begin{aligned} E_{\mathcal{P}} &= \frac{1}{H}\sum_{h=1}^{H} e_{\mathcal{P}}(X_{t+h}) \\ &= \underbrace{\frac{1}{H}\sum_{h=1}^{H} \bar{e}_{\mathcal{P}}(X_{t+h})}_{=:\bar{E}_{\mathcal{P}}} - \underbrace{\frac{1}{H}\sum_{h=1}^{H} \bar{a}_{\mathcal{P}}(X_{t+h})}_{=:\bar{A}_{\mathcal{P}}}. \end{aligned} \tag{4.3}$$

The RoCs $\mathcal{R}_j$ of each base model $j \in [N]$ can be used to calculate the degree of expertise of the corresponding model in forecasting the most recent input pattern $\boldsymbol{x}_t$ by computing the minimum distance to all RoC entries (see Eq. (4.1)). Thus, we implicitly assume that

the distance between the closest RoC entry and the current pattern is indicative of the expected error, i.e., we assume

$$(f_j(\boldsymbol{x}_t) - X_{t+h})^2 \propto \frac{1}{L}\sum_{l=1}^{L}(x_t^{(l)} - \hat{r}_j^{(l)})^2, \quad \forall j \in [N]$$

where $\hat{\boldsymbol{r}}_j = (\hat{r}_j^{(1)}, \ldots, \hat{r}_j^{(L)})^T \in \mathcal{R}_j$ is the closest entry according to Eq. (4.1). By choosing the closest entry for all forecasters and ensembling the predictions together we will also assume that the average closest RoC member is a good proxy for prediction performance of the ensemble, i.e.,

$$(\bar{f}(\boldsymbol{x}_t) - X_{t+h})^2 \propto \frac{1}{L}\sum_{l=1}^{L}(x_t^{(l)} - \bar{r}_{\mathcal{P}}^{(l)})^2$$

where $\bar{\boldsymbol{r}}_{\mathcal{P}}^{(l)} := 1/N \sum_{j=1}^{N} \hat{r}_j^{(l)} \;\; \forall l \in [L], |\mathcal{P}| = N$. With these assumptions, we will decompose the expected ensemble error for X_{t+h} using the set of all RoCs $\mathcal{R}$ the same as before:

$$\begin{aligned}\mathfrak{E}_{\mathcal{P}}^{\mathcal{R}} &= \frac{1}{L}\sum_{l=1}^{L}(x_t^{(l)} - \bar{r}_{\mathcal{P}}^{(l)})^2 \\ &= \underbrace{\frac{1}{NL}\sum_{l=1}^{L}\sum_{j=1}^{N}(x_t^{(l)} - \hat{r}_j^{(l)})^2}_{=:\bar{\mathfrak{E}}_{\mathcal{P}}^{\mathcal{R}}} - \underbrace{\frac{1}{NL}\sum_{l=1}^{L}\sum_{j=1}^{N}(\bar{r}_{\mathcal{P}}^{(l)} - \hat{r}_j^{(l)})^2}_{=:\bar{\mathfrak{A}}_{\mathcal{P}}^{\mathcal{R}}}.\end{aligned} \tag{4.4}$$

It is very intuitive to see from Equation 4.4 that the selection of the ensemble members should be made in favor of forecasters whose RoCs are close to the current pattern in order to minimize $\bar{\mathfrak{E}}_{\mathcal{P}}^{\mathcal{R}}$ and diverse from each other in order to maximize $\bar{\mathfrak{A}}_{\mathcal{P}}^{\mathcal{R}}$. To do so, we perform a two-staged pruning strategy, additionally illustrated in Fig. 4.1.

4.2.1. First pruning stage

In a first stage, we start by clustering the closest RoC entries $\hat{\boldsymbol{r}}_j$ of each forecaster using k-mean clustering [Llo82] (see also Section 2.2) with Euclidean distance into k clusters. Let $\mathcal{S} := \{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ be the resulting set of clusters. For each cluster $\mathcal{S}_j$ we define the *cluster representative* $\boldsymbol{\rho}_j$ as the element with the smallest distance to the current input, i.e.,

$$\boldsymbol{\rho}_j = \underset{\boldsymbol{\rho} \in \mathcal{S}_j}{\arg\min} \|\boldsymbol{\rho} - \boldsymbol{x}_t\|_2. \tag{4.5}$$

For convenience, let us define $\mathcal{K} := \{\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_k\}$. The optimization objective of k-means clustering minimizes the intra-cluster distances while maximizing the inter-cluster distances. That means that models belonging to different clusters are expected to have higher

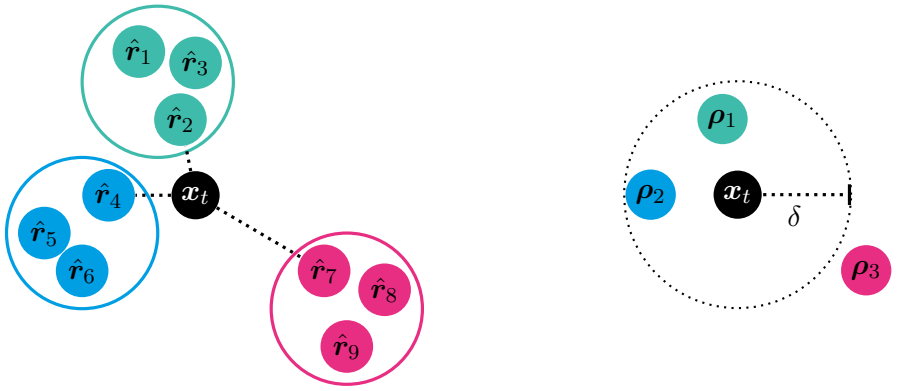

Figure 4.1.: A visualization of our two-staged online ensemble pruning approach. In the first stage (left), the closest RoC entries $\hat{\boldsymbol{r}}$ of all 9 models are clustered into $k = 3$ clusters, visualized by the three colors. Next, we find the closest cluster entry w.r.t. the current pattern $\boldsymbol{x}_t$ and call it the *cluster representative* $\boldsymbol{\rho}$. In this example, we have $\boldsymbol{\rho}_1 = \hat{\boldsymbol{r}}_2$, $\boldsymbol{\rho}_2 = \hat{\boldsymbol{r}}_4$ and $\boldsymbol{\rho}_3 = \hat{\boldsymbol{r}}_7$. In the second stage (right) we calculate the radius δ based on our proposed bound from Proposition 4.1, which results in the final ensemble $\mathcal{M} = \{f_2, f_4\}$ corresponding to $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_2$, respectively.

distances between each other compared to models belonging to the same cluster. In addition, models belonging to the same cluster have more or less the same distance to the current pattern $\boldsymbol{x}_t$. Therefore, the average error induced by all the models is expected to be similar to the averaged error induced by the clusters' representatives since one representative shows the same error level as all the models belonging to that same cluster. As a result, $\bar{\mathfrak{E}}^{\mathcal{R}}_{\mathcal{K}}$ is approximately preserved while decreasing the number of models. In this way, we achieve high diversity without increasing the averaged error, reducing the expected total ensemble error $\mathfrak{E}^{\mathcal{R}}_{\mathcal{K}}$ on $\boldsymbol{x}_t$.

4.2.2. Second pruning stage

In a second stage, we want to further reduce the number of models in the ensemble by using only the best performing cluster representatives. To do that, we derive a bound of a radius around $\boldsymbol{x}_t$ which, by including all models in that radius, gives us a pruned ensemble with a trade-off between expected error and diversity.

Proposition 4.1. Let $\boldsymbol{x}_t \in \mathbb{R}^L$ be the current input pattern and let $\mathcal{K} = \{\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_k\}$ be the k cluster representatives left from the first pruning stage. Then, we can derive a subset $\mathcal{M} \subseteq \mathcal{K}$ which improves upon the expected error and diversity as

$$\mathcal{M} = \{\boldsymbol{\rho}_j \in \mathcal{K} \mid \|\boldsymbol{\rho}_j - \boldsymbol{x}_t\|_2 \leq \delta\}$$

for the radius δ given via

$$\frac{1}{2}\sqrt{\frac{\sum_{l=1}^{L}\sum_{j=1}^{k}(\bar{r}_{\mathcal{K}}^{(l)}-\hat{r}_j^{(l)})^2}{k}} \leq \delta \leq \sqrt{\frac{\sum_{l=1}^{L}\sum_{j=1}^{k}(x_t^{(l)}-\hat{r}_j^{(l)})^2}{k}}.$$

Proof. To ensure a reduction of the averaged error, the error $\bar{\mathfrak{E}}_{\mathcal{M}}^{\mathcal{R}}$ induced by the subset of the $\mathcal{M}$ models should be lower than $\bar{\mathfrak{E}}_{\mathcal{K}}^{\mathcal{R}}$ induced by the subset of k cluster representatives. Let $M := |\mathcal{M}|$ and $k := |\mathcal{K}|$. Then, we have:

$$\begin{aligned}
&\bar{\mathfrak{E}}_{\mathcal{M}}^{\mathcal{R}} \overset{!}{\leq} \bar{\mathfrak{E}}_{\mathcal{K}}^{\mathcal{R}} \\
\Leftrightarrow \frac{1}{ML}\sum_{l=1}^{L}\sum_{j=1}^{M}(x_t^{(l)}-\hat{r}_j^{(l)})^2 &\overset{!}{\leq} \frac{1}{kL}\sum_{l=1}^{L}\sum_{j=1}^{k}(x_t^{(l)}-\hat{r}_j^{(l)})^2.
\end{aligned}$$

We have that $\sum_{l=1}^{L}(x_t^{(l)}-\hat{r}_j^{(l)})^2 \leq \delta^2 \quad \forall j \in [M]$ as each forecaster f_j should have at most distance δ to $\boldsymbol{x}_t$. As a result:

$$\begin{aligned}
\frac{\delta^2}{L} &\leq \frac{1}{kL}\sum_{l=1}^{L}\sum_{j=1}^{k}(x_t^{(l)}-\hat{r}_j^{(l)})^2 \\
\Leftrightarrow \delta^2 &\leq \frac{1}{k}\sum_{l=1}^{L}\sum_{j=1}^{k}(x_t^{(l)}-\hat{r}_j^{(l)})^2 \\
\Leftrightarrow \delta &\leq \sqrt{\frac{\sum_{l=1}^{L}\sum_{j=1}^{k}(x_t^{(l)}-\hat{r}_j^{(l)})^2}{k}}.
\end{aligned}$$

In addition, we want to ensure that the second stage either preserves or promotes diversity by preserving or increasing the ambiguity:

$$\begin{aligned}
&\bar{\mathfrak{A}}_{\mathcal{M}}^{\mathcal{R}} \overset{!}{\geq} \bar{\mathfrak{A}}_{\mathcal{K}}^{\mathcal{R}} \\
\Leftrightarrow \frac{1}{ML}\sum_{l=1}^{L}\sum_{j=1}^{M}(\bar{r}_{\mathcal{M}}^{(l)}-\hat{r}_j^{(l)})^2 &\overset{!}{\geq} \frac{1}{kL}\sum_{l=1}^{L}\sum_{j=1}^{k}(\bar{r}_{\mathcal{K}}^{(l)}-\hat{r}_j^{(l)})^2.
\end{aligned}$$

The average pattern $\bar{\boldsymbol{r}}_{\mathcal{M}}$ has to be contained within the sphere induced by radius δ of center $\boldsymbol{x}_t$ as it is a convex combination of vectors inside the radius by definition. Thus, the distance between $\hat{\boldsymbol{r}}_j$ and $\bar{\boldsymbol{r}}_{\mathcal{M}}$ can be at most 2δ for all $j \in [M]$. Therefore:

$$\begin{aligned}
\frac{4\delta^2}{L} &\geq \frac{1}{L}\sum_{l=1}^{kL}\sum_{j=1}^{k}(\bar{r}_{\mathcal{K}}^{(l)}-\hat{r}_j^{(l)})^2 \\
\Leftrightarrow \delta &\geq \frac{1}{2}\sqrt{\frac{1}{k}\sum_{l=1}^{L}\sum_{j=1}^{k}(\bar{r}_{\mathcal{K}}^{(l)}-\hat{r}_j^{(l)})^2}.
\end{aligned}$$

□

In practice, in order to identify the $\mathcal{M}$ models for a fixed number of clusters k, we set δ simply to the upper bound of Proposition 4.1 and models whose distance is lower or equal to δ are selected as the $\mathcal{M}$ models. The upper bound of Proposition 4.1 can then be interpreted as promoting the selection inside the δ-radius sphere close to the true recent pattern $\boldsymbol{x}_t$ which reduces the averaged error. The lower bound can be viewed as a regularization parameter that prohibits radius values that would result in a clustering of the base models very closely around $\boldsymbol{x}_t$, decreasing the ambiguity. After $\mathcal{M}$ is derived at time t it remains valid for the following time instants unless a concept-drift either in the models' performance or in the time-series data is detected. If a drift is detected, an alarm to update $\mathcal{M}$ is triggered, which we will discuss in the upcoming section.

4.3. Drift-aware Pruning Update

In order to update the pruned ensemble $\mathcal{M}$ as few times as necessary while also being able to react to concept drifts in the time-series we propose to detect two different types of drift. First, as we already detailed in Section 3.6, we use the Hoeffding-bound on the mean values of the raw time-series to trigger a concept drift if the mean changes significantly over time. Specifically, we will denote with Δ_t the difference of means at time t according to Eq. (3.5). The reasoning is that changes in the data distribution will allow us to adopt each forecasters RoC with novel patterns while also giving us the opportunity to re-create the pruned ensemble. We will refer to this type of concept drift as `Drift Type I` and, if our drift detection mechanism triggers, we will enrich the RoCs (see Section 3.6 and Section 4.1). Furthermore, we will also re-cluster and re-prune the ensemble based on the current input as described in Section 4.2. We do this to account to the fact that, given the enrichment of the RoCs, we might expect that ensemble based on new RoC entries can be formed with lower expected error since it can account for novel patterns in the data.

In addition, we propose to monitor another quantity over time, whose drift we will denote as `Drift Type II`. The performance of a forecaster j is reflected by the distance of its representative RoC entry $\hat{\boldsymbol{r}}_j$ to the current pattern. For notational simplicity, we will denote the current pattern again with $\boldsymbol{x}_t$. This distance can be continuously computed and monitored over time for all forecasters. The distance d_j^t between the representative RoC entry of the model f_j and $\boldsymbol{x}_t$ can be calculated using Euclidean distance as

$$d_j^t = \|\boldsymbol{x}_t - \hat{\boldsymbol{r}}_j\|_2, \quad \forall j \in [N]. \tag{4.6}$$

Let $\boldsymbol{D}_t := (d_j^t)_{j\in[M]}^T$ be the distance vector of all models in the current ensemble $\mathcal{M}$ to the current input pattern at time t. Also, we define $\boldsymbol{D}_0$ to be the distance vector at the last time the ensemble was pruned. Then, we can interpret the difference in absolute change between the current minimum distance and the initial minimum distance as

$$\varepsilon_t = |\min(\boldsymbol{D}_t) - \min(\boldsymbol{D}_0)|.$$

We can now monitor ε_t using the Hoeffding bound, similar to the change in mean Δ_t of `Drift Type I`. Essentially, we are monitoring a proxy of the smallest expected error at time t over time and if that quantity significantly deviates from zero, i.e., from the original smallest expected error, a drift in model performance is assumed to take place. Once this drift is detected the ensemble is updated by re-clustering and re-pruning $\mathcal{M}$. Note, however, that we do not enrich the RoCs since a drift in performance does not necessarily indicate novel patterns in the data. We proceed by setting $\boldsymbol{D}_0 = \boldsymbol{D}_t$ (assuming the drift was detected at time instant t).

Our method is denoted in the following as OEP-ROC: Online Ensemble Pruning using Performance Saliency maps-based Regions of Competence. All the steps of OEP-ROC are summarized in Algorithm 4.1.

Algorithm 4.1 OEP-ROC

1: Train each $f_k \in \mathcal{P}$ on X^{train}
2: Initialize RoCs: $\mathcal{R}_k = \{\}\ \forall j \in [N]$
3: **for** $\boldsymbol{X}_i^{\text{val}} \in W_{z,\omega}(\boldsymbol{X}^{\text{val}})$ **do** ▷ Eq. (3.1)
4: **for** forecaster $k \in [M]$ **do** ▷ Section 4.1
5: **for** $(\boldsymbol{x}_i^{(j)}, y_i^{(j)}) \in W_{1,L,1}(\boldsymbol{X}_i^{\text{val}})$ **do** ▷ Eq. (3.2)
6: Compute PGSM explanations $\boldsymbol{a}_i^{(j)}$ ▷ Section 3.4
7: Compute salient subsequences using $\boldsymbol{a}_i^{(j)}$ ▷ Fig. 3.1
8: Add subsequences to the corresponding RoC $\mathcal{R}_k$ ▷ Fig. 3.1
9: **end for**
10: **end for**
11: **end for**
12: Let $\boldsymbol{x}_t$ be the first test pattern
13: Compute the closest RoC patterns $\hat{\boldsymbol{r}}_j\ \forall j \in [M]$ ▷ Eq. (4.1)
14: Cluster the closest RoC patterns into k clusters ▷ Section 4.2.1
15: Let $\mathcal{K} = \{\boldsymbol{\rho}_j \mid j \in [M]\}$ ▷ Section 4.2.1 and Eq. (4.5)
16: Select $\mathcal{M}$ using δ radius ▷ Proposition 4.1
17: Predict $\boldsymbol{x}_t$ with ensemble $\mathcal{M}$
18: Initialize $\boldsymbol{D}_0$ ▷ Section 4.3
19: **for** $i \in \{1, \cdots\}$ **do**
20: Predict $\boldsymbol{x}_{t+i}$ with $\mathcal{M}$
21: **if** `Drift Type I` detected **then**
22: Update $\boldsymbol{X}^{\text{val}}$ with newly observed data
23: Enrich RoCs ▷ Lines 3-11
24: **end if**
25: **if** `Drift Type I` $\vee$ `Drift Type II` detected **then**
26: Re-cluster and re-prune $\mathcal{M}$ with $\boldsymbol{x}_{t+i}$ and X_{t+i+1} ▷ Lines 13-16
27: **end if**
28: **end for**

4.4. Experiments

Next, we present the experiments carried out to validate OEP-ROC as a whole and in terms of the benefit of each component we propose. To do that, we address the following research questions:

- **Q1:** How does OEP-ROC perform compared to the state-of-the-art and existing online ensemble pruning methods for time-series forecasting?
- **Q2:** What is the importance of each pruning stage in OEP-ROC?
- **Q3:** What is the benefit of each drift type detection for the performance of OEP-ROC?
- **Q4:** What is the impact of different values of the number of clusters k on the performance of OEP-ROC?
- **Q5:** What is the impact of choosing the size of $\mathcal{M}$ automatically using the bound in Proposition 4.1?
- **Q6:** How can different aggregation techniques benefit from our pruning method?
- **Q7:** How scalable is OEP-ROC in terms of computational resources compared to the most competitive online ensembling methods? What is the computational advantage of the **drift-aware** adaption of pruning?
- **Q8:** How can OEP-ROC be exploited to provide suitable explanations for the reason behind selecting specific models to compose the ensemble at a certain time interval or instant?
- **Q9:** How can OEP-ROC be used also to provide reasonable explanations for the performance of the selected ensemble at a certain time interval or instant?

4.4.1. Experimental Setup

The experimental setup proposed here is similar to Section 3.7. Specifically, we evaluate all methods in terms of their root mean squared error (see Section 2.4.3). Moreover, we again use the first 50% of each time-series to fit the individual forecasters ($\boldsymbol{X}^{\text{train}}$), the following 25% for validation ($\boldsymbol{X}^{\text{val}}$) and the last 25% for testing ($\boldsymbol{X}^{\text{test}}$). We use 100 real-world time-series shown in Table 4.1 for our experiments. For datasets from the Monash Forecasting Repository [God+21] we sample random time-series until we arrive at a total of 100 diverse time-series due to computational limitations. The code to reproduce all experiments is available online[6].

[6] https://github.com/MatthiasJakobs/os-pgsm/tree/ecml_journal_2022

Name	Nr.	Source	Characteristics
Amount registered	1	[Cer+17a]	Hourly, Jan. 1, to Mar. 01, 2011
AbnormalHeartbeat	1	[Bag+17]	3053 measurements (4kHz)
CatsDogs	1	[Bag+17]	14773 audio samples (16kHz)
Cricket	1	[Bag+17]	1197 accelerometer readings (184Hz)
EOGHorizontalSignal	1	[Bag+17]	1250 measurements (1kHz)
EthanolConcentration	1	[Bag+17]	1 second spectrum measurement
Phoneme	1	[Bag+17]	1024 samples of audio
Rock	1	[Bag+17]	2844 samples of spectrum analysis
SNP500	1	[DG17; HH19]	Daily closing, 2010 to 2017
DJI	1	[DG17; HH19]	Daily closing, 2010 to 2017
NYSE	1	[DG17; HH19]	Daily closing, 2010 to 2017
RUSSELL	1	[DG17; HH19]	Daily closing, 2010 to 2017
Electricity (Hourly)	11	[God+21]	Energy consumption measurements
KDD Cup 2018	13	[God+21]	Forecast air quality indices (AQIs)
Pedestrian Counts	12	[God+21]	Hourly pedestrian counts from Melbourne
Solar (10 minutes)	12	[God+21]	Solar power production
M4 (Daily)	12	[God+21]	Daily time-series from M4 dataset
M4 (Weekly)	13	[God+21]	Weekly time-series from M4 dataset
Weather	15	[God+21]	Daily weather forecasts

Table 4.1.: List of datasets used for our experiments. In total, 100 time-series were used.

4.4.2. OEP-ROC Setup and Baselines

We construct a pool $\mathcal{P}$ of CNN-based forecasters using different parameter settings (e.g., varying the number of filters) as mentioned in Section 4.1. For construction of the base learners, we define four architectural building blocks, three of which are shown in Fig. 4.2. We use the notation of `layer1`→`layer2` to imply a sequential connection between the two layers. Let `C1` be a sequential subnet made up of a 1D convolutional layer, followed by a ReLU activation and a batch normalization layer. `C2` is similar, except that we use max pooling instead of batch normalization. `C3` (which we do not show in Fig. 4.2) is the same as `C1` except that the number of filters for the convolutional layer is reduced by half as compared to `C1`. Lastly, we define a *residual block* `RB` [He+16] (Fig. 4.2, rightmost). A residual connection, which adds the unprocessed input of the block to the computed output of the block, allows for an efficient flow of gradients during backpropagation as well as to learn an identity function easier. The last part will help to combat overparameterization and overfitting. From these building blocks, we create our forecasters as outlined in Table 4.2. Notice that we also utilize Dropout layers `DO` with a probability parameter of $p = 0.9$. We denote a LSTM layer with `LSTM` and a simple linear layer with `Lin`. We also show the different configurations we created by varying the number of filters n_f in the convolutional layers as well as the number of hidden units n_h in the LSTM layers. Moreover, our proposed method OEP-ROC has a number of hyperparameters. We summarize

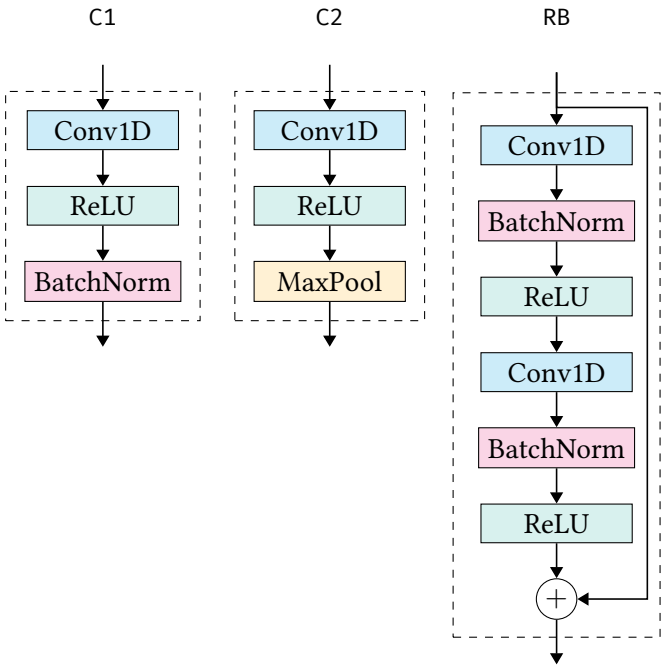

Figure 4.2.: Building blocks which we use to construct the forecasters.

the selection for OEP-ROC in Table 4.3 and will evaluate different parameterizations in this section.

We compare our OEP-ROC method against the following approaches which include state-of-the-art methods for forecasting and ensemble pruning. Some of them operate in an online fashion. Note that some methods were already mentioned in Section 3.7. However, for convenience, we will also describe them here briefly.

- `ARIMA` (Section 2.3.2): Auto-Regressive Moving Average model. As in Section 3.7 we estimate the parameters (p, d, q) automatically using the `forecast` library[7].
- `ETS` (Section 2.3.2): Exponential Smoothing model. As in Section 3.7 we estimate the parameter K automatically using the `forecast` library as well.
- `LSTM` [HS97; GES01]: A simple one-layer LSTM.
- `CNN-LSTM`, `CNN`: The two best performing models from $\mathcal{P}$ with and without a LSTM component (see Table 4.2), respectively.
- `Ran-Pr-`m: Random selection of base models to construct an ensemble with m models.
- `Ens`: Ensemble of all the models in the pool $\mathcal{P}$.

[7] https://github.com/robjhyndman/forecast

Forecaster name	Architecture	Configurations
Shallow FCN	C1→DO→Lin	$n_f \in \{32, 64, 128\}$
Small CNN	C2→LSTM→Lin	$n_f \in \{32, 64, 128\}, n_h \in \{10, 30\}$
Medium CNN	C1→C1→LSTM→Lin	$n_f \in \{32, 64, 128\}, n_h \in \{10, 30\}$
Large CNN	C1→C1→C1→LSTM→Lin	$n_f \in \{32, 64, 128\}, n_h \in \{10, 30\}$
Fewer Filter CNN	C3→C3→C3→LSTM→Lin	$n_f \in \{32, 64, 128\}, n_h \in \{10, 30\}$
One Residual	RB→DO→Lin→DO→Lin	$n_f \in \{32, 64, 128\}$
Two Residual	RB→RB→DO→Lin→DO→Lin	$n_f \in \{32, 64, 128\}$

Table 4.2.: Configurations and architectures for all forecasters. Different configurations are generated by taking all combinations of filters and hidden units as described in the last column. In total, this results in 33 models in our pool $\mathcal{P}$.

Parameter	Description	Value
L	Number of lagged values	5
ω	Size of chunks within the validation set	60
z	Step size for windowing	25
k	Number of clusters	15
τ	Threshold for subsequence extraction	0.1
σ	Hoeffding-Bound confidence	0.95

Table 4.3.: Hyperparameters of our method and their values for the experiments.

- NCL [MA14]: Negative correlation Learning for pruning ensembles of deep learning methods. NCL explicitly optimizes for high ensemble diversity by incorporating a diversity term to the loss of each model. We use the same pool of models as $\mathcal{P}$.
- OCL [SPM19]: An online ensembling approach that uses covariance-based clustering based on recent predictions of the models in $\mathcal{P}$.
- OTOP [SPM19]: A online approach which uses Scaled Root Correlation to select top-performing models from $\mathcal{P}$ for ensembling.
- DEMSC-C [SPM19]: An approach which combines OTOP for a pre-selection and uses OCL for further ensemble pruning.
- DEMSC-K [SPM19]: Same as DEMSC-C but uses k-means clustering with DTW as a distance measure instead of the covariance-based clustering approach in OCL.
- OS-PGSM and OS-PGSM-Int (Chapter 3): The approach presented in the previous chapter for online model selection, as well as the faster variant to compare ensemble performance in general to model selection.

Moreover, we want to investigate the importance of each component of our approach. To do that, we configure and investigate the following variants of OEP-ROC:

- OEP-ROC-C: Variant of OEP-ROC that uses only clustering without $\mathcal{M}$ selection, i.e., it performs only the first pruning stage and uses the set of cluster representatives $\mathcal{K}$ for ensembling.
- OEP-ROC-TOP: Variant of OEP-ROC that computes $\mathcal{M}$ without clustering, i.e., it performs only the second pruning stage on $\mathcal{P}$.
- OEP-ROC-ST: Static variant of OEP-ROC. Pruning decided at the initial forecasting instant and kept fixed along testing, i.e., both concept drift detection mechanisms are disabled.
- OEP-ROC-Per: Pruning is updated periodically in a blind manner (i.e. without taking into account the occurrence of the drift), similar to OS-PGSM-Per from Section 3.7.
- OEP-ROC-I: Variant of OEP-ROC that only monitors `Drift Type I`.
- OEP-ROC-II: Variant of OEP-ROC that only monitors `Drift Type II`.
- OEP-ROC-k: Variant indicating the number k of clusters used in the clustering stage of OEP-ROC.
- OEP-ROC-k-M: In OEP-ROC and all its above variants the size of $\mathcal{M}$ (i.e. $M = |\mathcal{M}|$) is decided automatically using the bound from Proposition 4.1. In this variant, we set the number of desired models to the fixed value of M (in addition to setting the number of clusters to k).

We also evaluate how different ensemble aggregation and weighting methods from the literature can benefit from our pruning strategy. Instead of using all forecasters in $\mathcal{P}$ into the weighting schema, we only use the models left after our proposed pruning approach. To do so, we report the evaluation results over various aggregation methods including:

- `SW` [Saa+20]: Sliding-window ensemble which weighs the ensemble member based on recent losses computed on sliding-windows. The variant OEP-ROC-`SW` uses this approach for weighting the final $\mathcal{M}$ members.
- `OGD` [Zin03] : Online Gradient Descent aggregation over all the models in $\mathcal{P}$. OEP-ROC-`OGD` uses `OGD` on $\mathcal{M}$.
- `EW` [CL06]: Compute weighting with Exponential Weighting aggregation over recent regret of models in $\mathcal{P}$. The OEP-ROC-`EW` variant uses this weighting on $\mathcal{M}$ instead.
- `FS` [CL06]: Similar to `EW`, but forces the weights to stay above a certain level. OEP-ROC-`FS` adopts this approach on $\mathcal{M}$.
- `MLPOL` [Win17]: Polynomial Potential aggregation rule for aggregation over all the models in $\mathcal{P}$. The variant OEP-ROC-`MLPOL` uses this aggregation rule on $\mathcal{M}$.

Method name	Losses	Wins	Average Rank	±
ETS	11(4)	89(71)	24.98	9.36
OS-PGSM	11(3)	89(71)	24.76	3.11
OS-PGSM-Int	17(8)	83(66)	23.30	4.84
CNN	14(7)	86(66)	20.32	6.90
Ran-Pr-5	20(5)	80(69)	18.35	4.93
Ran-Pr-10	25(6)	75(68)	15.91	5.03
DEMSC-K	19(7)	81(60)	15.52	7.45
OTOP	25(5)	75(60)	15.32	9.89
ARIMA	16(7)	84(73)	14.91	9.69
DEMSC-C	32(19)	68(55)	13.68	7.66
Ran-Pr-15	31(3)	69(61)	12.82	6.17
CNN-LSTM	26(12)	74(62)	12.63	7.55
Ran-Pr-20	32(16)	68(58)	10.42	6.17
OEP-ROC-II	30(7)	70(42)	10.08	5.83
OEP-ROC-PER	27(9)	73(61)	10.07	5.84
NCL	49(25)	51(26)	9.07	9.28
OEP-ROC-I	33(7)	67(35)	8.98	5.69
LSTM	43(30)	57(49)	8.88	8.41
Ens	47(27)	53(33)	7.73	3.34
OEP-ROC-ST	37(10)	63(34)	6.31	4.83
OEP-ROC-15-6	33(11)	67(33)	6.25	5.19
OCL	36(17)	74(61)	6.17	6.66
OEP-ROC-15-8	33(15)	67(36)	5.59	4.53
OEP-ROC-TOP	38(13)	62(33)	4.97	5.84
OEP-ROC-15-10	39(10)	61(35)	4.90	4.55
OEP-ROC-C	39(13)	61(33)	3.79	3.42
OEP-ROC	-	-	3.29	3.08

Table 4.4.: Comparison of OEP-ROC to different state-of-the-art methods for 100 time-series. An average rank of 1 means the model was the best performing on all time-series.

4.4.3. Results

Table 4.4 presents the average ranks plus their standard deviation for OEP-ROC, its variants and state-of-the-art methods for time-series forecasting and online ensemble pruning. For the paired comparison, we compare our method OEP-ROC against each of the other methods. We counted wins and losses for each dataset using the RMSE errors. We use the non-parametric Wilcoxon Signed Rank test to compute significant wins and losses, which are presented in parentheses (significance level $\alpha = 0.05$) (see Section 2.4.3 for more information).

Method	Avg. Rank	±
OEP-ROC-II	9.98	5.91
OEP-ROC-PER	9.96	5.91
OEP-ROC-I	8.88	5.76
OEP-ROC-5	6.50	6.28
OEP-ROC-ST	6.19	4.91
OEP-ROC-15-6	6.14	5.27
OEP-ROC-15-8	5.84	4.61
OEP-ROC-20	5.74	5.27
OEP-ROC-10	5.73	5.87
OEP-ROC-TOP	4.86	5.55
OEP-ROC-15-10	4.79	4.62
OEP-ROC-C	3.68	3.49
OEP-ROC	3.18	3.15

Table 4.5.: Comparison of OEP-ROC to its variants for all 100 time-series. The rank column presents the average rank and its standard deviation across different time-series. A average rank of 1 means the model was the best performing on all time-series.

Comparing OEP-ROC to the state-of-the-art approaches

As can be seen from the results in Table 4.4, OEP-ROC outperforms all the baseline methods in terms of ranks, wins and loses in pairwise comparison. In fact, the top 5 best performing methods are all variants of OEP-ROC. The best performing methods besides OEP-ROC are the full ensemble (`Ens`), ensembling using covariance-based clustering (`OCL`) and negative correlation learning (`NCL`). However, their average ranks are two to three time higher than that of OEP-ROC, showing that we are able to significantly outperform not only strong competitive methods but also the complete ensemble consisting of all 33 forecasters. In general, the online ensemble pruning methods, e.g., `OCL`, `OTOP`, `DEMSC-K` and `DEMSC-C`, show inferior performance compared to OEP-ROC. They are also outperformed by random ensembles, as well as the simple, one-layer `LSTM`. Individual forecasters such as `ARIMA`, `ETS`, and `CNN` are also considerably worse in average rank compared to OEP-ROC. While `LSTM` and `CNN-LSTM`, which are chosen based on average performance over all datasets, are slightly more competitive they are still performing significantly worse than OEP-ROC. We want to note again that the full ensemble `Ens` which uses all 33 models has worse average rank comparing to OEP-ROC which uses on average uses merely 6 models, almost a fifth of the pool $\mathcal{P}$ size. These results address research question **Q1**.

Comparing OEP-ROC to its variants

Next, we conduct an ablation study to investigate the importance of different aspects of our proposed method. As can be seen from Table 4.5, neither pruning stage on its own (i.e. OEP-ROC-C and OEP-ROC-TOP) is able to achieve good results similar to OEP-ROC,

which shows the importance of combining both stages. This is most likely due to the fact that out bound gets tighter for smaller pools of models, meaning that clustering first and pruning further afterwards is beneficial to each individual stage. We also observe that OEP-ROC-C has better performance compared to OEP-ROC-TOP. Only ranking the base models according to their closeness (i.e., OEP-ROC-TOP) diminished the diversity since the selection is made in favor of RoCs that are most similar to the current pattern and as a result more or less similar to each other. We think that a preceeding clustering stage is vital to account for RoC similarity and thus increasing diversity early. The results show that our two-staged procedure is able to establish a trade-off between ensemble's diversity and accuracy. With that we answer research question **Q2**.

In terms of the importance of both types of concept drift we can see from Table 4.5 that neither of the drift detection mechanisms (i.e. OEP-ROC-I and OEP-ROC-II) on its own is able to achieve good performance. The combination of both, as with the pruning stage before, is necessary. It is also clear that performing the updates periodically in a random blind manner with OEP-ROC-Per does not necessarily improve the performance even though it is designed to trigger more updates than all the drift-aware methods. Similar to our results from Section 3.7 we see that too many (false) concept drift detections are not only unnecessary but can actually reduce predicting performance. Our informed adaption is able to achieve high performance by updating the RoCs and ensembles only if necessary. With that we answer research question **Q3**.

Optimal choice of k and M

In OEP-ROC, we set $k = 15$ and let M be determined dynamically. We observe that the right number of clusters to be computed seems to also be an important factor for the performance. It can be seen from Table 4.5 that a low values of $k = 5$ does not help to achieve good performance. This is due to the fact that few clusters will lead to larger cluster sizes which means that the selection of the clusters' representatives will no longer be representative of the clusters overall error. Increasing the value of k too much, i.e., to $k = 20$, is also not desirable since it leads to small clusters and more similar cluster representatives which may alter the diversity by decreasing the ambiguity. With $k = 15$, which is roughly $|\mathcal{P}|/2$, we are able to strike a balance which results in the best performance compared to other sizes of clustering. This answers research question **Q4**.

Finally, Table 4.5 also shows the usefulness and the benefits of our theoretical insights in setting up the size of $\mathcal{M}$ automatically. This size is set up by the derived bound (Proposition 4.1) such that the base models' average error is reduced, and the ambiguity is increased. Fixed values for M could not achieve the same performance as OEP-ROC. The best fixed model selection configuration is OEP-ROC-15-10 which has a higher average rank than OEP-ROC. Using our bound is more theoretically principled, requires no hyperparameter optimization, works for different number of clusters and leads to better average ranking. With that, we answer research question **Q5**.

Method	Avg. Rank	$\pm$
SW	7.10	2.54
EWA	6.83	2.78
OEP-ROC-SW	6.21	3.95
OEP-ROC-EWA	5.25	2.24
FS	5.11	2.75
OGD	4.81	1.52
OEP-ROC-OGD	4.75	1.75
OEP-ROC-FS	4.61	2.16
MLPOL	3.85	1.71
OEP-ROC-MLPOL	3.11	2.05

Table 4.6.: Comparison of OEP-ROC combined with different aggregation methods for all 100 time-series.

Name	Mean runtime	Variance of runtime
OEP-ROC	14.41	13.03
OEP-ROC-ST	0.49	0.24
OEP-ROC-Per	27.61	0.66

Table 4.7.: Average runtime plus variance (both in seconds) for three variants of OEP-ROC over 5 datasets.

Combining OEP-ROC with different aggregation methods

It can be seen from Table 4.6 that all aggregation methods achieve better results when combined with OEP-ROC than using all models in $\mathcal{P}$. The advantage in performance is clearly seen especially for `SW`, `EWA` and `MLPOL`. This can be explained by the fact that the dimension of the input for the ensemble's weights learning is reduced from $|\mathcal{P}| = 33$ to an average of $|\mathcal{M}| = 6$. This make the meta-learning task in `EWA` and `MLPOL` much easier. In addition, the weighting schema is focused around the most suitable forecasters in terms of accuracy and diversity. Since we already optimize for that with our two-staged pruning approach this also makes the process easier. This addresses research question **Q6**.

Scalability analysis

To compare scalability between different versions of OEP-ROC we considered OEP-ROC, OEP-ROC-Per and OEP-ROC-ST, because these configurations nicely illustrate which steps of our algorithm are most costly. We show the results in Table 4.7. As can be seen, OEP-ROC-ST is by far the fastest method on average, since it does not adapt its RoCs during the algorithms runtime. We noticed that recreating the RoCs takes by far the longest time, which we already observed in Section 3.7. OEP-ROC-Per illustrates this problem the best, since it blindly and frequently recreates the RoCs and re-cluseters the models.

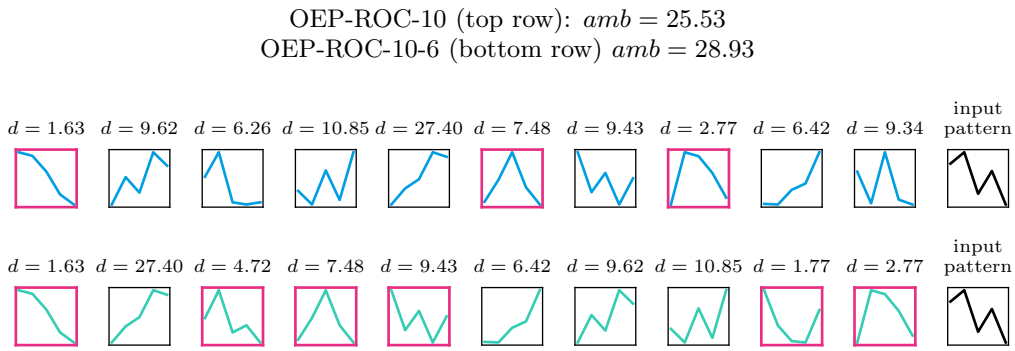

Figure 4.3.: Comparison of the clusters' representatives of OEP-ROC-10 (top row) with $k = 10$ and OEP-ROC-10-6 (low row) (also $k = 10$) on the Abnormal Heartbeat dataset. We report the ambiguity amb of the clustered ensemble as well as the euclidean distance of each RoC to the input pattern, which is shown in the right most column. In red, we visualize the models that where chosen by each method for prediction, i.e., each method's $\mathcal{M}$.

Thus, its runtime is always high, no matter if an adaption to new time-series properties is necessary or not. OEP-ROC strikes a balance between these two extremes and detects if an adaption to a drift is necessary. As can be seen from the table, this results in a high variance of the runtime, since some datasets contain more concept drifts than others. We see this behavior as beneficial, since OEP-ROC outperforms the other two methods we measured its runtime against, indicating that sometimes a higher runtime can be justified by overall better performance. With this we answer reserach question **Q7**.

Explainability Aspects

Lastly, we want to discuss the various explainability aspects which are possible due to our usage of refined RoCs. We can provide some insights into how OEP-ROC can be used to provide suitable explanations for the reason behind specific forecaster selection to construct the ensemble at a specific time instant. First, we compare the clusters' representatives of OEP-ROC-10 (top row) and OEP-ROC-10-6 in Fig. 4.3. The current input pattern is shown in black on the right side. It can be clearly seen that OEP-ROC selects the RoCs that are quite different from each other but which are still related to the current pattern. This shows the trade-off established by OEP-ROC between expected error and diversity. OEP-ROC-10-6 is promoting the selection of diverse pattern. However, the fixed size results in too many models being picked which increase the prediction error. This addresses research question **Q8**.

Figure 4.4 shows a comparison between the current input time-series pattern $\boldsymbol{x}_t$ (left plot, in black) with the RoCs of the pruned ensemble to perform the forecast on the right (light blue). A clear similarity between the trend in both patterns can be observed which justifies the choice of this ensemble construction since it has been proven to show some

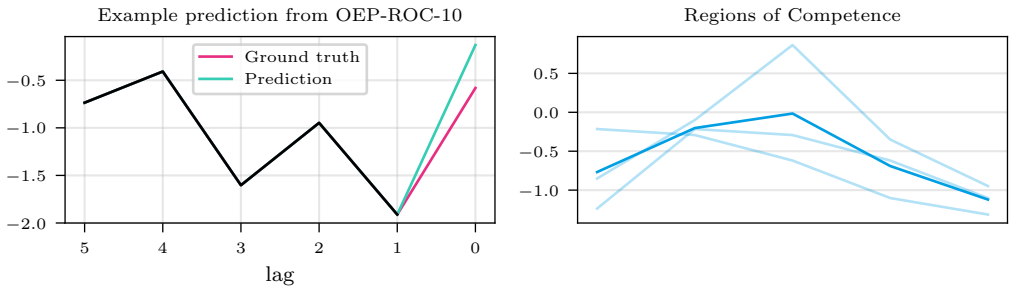

Figure 4.4.: Left: For the time-series to predict (black), we show the ground truth value (red) as well as the prediction of OEP-ROC-10 (green). **Right**: Visualization of each nearest RoC from the selected models (light blue), as well as the mean RoC (darker blue). Both plots were generated on the Abnormal Heartbeat dataset.

degree of competence in forecasting using patterns with similar trend as input. This is further validated when also comparing between the true time-series value (ground truth, red) and the predicted value (green). While these two values differ slightly, an evaluation of all the candidates in this point showed that the ensemble by OEP-ROC has the smallest error. This addresses research question **Q9**.

4.5. Concluding remarks

In this chapter we presented a novel online ensemble pruning method utilizing the previously proposed Performance Gradient-based Saliency Maps from Chapter 3. The results of our experiments indicate that OEP-ROC has performance advantages compared to popular forecasting methods and state-of-the-art approaches for online ensemble pruning. We show that our method utilizing both refined RoCs from Chapter 3 and a two-stage pruning approach is able to gain excellent and reliable empirical performance in our setting. The informed adaption and update of the pruning decision following concept drift detection in both time-series and forecasters' performance makes our method in addition to better predictive performance computationally cheaper than a blind, periodic update. OEP-ROC can also be used successfully for providing useful explanations behind the selection of a specific subset of forecasters to compose the ensemble at a given time instant or interval.

5. Tree-based Model Selection

In the previous two chapters we have seen how explainability methods, specifically Grad-CAM, can be used to make online model selection and ensemble pruning more explainable. Specifically, by identifying most salient subsequences and adding these to each model's RoC we are able to get an understanding about the types of input each model performs well at. However, one aspect which was not addressed in this thesis is the fact that CNNs, which are required for Grad-CAM and PGSM to work, can be very uninterpretable. Especially in a high-parameter setting where multiple convolution layers are stacked, followed by (deep) FCN blocks for prediction, it is often very unclear how a model actually computes the prediction.

As outlined in Section 2.4.2, tree-based models learn to partition the input domain into regions, each of which is associated with a fixed prediction. In this chapter we want to investigate the feasibility of using tree-based models for time-series forecasting under concept drift, specifically in the previously presented online model selection framework. However, we are not able to compute the PGSMs the same way as before as mentioned earlier. To address this, we propose to utilize Shapley values for explaining the predictions of the models instead. Shapley values are a popular attribution method adapted from Lloyd Shapley's Nobel Prize winning Game Theory approach [Sha53] to fairly attribute the outcome of a game onto its players. Essentially, we can use Shapley values in the same way as Grad-CAM explanations in order to find the most salient subsequences. In addition, we can provide a more in-depth analysis into the inner workings of the forecasters if they are tree-based. Specifically, we can compute and visualize intervals on the lagged time-series values for which a model will output the same prediction. That way it is possible to gain an understanding about the confidence in the prediction as well as a measure of robustness towards outliers.

This chapter is based on the publication [JS23]. The author of this thesis devised the idea of the paper and the Methodology. Moreover, he implemented the code for the experiments. He also wrote large parts of the original paper.

We will proceed in this chapter by presenting Shapley values in a general setting, followed by a method to efficiently estimate them if we assume that the models are tree-based. Afterwards, we will discuss how we use these Shapley values to compute tree-model RoCs efficiently. Moreover, we present our experiments which validate predicting performance, as well as computational efficiency, in the following section. This section will also contain a comprehensive discussion about the interpretability and explainability aspects which are possible by utilizing tree-based models.

5.1. Shapley values

As mentioned earlier, Shapley values were first presented in the field of *Game Theory* by Lloyd Shapley [Sha53]. They aim to describe how to fairly distribute the *value* of a *cooperative game* onto all participating *players*. For example, consider a team of employees in a company, collectively working towards producing a product that is sold for a fixed amount of money. One might be inclined to distribute the profit fairly onto the players according to how much they contributed to the final output. To do that, Shapley values formalize this problem in the following way:

Let $\mathcal{S} \subseteq \mathcal{N}$ be a (sub)set of all players $\mathcal{N}$ contributing to the game. In the above example, $\mathcal{S}$ would be a subset of all employees in the team. We will denote with S and N the size of sets $\mathcal{S}$ and $\mathcal{N}$, respectively. Then, let $v : 2^{\mathcal{N}} \to \mathbb{R}$ be a *game* or *value function*, which maps a subset of players onto the output that this set produces. Note that the outcome of no player contributing is set to 0, i.e., $v(\emptyset) = 0$.

Definition 5.1 (Shapley values [Sha53])**.** *Let $\mathcal{N}$ be a set of players and $v : 2^{\mathcal{N}} \to \mathbb{R}$ be a value function. Then, the Shapley value for player $i \in \mathcal{N}$ is given by*

$$\phi_i(v) = \sum_{\mathcal{S} \subseteq \mathcal{N} \setminus \{i\}} w(S, N) \left(v(\mathcal{S} \cup \{i\}) - v(\mathcal{S}) \right)$$

where $w(S, N) = \frac{S!(N-S-1)!}{N!}$.

Essentially, the Shapley value is the marginal contribution of a player i to a game considering each possible coalition of players with and without player i. Interestingly, the Shapley value was derived in [Sha53] as the unique attribution method that satisfies the following three axioms. This means that whenever any additive attribution method satisfies these three axioms it can be rewritten as the Shapley value of a certain value function.

Axiom 5.1 (Efficiency). Let $\phi_i(v)$ be the attribution of player i unto game v with N players. Then

$$\sum_{i=1}^{N} \phi_i(v) = v(\mathcal{N}).$$

Axiom 5.2 (Symmetry).

$$v(S \cup \{i\}) = v(S \cup \{j\}) \Leftrightarrow \phi_i(v) = \phi_j(v) \quad \forall \mathcal{S} \subseteq \mathcal{N} \setminus \{i, j\}.$$

Axiom 5.3 (Linearity). Let v and v' be two games and $a, b \in \mathbb{R}$. Then

$$\phi_i(av + bv') = a\phi_i(v) + b\phi_i(v') \; \forall i \in [N].$$

In the original manuscript, Shapley does not use the Linearity axiom as shown above, but an axiom called *law of aggregation*. This axiom, however, is equivalent to setting

$a = b = 1$ above, and it is easy to see that we can generalize the axiom to linearity. One often cited fourth axiom called *Null-player* or *Dummy* can technically be derived as a consequence of Axiom 5.1 [Sha53] and thus does not need to be an axiom itself.

Lemma 5.1 (Null-player). If there exists $i \in \mathcal{N}$ with $v(\mathcal{S} \cup \{i\}) = v(\mathcal{S})\ \forall \mathcal{S} \subseteq \mathcal{N} \setminus \{i\}$ then $\phi_i(v) = 0$.

In later works on Shapley values, Young [You85] was able to show that the Linearity and Null-player axioms can be replaced by the following Monotonicity axiom. This reformulation retains the fact that the unique solution to these three axioms are Shapley values.

Axiom 5.4 (Monotonicity). Let $\Delta_i(v, \mathcal{S}) := v(\mathcal{S} \cup \{i\}) - v(\mathcal{S})$ be the difference in value function when adding player i to subset $\mathcal{S}$. Further, let v and v' be two value functions over the same set of players $\mathcal{N}$. Then

$$\Delta_i(v, \mathcal{S}) \geq \Delta_i(v', \mathcal{S}) \quad \forall \mathcal{S} \subseteq \mathcal{N} \setminus \{i\} \Rightarrow \phi_i(v) \geq \phi_i(v').$$

Due to their desirable properties (in the forms of axioms described above) Shapley values have found a large adoption in XML literature [LL17; AJL21; Jak+22]. Most prominently, they are used as a method for local feature attributions. Recall from Section 2.5 that a local feature attribution estimates the importance of each feature towards a specific, individual prediction. Let $f : \mathcal{X} \to \mathcal{Y}$ be a machine learning model which maps vectors from the input space to some output, e.g., to a class probability (in the classification case) or a scalar (in the regression and forecasting case). There are no further assumptions about the model, such as the presence or absence of feature interaction or a (non)linear behavior made at this point, besides that the model is deterministic. Thus, the model is assumed to be a *black-box* model which we can merely observe as mapping some input to outputs. We have seen earlier that Shapley values measure the marginal contribution that player or feature i contributes to all subsets of the remaining features. Thus, we have to somehow encode the fact that certain features are *missing* for a fixed input function such as f. This is often achieved by using a conditional expectation as the game [ŠK14; LL17] where, for a subset of features $\mathcal{S} \subseteq \mathcal{N}$ of size S and a fixed input $\boldsymbol{x} \in \mathcal{X}$, the value function is defined as

$$v(\mathcal{S}, \boldsymbol{x}) = \mathop{\mathbb{E}}_{\boldsymbol{X}}[f(\boldsymbol{X} | \boldsymbol{X}_\mathcal{S} = \boldsymbol{x}_\mathcal{S})]. \tag{5.1}$$

The expectation is taken over all data points from $\mathcal{X}$ where the features in $\mathcal{S}$ are equal to that of the data point $\boldsymbol{x}_S$. However, there are downsides to this approach. First, the exact distribution of the data is often unknown, making estimating the true expected value difficult. But even if the distribution is empirically estimated how to model the dependence structure of the features to accurately sample examples is a hard problem [AJL21]. Moreover, iterating over all possible subsets $\mathcal{S} \subseteq \mathcal{N}$ requires exponentially many function evaluations, making a straight-forward computation impossible for datasets with

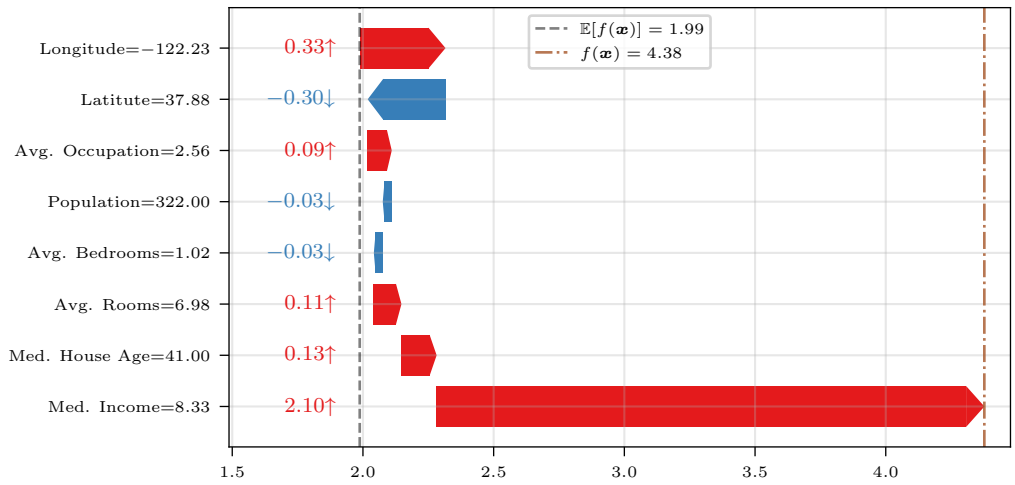

Figure 5.1.: Visualization of Shapley values computed on one instance of the California house price prediction task [PB97]. The features and their values are shown on the y axis. Beginning with the expected prediction $\mathbb{E}[f(\boldsymbol{x})]$ (shown in gray) Shapley values indicate the amount of change in prediction due to the specific feature value (shown in red and blue). Notice that the sum of all changes adds up to the actual prediction $f(\boldsymbol{x})$ (shown in brown), as expected from Axiom 5.1. The target value (x axis) is given in a hundred thousand Dollars.

many features. Thus, there are numerous approaches presented in the literature on how to compute the expectation in Eq. (5.1) in practice. Two common approaches include a straight-forward sampling approach [ŠK14] and KernelSHAP [LL17]. If we denote with $\overline{\mathcal{S}}$ the complement of $\mathcal{S}$ w.r.t. the set of features $\mathcal{N}$, both works assume feature independence by approximating Eq. (5.1) via

$$v'(\mathcal{S}, \boldsymbol{x}) = \mathop{\mathbb{E}}_{\boldsymbol{X}_{\overline{\mathcal{S}}}} [f(\boldsymbol{x}_{\mathcal{S}}, \boldsymbol{X}_{\overline{\mathcal{S}}})] \tag{5.2}$$

where the input to f is constructed with the original values of $\boldsymbol{x}$ at $\mathcal{S}$ and independent samples for the remaining features. In the popular KernelSHAP method [LL17], the authors estimate a linear model using these permuted samples (and the corresponding model predictions as a target), effectively using Shapley values to explain the local neighborhood around the original sample. See Section 5.1 for a visualization of Shapley values for a tabular regression task.

One downside of assuming independence among the features of the model f is that the model is evaluated on possibly unrealistic and out-of-distribution samples. As a practical example, consider some given tabular dataset with personal information, for example containing the features *age* and *number of children*. In both the sampling and KernelSHAP approach, one could sample and evaluate a sample where *age* is 5 and *number of children* is 10, which is obviously impossible. Note that this problem also occurs for other data modalities, including time-series data. In practice, however, these methods

are widely used in explainable machine learning pipelines as they are fast, simple and model-agnostic.

5.2. TreeSHAP

We established that one can accomplish the task of estimating Shapley values regardless of the model type by using Monte Carlo sampling techniques or the popular KernelSHAP. However, there are also methods of estimating Shapley values that are not *model-agnostic* and can take advantage of the architecture and properties of specific model families. By assuming specific models, such as tree-based models, Shapley values can be calculated much more efficiently.

TreeSHAP [Lun+20] is a tree-specific method which is able to compute Shapley values in polynomial instead of exponential time. We will explain TreeSHAP via a reformulation based on [LP22] with a similar notation. Assume a trained regression tree $T = (\mathcal{V}, \mathcal{E})$. Recall that a decision tree can be expressed as a sum over all paths from root note to leafs. Let $P : \mathbb{R}^n \to \mathbb{R}$ be a path mapping some input $\boldsymbol{x} \in \mathbb{R}^n$ to a leaf prediction $v_l \in \mathbb{R}$ if $\boldsymbol{x}$ lands in that leaf and 0 otherwise. Also, let $\mathcal{Z}$ be the set of all paths that form a tree. Formally, let $h_P(\boldsymbol{x}) = v_l \prod_{e \in P} R_e(\boldsymbol{x})$ and

$$R_e(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ traverses } e \\ 0 & \text{otherwise.} \end{cases}$$

We can then define the prediction of a tree as

$$f(\boldsymbol{x}) = \sum_{P \in \mathcal{Z}} h_P(\boldsymbol{x}).$$

Recall from Eq. (5.1) that the value function for some feature-subset $\mathcal{S}$ is often expressed as the following expectation:

$$v(\mathcal{S}, \boldsymbol{x}) = \mathop{\mathbb{E}}_{\boldsymbol{X}}[f(\boldsymbol{X} | \boldsymbol{X}_\mathcal{S} = \boldsymbol{x}_\mathcal{S})].$$

To calculate one of the entries in the expectation the authors of TreeSHAP propose the *interventional approach*, i.e., they define for the data point which is to be explained $\boldsymbol{x}$, one sample of the background dataset $\boldsymbol{z} \in \boldsymbol{X}$ and some feature subset $\mathcal{S}$ the following:

$$\boldsymbol{r}^{\boldsymbol{z}}_\mathcal{S}(\boldsymbol{x}) := \begin{cases} x_i & \text{if } i \in \mathcal{S} \\ z_i & \text{otherwise.} \end{cases}$$

The background data point is used to substitute "missing" features, i.e., features in $\overline{\mathcal{S}}$. We can reformulate the Shapley value formula for one background data point and one path h_P as

$$\phi_i(h_P, \boldsymbol{x}, \boldsymbol{z}) = \sum_{\mathcal{S} \subseteq \mathcal{N} \setminus \{i\}} w(S, N)(h_P(\boldsymbol{r}^{\boldsymbol{z}}_{\mathcal{S} \cup \{i\}}(\boldsymbol{x})) - h_P(\boldsymbol{r}^{\boldsymbol{z}}_\mathcal{S}(\boldsymbol{x}))). \tag{5.3}$$

Next, we know that edges on the path which are traversed by neither $\boldsymbol{x}$ nor $\boldsymbol{z}$ will result in no attribution:

Proposition 5.1 (Laberge et al. [LP22]). If a path P contains an edge not traversed by $\boldsymbol{x}$ or $\boldsymbol{z}$ then $\phi_i(h_P, \boldsymbol{x}, \boldsymbol{z}) = 0 \; \forall i \in \mathcal{N}$.

Intuitively, this is clear from the definition of h_P as a product over binary values, which is 0 whenever at least one edge is not traversed. For a more thorough proof we refer the reader to Laberge et al. [LP22]. Furthermore, let $\mathcal{S}_{\boldsymbol{x}}, \mathcal{S}_{\boldsymbol{z}}$ be the sets of features used only *only* by $\boldsymbol{x}$ and $\boldsymbol{z}$, respectively, and let $\mathcal{S}_{\boldsymbol{xz}} := \mathcal{S}_{\boldsymbol{x}} \cup \mathcal{S}_{\boldsymbol{z}}$.

Proposition 5.2 (Laberge et al. [LP22]). If $\mathcal{S}_{\boldsymbol{x}} \cap \mathcal{S}_{\boldsymbol{z}} \neq \emptyset$ then $\phi_i(h_P, \boldsymbol{x}, \boldsymbol{z}) = 0 \; \forall i \in \mathcal{N}$.

Again, we refer the reader to Laberge et al.[LP22] for a thorough proof. With both propositions established, the authors are able to show that the Shapley value in Eq. (5.3) can be computed as

$$\phi_i(h_P, \boldsymbol{x}, \boldsymbol{z}) = \begin{cases} w(|\mathcal{S}_{\boldsymbol{x}}| - 1, |\mathcal{S}_{\boldsymbol{xz}}|)v_l & \text{if } i \in \mathcal{S}_{\boldsymbol{x}} \\ -w(|\mathcal{S}_{\boldsymbol{x}}|, |\mathcal{S}_{\boldsymbol{xz}}|)v_l & \text{if } i \in \mathcal{S}_{\boldsymbol{z}} \\ 0 & \text{otherwise.} \end{cases}$$

By efficiently traversing the tree once to compute all sets $\mathcal{S}_{\boldsymbol{x}}, \mathcal{S}_{\boldsymbol{z}}$ and precomputing the weights w one can show that the total runtime to explain one data point $\boldsymbol{x}$ is given by $\mathcal{O}(LDN)$ where L is the number of leafs, D is the depth of the tree and N is the number of background samples. In the literature, it is common to set the background data set to be the training set of the tree. Since there can be at most as many leafs in the tree as there were training samples, we can assume $N = L$. This leads to the total runtime $\mathcal{O}(DN^2)$ to explain one data point.

Finally, let us discuss the estimation of Shapley values with TreeSHAP for tree ensembles. Specifically, we will discuss Random Forests and Gradient Boosting Trees (see Section 2.4.2). Due to the Axiom of Linearity (Axiom 5.3) from Section 5.1 we know that the Shapley value of two games which are combined as linear combinations is equivalent to linearly combining the individual Shapley values. Since RFs (in the case of regression and forecasting) are an ensemble of trees where each tree has equal weight it sufficies to compute the Shapley values for each tree in the ensemble and combining the resulting Shapley values. In terms of computational overhead, if we assume M trees in the RF this results in a total runtime of $\mathcal{O}(MDN^2)$. Additionally, the tree predictions are independent of each other, which means that each Shapley value could be calculated in parallel for more efficient estimation practice. This is not possible for GBT as this is a sequential model. However, the application of the Axiom of Linearity still holds, meaning that we can compute Shapley values for GBTs in the same runtime of $\mathcal{O}(MDN^2)$.

5.3. TreeSHAP Model Selection (TSMS)

Next, we present our approach, TreeSHAP Model Selection (TSMS). As mentioned earlier, we take the approach of RoC creation as a basis for this work.

Model pool In terms of the pool $\mathcal{P}$ of forecasters we used three different models, namely Decision Trees, Random Forests and Gradient Boosting Trees. This is due to the fact that they are i) prevalent tree-based methods in the literature and ii) we can compute Shapley values in polynomial runtime, as mentioned in the previous section.

Extension of PGSMs Consider a forecaster $f \in \mathcal{P}$ and the current window $\boldsymbol{x}_t$. Since we are again interested in computing attributions based on predictive performance we compute the Shapley values not w.r.t. the model prediction but w.r.t. the model loss. This is easily done by overwriting the leaf predictions v_i for each leaf i by the computed error for the current window $\boldsymbol{x}_t$. Formally, if X_{t+1} is the correct value at time $t+1$ and $\boldsymbol{x}_t$ is the window of last known value we will set all leaf node v_i to

$$v_i' := (v_i - X_{t+1})^2.$$

Afterwards, we can compute the Shapley value using TreeSHAP as described earlier, resulting in an attribution $\boldsymbol{\phi}_t$. To get a measure of saliency, we need to invert the Shapley values, so that large positive values now indicate a large reduction of loss and vice versa. This is due to the Axiom of Efficiency where the loss is strictly decomposed into the sum of Shapley values (plus the so-called baseline, which in our case is the expected loss each tree over all data points). To compute the most salient subsequences, we apply thresholding with a positive constant τ to $-\boldsymbol{\phi}_t$, setting every explanation below the threshold to zero, similar to Section 3.4 (visualized in Fig. 3.1).

The remaining methodology is identical to Chapter 3. That is: We compute RoCs for each model in $\mathcal{P}$, monitor for concept drifts during inference and enrich the RoCs whenever we detect such a drift. To choose a forecaster at time t we query the RoCs for the closest pattern to the current input in terms of DTW distance.

5.4. Computing Feature intervals

We can utilize the tree-based model structure to gain further insight into the model prediction process. Specifically, we can compute intervals around lagged values for which a tree (or a tree ensemble) will have the same prediction. This can be used to inspect the robustness of the model: Larger intervals will indicate that the lagged value is less relevant since the feature space partition is large and vice versa. For example, small variations around a particular lagged value due to noise (in the case of forecasting) will lead to the same leaf node and thus the same prediction, making trees robust to noise. To quantify

the robustness, we will present a simple algorithm to quickly derive these intervals for individual trees, which we subsequently extend to tree ensembles.

Let $\boldsymbol{x} \in \mathbb{R}^L$ be the current window of the last L observed values. Additionally, let $T = (\mathcal{V}, \mathcal{E})$ be a trained tree and let P be the path that $\boldsymbol{x}$ takes through T. Recall from Section 2.4.2 that a Decision tree partitions the input space into regions, where the prediction of each region is given by the value of the corresponding leaf node at the end of the path P. Thus, the robustness interval of feature j for $\boldsymbol{x}$ is given by the start and end point of the corresponding region of $\boldsymbol{x}$ w.r.t. the feature dimension j. The algorithm to compute the intervals (as shown in Algorithm 5.1) starts by initializing the interval boundaries for each feature to positive and negative infinity (as we do not know anything about their start and end points a priori). Let $\iota(v)$ be the feature index on which node v splits and let τ_v be the corresponding threshold. For each edge $(a, b) \in P$ in the path taken by $\boldsymbol{x}$ we note the feature ($i = \iota(a)$) that is split on (and terminate the algorithm if we are at the leaf node). Then, if we continue down the left child we add the threshold τ_a to the set of upper limits (since we verified that x_i has to be lower or equal to τ_a). Conversely, we add τ_a to the set of lower limits if $x_i > \tau_a$. Lastly, after traversal of the path is finished, we compute the empirical interval $[I_j^{\text{lb}}, I_j^{\text{ub}}]$ with $I_j^{\text{lb}} = \min(\mathcal{I}_j^{\text{ub}}), I_j^{\text{ub}} = \min(\mathcal{I}_j^{\text{ub}})$ for each feature j. This is possible since

$$l \leq x_j \quad \forall l \in \mathcal{I}_j^{\text{lb}} \Rightarrow \max(\mathcal{I}_j^{\text{lb}}) \leq x_j, \qquad x_j \leq u \quad \forall u \in \mathcal{I}_j^{\text{ub}} \Rightarrow x_j \leq \min(\mathcal{I}_j^{\text{ub}}).$$

We also know that every other data point $\boldsymbol{x}' \in \mathbb{R}^L$ with $x'_j \in [I_j^{\text{lb}}, I_j^{\text{ub}}]\ \forall j \in [L]$ will land in the same leaf node and thus have the same prediction. It follows that any noise added to any feature within the defined interval will not lead to a change in prediction, meaning we can use the size of the interval as a robustness measure.

Algorithm 5.1 Compute feature intervals

Initialize $\mathcal{I}_j^{\text{ub}} = \{\infty\}, \mathcal{I}_j^{\text{lb}} = \{-\infty\} \quad \forall j \in [L]$
for $(a, b) \in P$ **do**
  if isLeaf(a) **then**
    break
  end if
  $j = \iota(a)$
  if $x_j \leq \tau_a$ **then**
    $\mathcal{I}_j^{\text{ub}} = \mathcal{I}_j^{\text{ub}} \cup \{\tau_a\}$
  else
    $\mathcal{I}_j^{\text{lb}} = \mathcal{I}_j^{\text{lb}} \cup \{\tau_a\}$
  end if
end for
$I_j^{\text{lb}} = \max(\mathcal{I}_j^{\text{lb}}), I_j^{\text{ub}} = \min(\mathcal{I}_j^{\text{ub}}) \quad \forall j \in [L]$

Next, we show that this approach can be easily transferred to tree ensembles. Consider that we have run Algorithm 5.1 on T different trees for the same input $\boldsymbol{x}$, where we denote

with $I_j^{\text{lb},i}$ the lower bound for feature j computed on tree $i \in [T]$. Thus, let the sets of all lower and upper bounds computed be defined as

$$\boldsymbol{\mathcal{I}}_j^{\text{lb}} := \left\{ I_j^{\text{lb},i} \right\}_{i \in [T]}, \quad \boldsymbol{\mathcal{I}}_j^{\text{ub}} := \left\{ I_j^{\text{ub},i} \right\}_{i \in [T]}.$$

It is easy to see that we can devise the lower and upper bounds for the entire ensemble by computing

$$\boldsymbol{I}_j^{\text{lb}} = \max(\boldsymbol{\mathcal{I}}_j^{\text{lb}}), \quad \boldsymbol{I}_j^{\text{ub}} = \min(\boldsymbol{\mathcal{I}}_j^{\text{ub}})$$

since the trees in the ensemble cannot disagree about the feature boundaries. Thus, as seen before, we have that

$$l \leq x_j \quad \forall l \in \boldsymbol{\mathcal{I}}_j^{\text{lb}} \Rightarrow \max(\boldsymbol{\mathcal{I}}_j^{\text{lb}}) \leq x_j, \qquad x_j \leq u \quad \forall u \in \boldsymbol{\mathcal{I}}_j^{\text{ub}} \Rightarrow x_j \leq \min(\boldsymbol{\mathcal{I}}_j^{\text{ub}}).$$

We will also address the runtime of computing these intervals. For each tree, we traverse the tree once along the path P, which can be done in D (where D is the maximum depth of the tree). While we introduced $\mathcal{I}^{\text{lb}}$ and $\mathcal{I}^{\text{ub}}$ as sets for which we compute min and max values for notational and conceptual simplicity, we want to note that we can keep track of the smallest and largest thresholds τ in constant time using two arrays. Thus, computing the intervals for one tree can be implemented in $\mathcal{O}(D)$, which leads to a runtime of $\mathcal{O}(TD)$ for an ensemble of T trees. Notice that the calculation for each tree can be done in parallel, leading to a significant speedup in practice.

5.5. Experiments

Next, we present the experiments conducted to validate our proposed approach TSMS, as outlined in the previous sections, as well as to investigate how crucial individual components of the method are. To do that, we aim to answer the following research questions:

- **Q1:** How does TSMS perform against state-of-the-art model selection methods for tree-based models and other relevant baselines?
- **Q2:** What is the impact of the concept drift adaptation in terms of performance and computational resources, compared to both static pre-computation and blind periodic recreation of the RoCs?
- **Q3:** How can TSMS in conjunction with tree-based models serve explainability for time-series forecasting?

5.5.1. Experimental Setup

We utilize 1032 univariate datasets from various application domains, including financial, weather, and synthetic data. These datasets are provided by the Monash Forecasting Repository [God+21]. We process each time-series $\boldsymbol{X}$ by using the first 50% for training

Name	Nr. of time series	Min. length
M4	548	251
Tourism	100	252
Australian Electricity Demands	5	500
Dominick	100	251
Bitcoin	16	500
Pedestrian counts	63	500
KDD Cup	100	500
Weather	100	500
Total	1032	

Table 5.1.: Breakdown of the used univariate time-series from the Monash Forecasting Repository [God+21].

Model family	Hyperparameters
Decision Tree	$d_{max} \in \{4, 8, 16\}$
Random Forest	$d_{max} \in \{2, 4, 6\}, n_{trees} \in \{16, 32, 64\}$
Gradient Boosting Trees	$d_{max} \in \{2, 4, 6\}, n_{trees} \in \{16, 32, 64\}$

Table 5.2.: List of all models in the model selection pool. Through combination of the listed hyperparameters, there are 21 models in the final model pool.

($\boldsymbol{X}^{\text{train}}$), the following 25% for validation ($\boldsymbol{X}^{\text{val}}$), and the remaining 25% for testing ($\boldsymbol{X}^{\text{test}}$). We normalize the entire time-series using mean and standard deviation estimated over the training portion of the time-series. Due to this way of splitting the time-series, we discard series that are shorter than 250 to allow enough training and validation data. Additionally, we took a random subsample of length 500 for series that are longer to limit computation time. A breakdown of the used datasets and characteristics is shown in Table 5.1. The code to reproduce all experiments is available online[8].

We set the amount of lagged values to $L = 15$ to allow for a large context in terms of lags. The single models consist of various parametrizations of Decision Trees, Random Forests, and Gradient Boosting Trees from `scikit-learn`. To create the model pool $\mathcal{P}$, we train a total of 21 tree-based models. Each model is created by varying the maximum tree depth d_{max} and (if available) the number of ensemble estimators n_{trees} (details shown in Table 5.2). Notice that we do not allow the trees to grow arbitrarily deep. That way, we retain the possibility to inspect them, further boosting the interpretability of our approach.

[8] https://github.com/MatthiasJakobs/tsms

5.5.2. TSMS Setup and Baselines

To investigate the impact of drift detection on model performance and runtime, we propose a total of three variants of our method:

- TSMS: Our described method, using drift detection to adapt the RoCs to changes in the time-series
- TSMS-ST: Static variant where drift-detection is disabled. RoCs are created at the beginning of the forecasting process and remain unchanged over time
- TSMS-Per: A periodic, blind update of the RoCs. We chose to update the RoCs automatically 10 times in total over the length of $\boldsymbol{X}^{\text{test}}$ at fixed, equally spaced intervals

Additionally, there are two hyperparameters in our method that we tuned: ω, the size of larger, equal-size windows during the RoC creation process, and τ, the threshold used when refining the data points before addition to an RoC. We conducted a hyperparameter search and found $\omega = 25$ and $\tau = 0.01$ to perform well.

In terms of state-of-the-art methods we compare ourselves against, we utilize the following methods (see Section 3.7 for more information): Exponential Smoothing (`ETS`), `ARIMA`, `KNN-RoC`, `DETS` and `ADE`. Moreover, we utilize two deep learning models, namely `CNN` and `CNN-LSTM`, which are simple architectures trained for comparison between DL and tree-based approaches. Because of our use of tree-based model, we want to highlight that we cannot fairly compare ourselves to OS-PGSM as the model pool has to be different (and we aim to evaluate the performance of the selection and adaption methodology, not the base models).

5.5.3. Performance Comparison Results

Table 5.3 shows a comparison between our method, including its different variations, and the previously mentioned baselines over 1032 time-series. We computed the average rank achieved after measuring the loss using the RMSE score (see Section 2.4.3). As can be seen, the drift-aware variant of TSMS achieves the smallest average rank and outperforms other online model selection baselines such as `DETS`, `ADE`, and `KNN-RoC`. For comparison, we also include the best-performing model from $\mathcal{P}$, denoted as `Best-Single`, and find that it also outperforms the model-selection baselines, indicating that they are not able to predict the performance of this single model on the test data very well. TSMS is also able to outperform `CNN-LSTM` and `CNN` while being more interpretable at the same time due to our restriction to use shallow-depth tree-based models. We also ran a signed Wilcoxon rank test on the wins and losses and indicate the significant wins/losses in parenthesis (significance level $\alpha = 0.05$) (see Section 2.4.3). The results clearly answer research question **Q1**.

Next, we compare TSMS with the two variants, namely the static (TSMS-St) and blind, periodic (TSMS-Per) variants. We observe that an informed and infrequent enrichment of

Method name	Losses	Wins	Average Rank	±
ETS	212 (209)	789 (740)	7.14	4.53
ADE	314 (218)	687 (601)	7.03	3.01
ARIMA	264 (200)	737 (679)	6.46	5.04
KNN-RoC	368 (221)	633 (578)	6.36	2.71
DETS	360 (295)	641 (596)	6.31	2.61
CNN	319 (314)	682 (638)	5.49	3.35
CNN-LSTM	399 (378)	602 (518)	4.85	2.68
TSMS-St	640 (427)	361 (279)	4.64	2.88
Best-Single	419 (414)	582 (502)	4.36	2.71
TSMS-Per	667 (482)	334 (240)	4.19	2.60
TSMS	- (-)	- (-)	4.13	2.49

Table 5.3.: Comparison (in terms of average rank achieved over 1032 datasets) between our method and the baselines. Best-Single is the model from $\mathcal{P}$ that performed best over all datasets.

Method	TSMS	TSMS-St	TSMS-Per
Runtime [s]	1.39 ± 1.28	0.41 ± 0.31	1.71 ± 1.28

Table 5.4.: Mean runtime (in seconds) plus/minus one standard deviation, measured over all datasets.

the RoCs leads to better results in terms of predictive performance. We observe that the periodic approach is only slightly better in terms of average ranking than the static approach, indicating that a blind recreation of RoCs is in most cases not worth the additional computational overhead. These findings are in line with the findings when using CNNs in Section 3.7 and Section 4.4 and thus are not only applicable to deep learning model pools. We additionally compare the runtime of each variant over all datasets and report the average runtime, and its standard deviation, in Table 5.4. Unsurprisingly, TSMS-St is, on average, the fastest method since the RoC recreation procedure takes up most of the runtime. The adaptive variant TSMS falls in between the static and periodic variants, suggesting that the small trade in runtimes is worthwhile for improving predictive performance. With these insights, we provide answers for research question **Q2**.

5.5.4. Explainability

In this part, we answer research question **Q3** by showing how TSMS supports explainability for the forecasting task:

- **A1**: Which input time-series parts, i.e., lagged values, are more relevant for the prediction, and how does this relevance evolve over time?
- **A2**: Why is a model i chosen to forecast the series at time t?

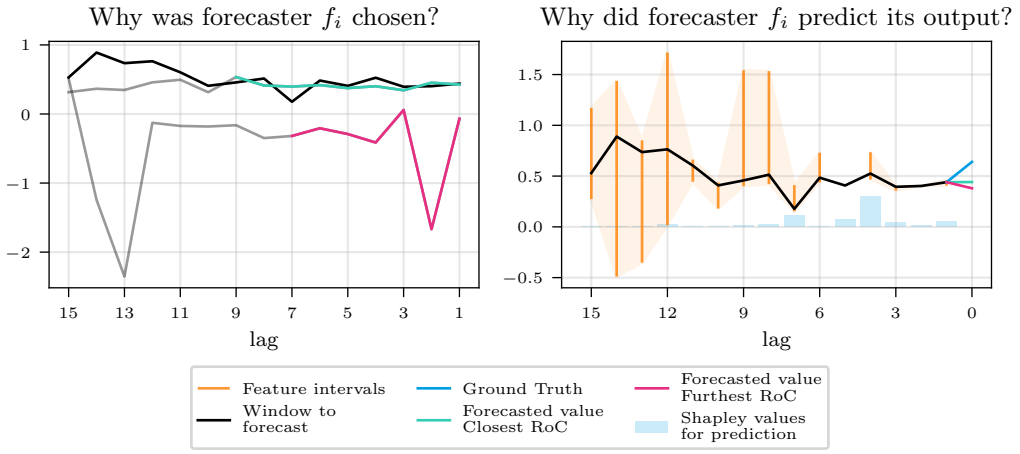

Figure 5.2.: Visualization of **A2** and **A3** by investigating the closest and furthest RoC members, in addition to Shapley values and feature decision boundaries.

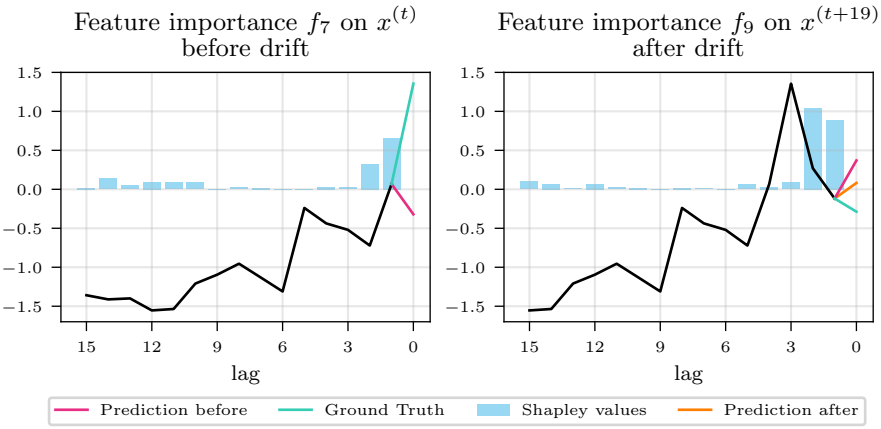

Figure 5.3.: Comparison of Shapley values for prediction before and after concept drift where the chosen forecaster changes.

- **A3**: Why a specific predicted value $\hat{X}^{(t+1)}$ is output by the selected model at time instant t?
- **A4**: What is the impact of concept drift occurrence on the RoCs?

First, in the right half of Fig. 5.2, we can see the Shapley values of the 15-time-lagged values (in blue) w.r.t. the prediction. Notice that, while we use the Shapley values of the empirical loss for RoC creation we can also compute the Shapley values w.r.t. the prediction to get further insight into the prediction process. For the prediction, the higher the bars (in blue), the more important the lag information fed to the forecasting model is.

As can be seen, many lags among the 15 lags seem to be irrelevant to predict the next value of the series. Surprisingly, this information is sparse and not strongly condensed around the most recent lags, in opposite to the assumption made by most of the traditional state-of-the-art models, including the Box-Jenkins ARIMA family of models (see Section 2.3.2). This can be explained by the fact that these models assume the stationarity of the time-series-generating process and the absence of concept drifts. So, once analyzed, lag information and contribution in these models are kept fixed and restricted to the most three to four recent lagged observations [Saa+20]. However, this is not the case for many real-world scenarios. As can be seen in Fig. 5.3, the second most recent lag (lag 2) that used to have very low importance for the prediction before the occurrence of concept drift has become the most relevant one after the detection of drift. This shift in the importance of the input highlights the necessity of either triggering an update of the model using retraining or switching to another model that better handles the new dependence structure in the data after the occurrence of concept drift. Fig. 5.3 shows that our framework ensures this update by switching to another model after drift detection, which predicts the true value of the series better. The prediction marked in red would be the forecast value if we stuck to the old model after drift detection, while the prediction marked in orange is the value output by the model selected by TSMS, which is closer to the true target value marked in green.

Another potential insight provided by the tree-based models in relation to understanding the contribution of the lag information in the input time-series to the prediction is provided by the paths taken in each tree. These paths enable the construction of boundaries around each lag via each node split criterion (as detailed in Section 5.4). In Fig. 5.2 (right-hand side) we visualize the intervals in orange. Changing the input value anywhere inside the orange intervals will have no effect on the models' prediction, which helps in understanding the robustness of the tree-based models. Also, notice that the intervals are significantly larger for higher lag values, i.e., for time-steps which are further away in time from the prediction. We can deduce that more recent time-steps are much more influential in the prediction, together with the values at lag 10 and 11, suggesting a slight periodicity that was used for prediction. Moreover, observe that the intervals are not symmetric, suggesting that (in the case of lags 8 and 9, for example) outliers towards higher values are less critical in comparison to outliers towards smaller values. This knowledge is also of great help to forecasting practitioners as it helps them in conducting input change sensitivity analysis which is required, for example, in financial forecasting applications. This addresses the explainability aspect raised in Question **A1**.

As previously explained, the reason for selecting forecaster i at time t is because its RoC contains a subseries that is the most similar to the current input pattern $\boldsymbol{x}_t$ in terms of DTW distance. In Fig. 5.2 (left half), we show the current input pattern $\boldsymbol{x}_t$ in black, the closest RoC marked in green and the least close RoC entry over all forecasters in red. Notice that we chose to visualize the entire window in gray and only highlight the RoC member in color to get a better understanding about where in the window this salient subsequence is found. Due to the obviously smaller distance of the green RoC entry to the input pattern the corresponding forecaster is chosen for prediction. The predictions

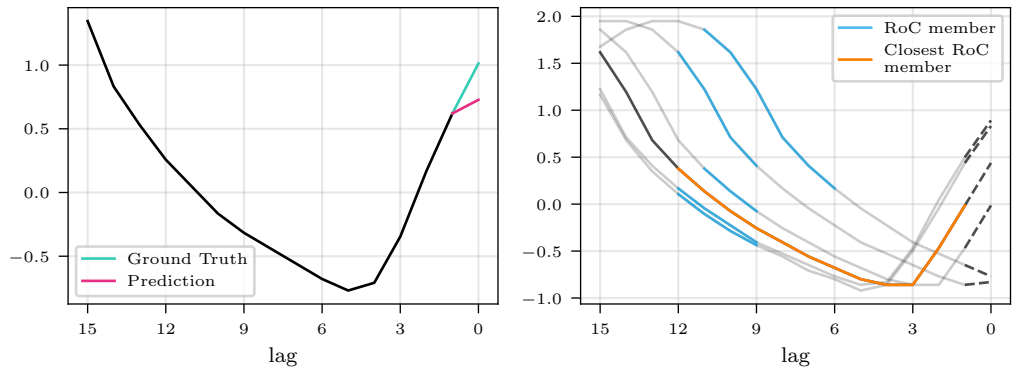

Figure 5.4.: Visualization of RoC $\mathcal{R}_{17}$ (right side) where the closest entry to the current pattern (left side, black) is shown in orange.

of both the forecaster of the green and red RoC entries is shown on the right-hand side. Examining these predictions validates our RoC-based selection as showing competence on patterns similar to the input pattern in question reflects the readiness of the model to process this pattern and output a prediction based on it. Hence, the large deviation between the RoC of the furthest model and the current pattern in Fig. 5.2 in the left half is reflected in its prediction shown in right half. This answers the explainability aspect raised in Question **A2**.

Next, we provide some insights into how our method can help us to anticipate the model's output. This is one of the main important aspects of explainability as stated by Kim et al. [KKK16], who define explainability as the degree to which a human can consistently predict the model's result. In Fig. 5.4, we visualize the RoC of the selected forecaster f_{17}. Additionally, we show the true next values in the validation set with dotted lines. A clear similarity between the output of the model (left, red) and the values that are subsequent to the RoCs sequences (dotted, right) can be observed. The range of the subsequent values helps us in estimating the expected forecast value by the selected model on average. Comparing the predicted value by the model with this estimated average can be considered a debugging tool that helps us in detecting abnormal behavior of the model or check for significant changes in the data. This addresses the explainability aspect raised in Question **A3**.

To get a better understanding of the impact of drift detection, we investigate two scenarios. First, shown in Fig. 5.5, is a scenario where the drift detection was triggered but where the same model that would have been chosen without a drift was chosen regardless. Nevertheless, we can see between the middle and right side that the closest RoC member did change. The new closest RoC member (shown in orange) much more resembles the input in comparison to the previously closest RoC member (shown in red), indicating that the enrichment of the RoCs after the detected drift did successfully adapt the RoCs to the time-changing data.

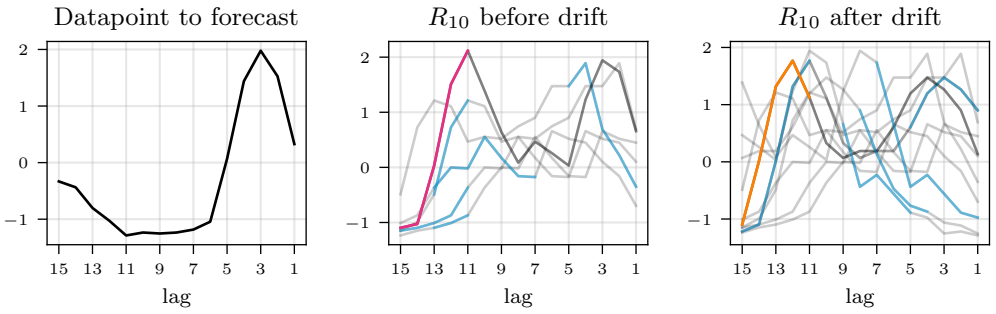

Figure 5.5.: Comparison of $\mathcal{R}_{10}$ before and after a drift. Closest RoC member shown in red and orange, respectively.

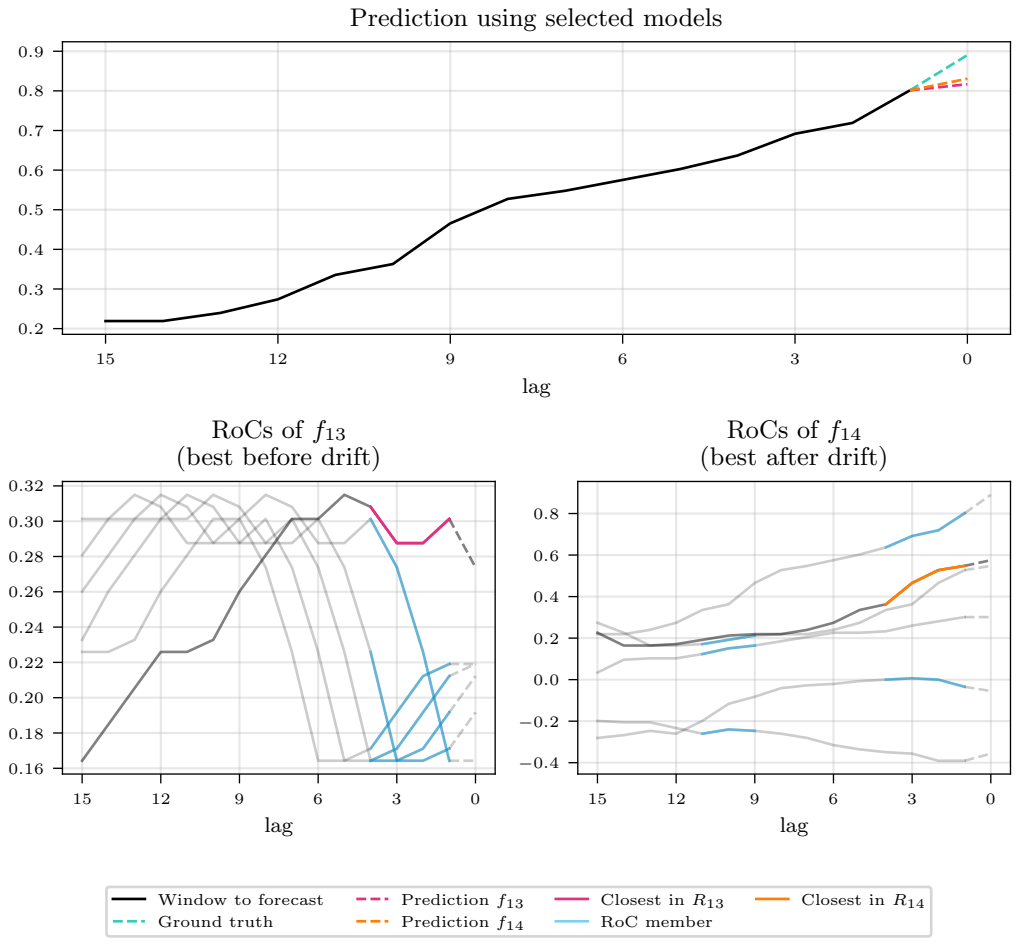

Figure 5.6.: Comparison of $\mathcal{R}_{13}$ and $\mathcal{R}_{14}$ before and after a drift. Closest RoC member shown in orange and red, respectively.

Lastly, we consider a scenario where the chosen model did, in fact, change after a drift was triggered. As can be seen in Fig. 5.6, without enrichment of the RoCs, f_{13} would have been chosen over f_{14} in this case. However, when investigating the prediction (top), the prediction of f_{16} (yellow) is closer to the ground truth value (green) compared to the prediction of f_{13} (red). Additionally, by investigating each model's RoC (lower left and lower right), it becomes clear that $\mathcal{R}_{14}$ contains a member that resembles the input a lot more in comparison to the closest member in $\mathcal{R}_{13}$. These observations highlight the usefulness of the drift-aware adaptation of the RoCs over time. This answers the research question **A4**.

5.6. Concluding Remarks

In this chapter we presented TSMS, a tree-based online model selection method for time-series forecasting. By limiting the pool of models to interpretable tree-based models we were able to increase the overall interpretability over the previously presented CNN-based approaches in Chapter 3 and Chapter 4. The empirical results indicate that TSMS performs very well in comparison to the tested baselines while simultaneously allowing insight into the decision-making process. We showcased that TSMS is able to adapt well to changes in the underlying time-series data without the need to retrain every model in the pool. By using tree-based models we can increase interpretability and explainability in multiple ways, such as by investigating the robustness to noise, providing a foundation for further uses of trees for safety-critical time-series forecasting settings.

Part III.

Interpretability in Forecasting and Predictive Maintenance

6. Almost Always Linear Forecasting

In Part II we presented approaches for Online Model Selection and Ensembling that give practitioners more insights by incorporating explainability methods such as Grad-CAM and Shapley values. In addition, we saw in Chapter 5 that the use interpretable base models in the pool leads to additional insights into the prediction process which were not possible due to the methodological limitation to CNNs in Chapter 3 and Chapter 4. We also saw in the last chapter that tree-based models, which are inherently interpretable, can compete with state-of-the-art LSTM and CNN-based models when used in the model selection framework.

Based on these findings we want to investigate in this chapter the feasibility of other interpretable time-series forecasting methods when utilizing the online model selection approach. As mentioned in Section 2.5.2 there is an often stated trade-off between interpretability and predictive performance: The more interpretable a model is the smaller its predictive performance tends to be. However, in the time-series forecasting literature, this conventional wisdom is often found not to be the case. First, as mentioned before, we were able to achieve state-of-the-art results with tree-based forecasters in Chapter 5, even when using one fixed model (which we denoted with `Best-Single` in Table 5.3). Additionally, in the M4 forecasting competition [MSA20], the organizers found that traditional, simple and interpretable methods such as ARIMA and Exponential Smoothing performed surprisingly well, and that by combining interpretable methods with high performing Deep Learning methods, an overall increase in predictive performance can be observed [Smy20]. Specifically, the auhors in [Smy20]combine Exponential Smoothing with LSTM Neural Networks to improve prediction quality, winning the competition in the process. This is an example of so-called Hybrid Model Selection [HK19], where many approaches first apply a linear model to the time-series, followed by a non-linear model on the residuals. For example, in [Zha03] the authors first predict with ARIMA, followed by a simple Neural Network on the (assumed to be) non-linear residuals. However, the forecasting process on the residuals is still done by uninterpretable DL methods, which makes them hard to deploy in safety-critical applications due to their opaque nature.

In this chapter we will investigate a different way of combining complex DL methods with high predictive performance models and small, domain-specific interpretable methods. Specifically, we will investigate the hypothesis that interpretable models are performing "good enough" for most predictions. By employing online model selection to learn when this is not the case we can resort back to DL methods. That way, we are able to achieve high predictive performance while also utilizing an interpretable model most of the time. We thus restrict the pool of forecasters to two methods, one of which (the interpretable

forecaster) we want to choose as often as possible. Online Model Selection with similar constraints have been proposed in the literature before. For example, in [BC23; DPP22; Bus24], the authors are concerned with reducing energy consumption by considering smaller models (in their case, Decision Trees) that are quick and energy efficient to evaluate over larger methods (in their case, Convolutional Neural Networks), that need to be run on special, high power hardware. If a Decision Tree predicts a class on which it is deemed to be not good enough the Convolutional Neural Network is queried instead, increasing the energy consumption. However, since lowering energy consumption is a focus of their work the models are only evaluated if they are chosen by the model selection procedure. Since we are not concerned with preserving energy in this chapter we will also incorporate both model predictions during the selection approach. Our proposed method is a generic framework which is easily adaptable to different application scenarios. While we showcase the validity of our method with an autoregressive AR(p) model and popular DL forecasting models we want to stress that we do not make assumptions about the specific model families.

This chapter is based on the publication [JL25]. The author of this thesis devised the main idea of the paper, including the original methodology section. He additionally implemented the code for the experiments and wrote the majority of the original paper.

This chapter is structured as follows. We start by formalizing the scenario of online model selection with a preference towards one model in Section 6.1, where we also derive a closed-form solution to the described optimization problem. Next, we present our approach to estimate the solution on unseen data in Section 6.2. We will conduct extensive experiments in Section 6.3 where we will investigate the setting of perfect selection, as well as how close we can approximate this selection using various learners. We also compare our proposed method against state-of-the-art online model selection methods in terms of predictive performance and overall interpretability. Lastly the chapter will conclude with some final remarks.

6.1. Optimal Model Selection

We will start by describing the problem in a more formal manner by comparing it to generic Online Model Selection from Section 3.1. Let $\mathcal{P} = \{f, g\}$ be the pool of two trained forecasters, mapping from $\mathbb{R}^L$ to $\mathbb{R}$. As already discussed in previous sections we are interested in predicting, at time t, the next value y_t given a vector of the previous L lagged values, denoted as $\boldsymbol{x}_t$. Specifically, our proposed selection strategy should do two things: First, we want to minimize the overall prediction error over some test period of length T. Crucially, we want to choose one of the forecasters in $\mathcal{P}$ (in our case f) as often as possible. We will proceed by referring to f as an *interpretable* model and denote with g a more complex, DL method. However, as already mentioned, our framework is flexible and can be used for other scenarios where choosing one model over another might be

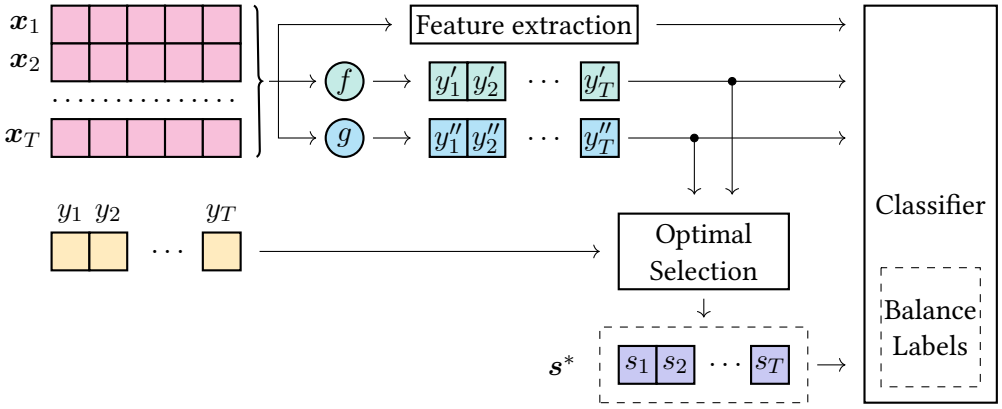

Figure 6.1.: Schematic overview of our proposed method AALF: Given a set of windowed time-series $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$ with the corresponding true values $\{y_1, \ldots, y_T\}$ we compute both models predictions $f(\boldsymbol{x}_t) := y'_t$ and $g(\boldsymbol{x}_t) := y''_t$, as well as some additional features. The predictions, along with the label, are used to predict the optimal model selection $\boldsymbol{s}^*$, which we use as labels to estimate a classifier.

beneficial. Formally, the observed squared error if we choose between f and g over T is given by

$$\mathcal{L}(\boldsymbol{s}) = \sum_{t=1}^{T} \left(y_t - s_t f(\boldsymbol{x}_t) - (1 - s_t) g(\boldsymbol{x}_t)\right)^2, \quad s_t \in \{0, 1\}$$

where $\boldsymbol{s} = (s_1, \ldots, s_T)^\intercal$. That means that f is chosen if $s_t = 1$ (and g if $s_t = 0$). In order to choose f more often than g we introduce a constraint to select f at least B times out of T. This leads to the following constraint optimization problem:

$$\boldsymbol{s}^* = \arg\min_{\boldsymbol{s}} \ \mathcal{L}(\boldsymbol{s}) \quad \text{s.t.} \quad \boldsymbol{s} \in \{0, 1\}^T \wedge \|\boldsymbol{s}\|_1 \geq B. \tag{6.1}$$

Essentially, B is what we will refer to as an *interpretability* constraint which requires that f is chosen a certain number of times. Next, we show that we can compute the optimal solution to this constraint optimization problem in a closed form:

Proposition 6.1. The optimal solution to Eq. (6.1) is the vector $\boldsymbol{s}^* = (s_1^*, \ldots, s_T^*)^\intercal$ with elements

$$s_t^* = \begin{cases} 1 & \text{if } \ell(t) \leq \max(0, \ell(\pi(B))) \\ 0 & \text{otherwise} \end{cases}$$

where $\ell(t) := (f(\boldsymbol{x}_t) - y_t)^2 - (g(\boldsymbol{x}_t) - y_t)^2$ and $\pi : [T] \to [T]$ is a permutation satisfying $\ell(\pi(t)) \leq \ell(\pi(t+1)) \; \forall t \in [T-1]$.

Proof. We start by reformulating the objective $\mathcal{L}(\boldsymbol{s})$ as

$$\begin{aligned}\mathcal{L}(\boldsymbol{s}) &= \sum_{t=1}^{T} (\underbrace{y_t - g(\boldsymbol{x}_t)}_{=:c_t} + s_t \underbrace{(g(\boldsymbol{x}_t) - f(\boldsymbol{x}_t))}_{=:\delta_t})^2 \\ &= \sum_{t=1}^{T} c_t^2 + 2c_t s_t \delta_t + s_t^2 \delta_t^2 \\ &= \left(\sum_{t=1}^{T} (y_t - g(\boldsymbol{x}_t))^2 \right) + \left(\sum_{t=1}^{T} s_t (2c_t \delta_t + \delta_t^2) \right)\end{aligned}$$

For the reformulation, notice that $s_t^2 = s_t$ since $s_t \in \{0, 1\}$. Moreover, notice that each summand of the second sum is the difference of squared errors between f and g at time t:

$$\begin{aligned}2c_t\delta_t + \delta_t^2 &= 2(y_t - g(\boldsymbol{x}_t))(g(\boldsymbol{x}_t) - f(\boldsymbol{x}_t)) + (g(\boldsymbol{x}_t) - f(\boldsymbol{x}_t))^2 \\ &= f(\boldsymbol{x}_t)^2 - 2f(\boldsymbol{x}_t)y_t + y_t^2 - (y_t^2 - 2g(\boldsymbol{x}_t)y_t + g(\boldsymbol{x}_t)^2) \\ &= (f(\boldsymbol{x}_t) - y_t)^2 - (g(\boldsymbol{x}_t) - y_t)^2 \\ &= \ell(t).\end{aligned}$$

Since the first sum in $\mathcal{L}(\boldsymbol{s})$ is independent of $\boldsymbol{s}$ the optimization problem is equivalent to

$$\boldsymbol{s}^* = \arg\min_{\boldsymbol{s}} \sum_{t=1}^{T} s_t \ell(t) \quad \text{s.t.} \quad \boldsymbol{s} \in \{0, 1\}^T \wedge \|\boldsymbol{s}\|_1 \geq B. \tag{6.2}$$

With this reformulation it is easy to see that the optimal solution (ignoring the constraint $\|\boldsymbol{s}\|_1 \geq B$) is given by choosing all indices t for which $\ell(t) \leq 0$, i.e.,

$$s_t^* = \begin{cases} 1 & \text{if } \ell(t) \leq 0 \\ 0 & \text{otherwise.} \end{cases}$$

In fact, if $\|\boldsymbol{s}^*\|_1 \geq B$ we have found a constraint satisfying solution (and included as many predictions of f as possible) and are done. Otherwise, we know that we already chose $t \in \{\pi(1), \ldots, \pi(N)\}$ with $N = \|\boldsymbol{s}^*\|_1$ and we know that $N < B$. This means that by choosing all t with $\ell(t) \leq \ell(\pi(B))$ we include the negative error differences from $\boldsymbol{s}^*$ as well as the necessary smallest differences to satisfy the constraint. Thus, we can rewrite the optimal solution to

$$s_t^* = \begin{cases} 1 & \text{if } \ell(t) \leq \max(0, \ell(\pi(B))) \\ 0 & \text{otherwise.} \end{cases}$$

□

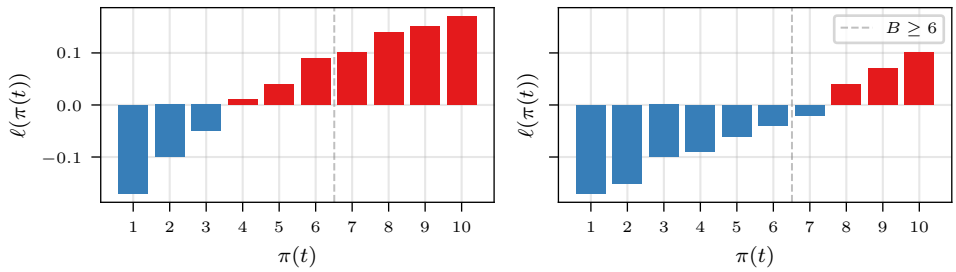

Figure 6.2.: Illustrative example of how many (and which) entries to choose to get the optimal, constraint-satisfying solution. On the left we have to choose the indices $\pi(4), \pi(5)$ and $\pi(6)$ even though their corresponding loss difference $\ell(t)$ is positive to satisfy the constraint $B \geq 6$. On the right side we see that after choosing the first 6 entries we can continue and further decrease the loss, choosing 7 entries in total.

Intuitively, Proposition 6.1 states that the optimal solution is given by choosing f whenever it outperforms g. In addition, we (might) need to choose f even if the total error increases until the constraint is satisfied. Both scenarios are visualized with an example in Fig. 6.2. For both examples we have $B \geq 6$, i.e., we have to choose f at least 6 times. On the left side the optimal solution is given by choosing B exactly 6 times since $\ell(\pi(t)) > 0$ already for $t > 3$. On the right side however we choose f a total of 7 times to get the optimal solution.

6.2. Meta Feature extraction

The optimal selection according to Proposition 6.1 requires that the label is known since the squared error of both models from the pool are necessary during the calculation. However, we obviously do not have access to the labels during inference on unseen data. Thus, we propose to compute $\boldsymbol{s}^*$ on validation data (where all information is available) and use these selections as labels in a supervised learning setting. Specifically, we propose to learn a classifier which selects, for each t, which model to select by learning the optimal selection on validation data using the lagged window $\boldsymbol{x}_t$ combined with additional meta-features.

Again, let $(\boldsymbol{x}_t, y_t)$ be a window of data and label from known validation data. Then, we compute the optimal selection s_t^* using $f(\boldsymbol{x}_t)$, $g(\boldsymbol{x}_t)$ and y_t according to Proposition 6.1. Furthermore, we generate the following meta features to aid in fitting the classifier. $\Delta_p(t) = f(\boldsymbol{x}_t) - g(\boldsymbol{x}_t)$ is the difference in prediction between both models at time t. We can always compute this feature as it is independent of the true prediction (and since we do not consider the computational overhead or additional resource consumption of evaluating both models as mentioned in the beginning of this chapter). $\Delta_e(t) = (f(\boldsymbol{x}_{t-1}) - y_{t-1})^2 - (g(\boldsymbol{x}_{t-1}) - y_{t-1})^2$ is the last known difference in observed errors between the models. This feature helps in modeling the most recent

performance and allows for simple estimates of the expected, current error. Lastly, let $\Sigma(t) = (L^{-1} \sum_{i=1}^{L} x_t^{(i)}, \min(\boldsymbol{x}_t), \max(\boldsymbol{x}_t))^T$ be the empirical mean, min and max values over the current window $\boldsymbol{x}_t$. Combining all of these features, we have the new data point $(\boldsymbol{x}'_t, s_t^*)$ with

$$\boldsymbol{x}'_t = \boldsymbol{x}_t \oplus \Delta_p(t) \oplus \Delta_e(t) \oplus \Sigma(t)$$

where $\oplus$ is the vector concatenation operator. By computing these features over the available validation data range we are able to create a new dataset to train a classifier for selection. Notice that we also include the raw lagged values $\boldsymbol{x}_t$ as additional features.

6.3. Experiments

After describing our general methodology we proceed by investigating the validity of our approach by answering the following three research questions:

- **Q1**: What is the impact of (large) interpretability constraints B w.r.t. predictive performance?
- **Q2**: How accurate can we estimate the optimal selection according to Proposition 6.1 for different values of B?
- **Q3**: How does our method perform in terms of prediction error and interpretable model selection against state-of-the-art online model selection methods?

We utilize 6 univariate time-series datasets of different application domains and with varying characteristics for our experiments (see Table 6.1 for a overview of the datasets and their characteristics). These datasets are provided by the Monash Forecasting Repository [God+21]. We process each time-series by using the first 80% as the training portion ($\boldsymbol{X}^{\text{train}}$), the following 10% for validation ($\boldsymbol{X}^{\text{val}}$) and the final 10% for testing ($\boldsymbol{X}^{\text{test}}$). To only keep meaningful time-series we discard series where all time steps in the training, validation or test data are constant. Since the time-series frequency of the datasets vary we chose to also vary the amount of lagged values accordingly. We chose to use one day ($L = 48$) for the 30 minute datasets, 14 days for daily and 24 hours for the hourly datasets. We set $H = 1$ for all experiments. The code to reproduce all experiments is available online[9].

6.3.1. Training the base models

In order to investigate the impact of our model selection method on predictive performance and interpretability we have to train both f and g. To make f as simple and interpretable as possible we chose an autoregressive model AR (see Section 2.3.2) since it is a very competitive baseline, even for current time-series forecasting methods. Moreover, it is a well-understood and well-studied forecasting method that is fairly interpretable

[9] https://github.com/MatthiasJakobs/aalf/tree/mlj

Name	Datapoints	Length (min/max/avg)	Freq.	L	Type
Aus. Electricity Demand	5	230,734/232,271/231,051.20	30m	48	Energy
NN5 (Daily)	111	790/790/790.00	1d	14	Banking
Weather	2997	1,331/59,040/14,263.00	1d	14	Nature
Pedestrian Counts	66	574/96,423/47,458.39	1h	24	Transport
KDD Cup	255	9,503/10,920/10,895.24	1h	24	Nature
Solar	137	8,760/8,760/8,760.0	1h	24	Energy

Table 6.1.: Properties of the used datasets. All datasets are provided by the Monash Forecasting Repository [God+21]. We resampled the Solar dataset to a frequency of one hour from 10 minutes to decrease computational demand.

for experts since it is a purely linear forecaster. Since `AR` is a local model we trained one model per time-series, more specifically on the training split of each time-series. For g we initially trained three different models. The first one is a simple multi-layer neural network (denoted `FCNN`) with two hidden layers with ReLU activations. The second model is a Convolutional Neural Network (denoted `CNN`) with two layers of convolutions, followed by one prediction layer. Model three is `DeepAR` [SFG19], which is a competitive state-of-the-art time-series forecasting model based on an LSTM architecture. Specifically, it processes the current given window of lagged values element-by-element with a LSTM layer. Afterwards, the resulting hidden vector (see Section 2.4.1 for more details) is used to estimate mean and standard deviation values of a Gaussian distribution. Predictions are computed by sampling from this empirical distribution (or by using the mean value directly). All three models (`FCNN`, `CNN` and `DeepAR`) are *global* forecasters, meaning that we trained one model per dataset (and not per individual time-series). Specifically, they were trained on the combined training of all time-series in a dataset and afterwards evaluated on the validation and test parts of each time-series individually.

We evaluated all models on the test portion of each dataset using two metrics, namely root mean squared error (RMSE) and the symmetric mean absolute percentage error (SMAPE) (see Section 2.4.3). In order to produce one aggregated value of RMSE and SMAPE per dataset we averaged the individual values computed on the test portions of each time-series. We present these results in Table 6.2. Except for the KDD Cup dataset we find that one of the larger, global base models always outperforms the `AR` model, both in terms of RMSE and SMAPE. For the NN5 (Daily) and Solar datasets we see that the CNN performs best in terms or RMSE error, while `DeepAR` performs best (also in terms of RMSE) on the Weather and Pedestrian Counts dataset. Finally, the FCNN outperforms the other models on the Australian Electricity Demand dataset. However, we can see that the SMAPE value distribution is not necessarily correlated with RMSE.

To investigate this difference and to gain more insight into the individual model performances we analyzed how significant the performance differences were using a Critical Difference (CD) diagram (see Section 2.4.3). Each row corresponds to the CD diagram

	AR		FCNN		DeepAR		CNN	
	RMSE	SMAPE	RMSE	SMAPE	RMSE	SMAPE	RMSE	SMAPE
Aus. Electricity Demand	0.086	0.181	**0.063**	0.151	0.069	**0.139**	0.069	0.152
NN5 (Daily)	0.735	0.836	0.688	**0.758**	0.705	0.786	**0.685**	0.762
Weather	0.673	0.859	0.667	0.813	**0.665**	0.811	0.670	**0.801**
Pedestrian Counts	0.287	0.366	0.245	0.312	**0.232**	**0.290**	0.278	0.347
KDD Cup	**0.362**	0.498	0.367	0.492	0.416	**0.489**	0.367	0.493
Solar	0.220	0.310	0.202	0.272	0.250	0.318	**0.193**	**0.264**

Table 6.2.: The averaged root mean squared error (RMSE) and symmetric mean absolute percentage error (SMAPE) values for all base models over all datasets. The average was computed over each time-series per dataset. Notice that we highlighted the model with the smallest error (per dataset and metric) in bold face.

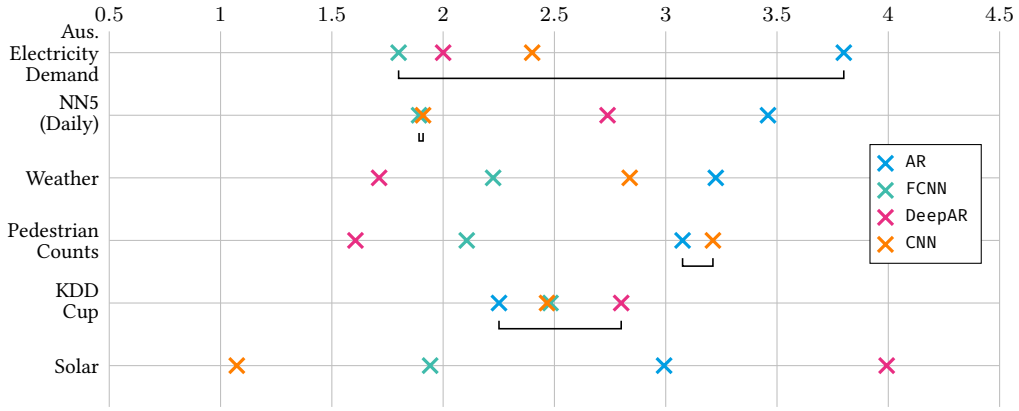

Figure 6.3.: Critical Difference diagrams for all 6 datasets (each row corresponds to one dataset). The x axis shows the average rank of each model over the time-series in each dataset. If two models are not found to have significantly different average ranks based on a Wilcoxon signed-rank test they are connected with a horizontal bar.

over the RMSE values per dataset, with the average model rank (computed over all time-series in the dataset) shown on the x axis. If the Wilcoxon test cannot distinguish two models a horizontal bar connects them, signifying that they are not significantly different enough (with significance level $\alpha = 0.05$).

After the analysis of the base models we now select one of the black-box models as g for each dataset (since our methods assumes a binary selection as outlined in Section 6.1). Specifically, based on the results in Fig. 6.3 and Table 6.2 we choose `DeepAR` for the Weather and Pedestrian Counts datasets, `CNN` for the NN5 (Daily) and Solar dataset as well as `FCNN` for the Australian Electricity Demand and KDD Cup datasets.

6.3.2. Impact of the interpretability constraint

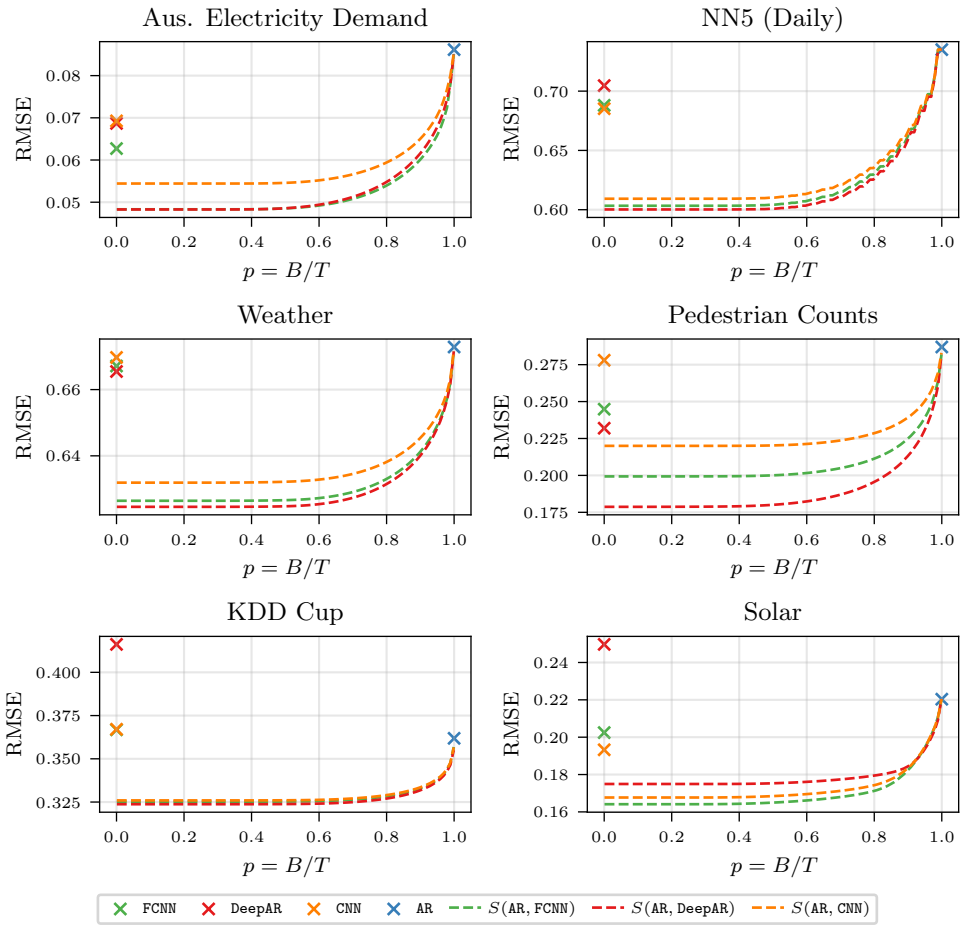

Figure 6.4.: The optimal selection $S(f, g)$ between pairs of models (shown as lines) versus the performance of the individual models. The y axis corresponds to average RMSE over the datasets while the x axis corresponds to how often f is chosen over g. Note that S is computed using ground-truth data and thus represents the best possible loss achievable for a given $p = B/T$.

In this section we will discuss the impact of the interpretability constraint B on predictive performance. To do that we will investigate the RMSE of combining f and g if we assume the optimal solution as given by Proposition 6.1. Formally, we define $S(f, g)$ to be the optimal selector according to Proposition 6.1 for the specific models f and g. Further, since the length of the time-series in each dataset can vary drastically we define the relative interpretability constraint $p = B/T$. In Fig. 6.4 we show (for all datasets and combinations of f and g) how good the optimal selector would perform. To do that we generated 100 samples of $p \in (0, 1)$ and plot the RMSE (y axis) versus p (x axis) for different combinations of f and g.

First, notice that in order to achieve comparable loss to g the optimal selector rarely needs to use g more than 10 to 15 percent of the cases (i.e., we achieve comparable performance at $p = 0.85$ to $p = 0.9$). This suggests that little help from g would be needed to drastically increase the performance of f. The notable exception to this is the KDD Cup dataset since (as mentioned earlier) f is performing very well on this dataset. Second, notice that for roughly $p < 0.6$ the optimal selector stops improving in terms of error. This is due to the fact that in Eq. (6.1) the constraint is stated for *at least* B/T predictions.

Our experiment suggest that (given a perfect model selection mechanism) it is possible to achieve a low RMSE error with very high interpretability constraints, resulting in an overall more interpretable time-series forecasting process. This answers research question **Q1**.

6.3.3. Estimating the optimal solution

We mentioned earlier that, during inference, the optimal selection is not given, and that we estimate it by training classifiers on held-out validation data. Thus, we proceed to evaluate the performance of multiple different classifier from different model families on this classification task:

- `RM`: A random baseline that (given a desired p) generates a selection $\hat{\boldsymbol{s}} \sim \mathcal{B}(T, p)$ where $\mathcal{B}$ is the binomial distribution.
- `LR`: A Logistic Regression, fitted with L-BFGS
- `RF`: A Random Forest with 128 estimators.
- `SVM`: A Support Vector Machine with an RBF kernel.

To also investigate the impact of high class imbalance on the classifiers' performance we also evaluated `RFu`. This is an ensemble of Random Forests where each ensemble member is trained using a balanced version of the dataset. The balancing is done using upsampling (with replacement) of the minority class so that both classes appear in equal proportion. We did not find that upsampling increased performance on either Logisitc Regression or SVMs, which is why we omit it from further discussion.

The result of the experiment can be found in Table 6.3. Specifically, for each dataset and classifier combination we evaluated the F_1 score for different levels

$$p \in \{0.5, 0.6, 0.7, 0.8, 0.9, 0.95\}.$$

We follow the approach outlined in [FS10] to compute one F_1 value per dataset and classifier by computing the true positive (TP), false positive (FP) and false negative (FN) predictions per time-series and summing them over the entire dataset. Then, one F_1 score is computed using the aggregated values via

$$F_1(\mathrm{TP}, \mathrm{FN}, \mathrm{FP}) = (2\mathrm{TP})/(2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}).$$

		$p = 0.5$	$p = 0.6$	$p = 0.7$	$p = 0.8$	$p = 0.9$	$p = 0.95$
Aus. Electricity Demand	RND	0.501	0.600	0.700	0.800	0.900	0.950
	LR	0.620	0.751	0.834	0.901	0.956	0.980
	SVM	0.653	0.769	0.842	0.903	0.954	0.978
	RF	0.700	0.799	0.867	0.920	0.965	**0.984**
	RFu	**0.719**	**0.804**	**0.868**	**0.920**	**0.965**	0.983
NN5 (Daily)	RND	0.516	0.599	0.708	0.801	0.907	0.954
	LR	0.578	0.684	0.795	0.869	0.943	0.973
	SVM	0.587	**0.734**	**0.828**	**0.889**	**0.952**	0.976
	RF	0.585	0.716	0.819	0.884	0.951	0.976
	RFu	**0.637**	0.724	0.812	0.877	0.949	**0.976**
Weather	RND	0.511	0.602	0.701	0.801	0.901	0.951
	LR	0.619	0.757	0.825	0.890	0.947	0.974
	SVM	0.595	0.745	0.824	0.890	0.948	0.974
	RF	0.651	0.759	**0.837**	**0.898**	**0.950**	**0.974**
	RFu	**0.655**	**0.768**	0.833	0.892	0.946	0.972
Pedestrian Counts	RND	0.501	0.600	0.700	0.800	0.900	0.950
	LR	0.636	0.757	0.834	0.898	0.952	0.976
	SVM	0.657	0.769	0.849	0.908	0.957	0.979
	RF	**0.728**	0.809	0.868	0.918	0.961	0.981
	RFu	0.723	**0.819**	**0.875**	**0.922**	**0.962**	**0.981**
KDD Cup	RND	0.531	0.601	0.700	0.800	0.901	0.950
	LR	0.568	0.685	0.785	0.863	0.933	0.965
	SVM	0.595	**0.709**	**0.806**	**0.883**	**0.947**	**0.974**
	RF	0.579	0.674	0.782	0.865	0.933	0.964
	RFu	**0.643**	0.705	0.780	0.857	0.933	0.967
Solar	RND	0.500	0.601	0.700	0.801	0.900	0.950
	LR	0.642	0.758	0.833	0.900	0.949	0.975
	SVM	0.712	0.808	0.852	0.907	0.948	0.975
	RF	**0.784**	0.832	**0.871**	**0.919**	**0.961**	**0.978**
	RFu	0.759	**0.835**	0.865	0.911	0.960	0.976

Table 6.3.: The F_1 scores of all classifiers for every datasets and different settings of p. The best value per dataset and value of p are shown in bold face, while the second-best value is shown with an underline.

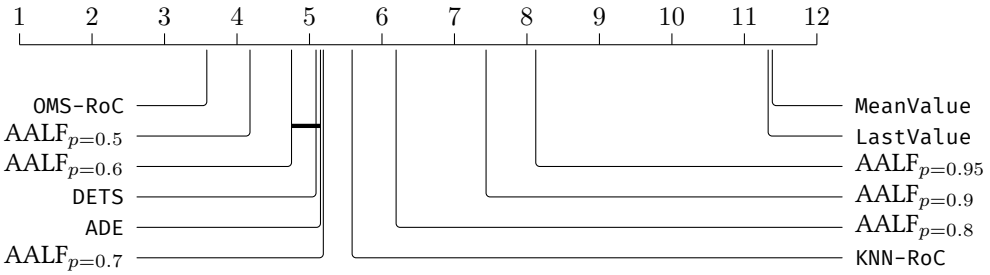

Figure 6.5.: Critical Difference Diagram (over RMSE error) of all evaluated model selectors, computed over all datasets. The average rank is shown above (smaller is better). Two selectors are connected with a horizontal line if they did no show significantly different performance.

Notice that we highlighted the best F_1 score per dataset and value of p in bold, as well as the second-highest value with an underline. As can be seen, both versions of the Random Forest (RF and RFu) perform very well over multiple datasets and values of p, achieving the highest or second highest F_1 scores fairly consistently. Interestingly, the superior performance of RFu is especially noticeable for low values of p (i.e., $p = 0.5$ and $p = 0.6$), suggesting that the balancing of class label proportions is not as important there and that the ensembling of base learners is increasing the performance in these cases instead. Due to the highest empirical performance, especially for lower values of p, we decide to choose RFu for the next experiment. This answers research question **Q2**.

6.3.4. Comparison to state-of-the-art

Finally, we compare our framework, AALF, to state-of-the-art online model selection methods, both in terms of predictive performance and in terms of how often f is chosen over g. AALF consists an ensemble of Random Forest classifiers, each of which was trained on a differently sampled, balanced dataset of input features and optimal selections as outlined before (RFu). In this experiment, we will again (as in the previous experiment) consider multiple different values of p, namely $p \in \{0.5, 0.6, 0.7, 0.8, 0.9, 0.95\}$. The state-of-the-art methods and baselines which we compare ourselves against are:

- ADE [Cer+17a] tries to learn the expected error for each model on unseen data based on measured validation error. Afterwards, a weighting scheme is proposed, assigning each model a weight which can be used for ensembling. We, however, choose the model with the highest weight at each time step for prediction.
- DETS [Cer+17b] is an ensembling method that selects a subset of models for prediction based on recent errors. These errors are computed over a window of past predictions for each model, smoothed and aggregated into a weighting for ensembling. We again choose the model with the highest weight at each time step for prediction.

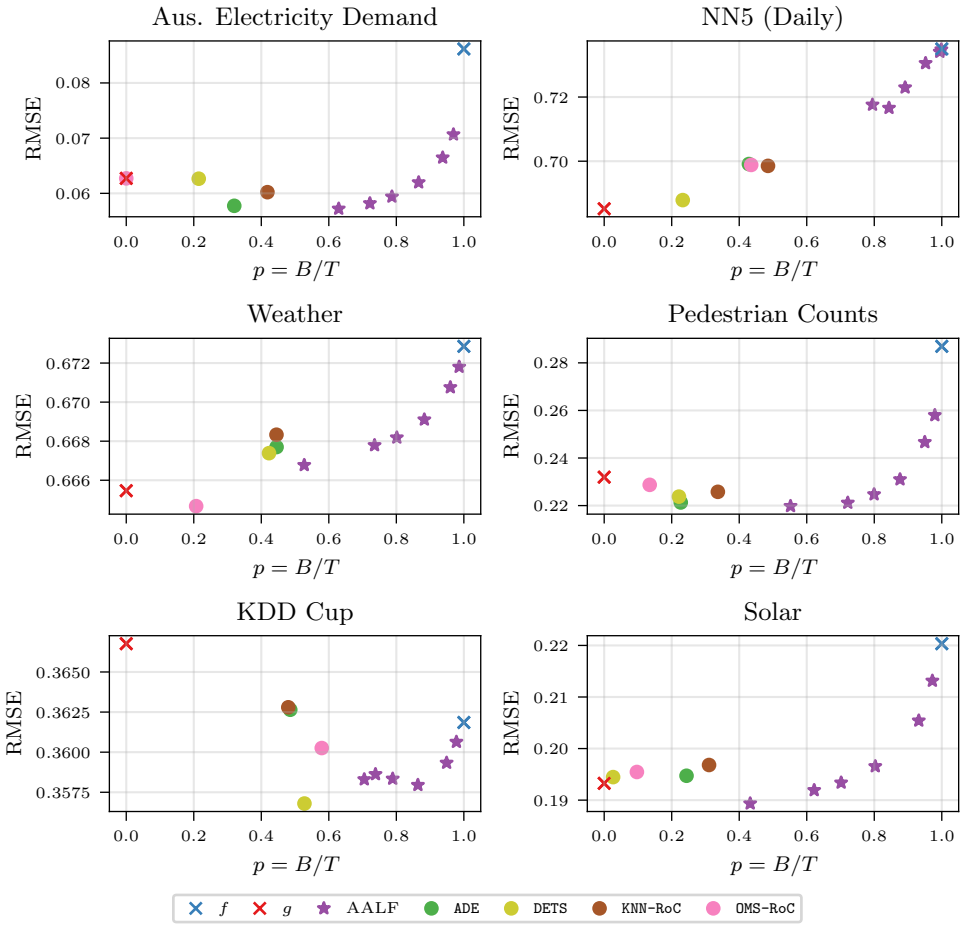

Figure 6.6.: Comparison between average RMSE (y axis) and empirical selection of f (x axis). AALF was evaluated for multiple hyperparameter values of p.

- `KNN-RoC` [Pri19] utilizes the idea of Regions of Competence, where each model keeps a buffer of data points it performed best at during validation. At test time, the new input window is compared to all Regions of Competence and the model with the highest similarity is chosen to predict the next value.
- `OMS-RoC` [Saa23] also utilizes Regions of Competence. However, `OMS-RoC` clusters the data and assigns model expertise to entire clusters, instead of to individual data points.
- `MeanValue` and `LastValue` serve as simple, yet important baselines, where the prediction is the average or last value of the window $\boldsymbol{x}_t$, respectively.

First, we will analyze the overall predictive performance. For that, we created a Critical Difference Diagram (see Fig. 6.5) over all time-series in all datasets. Similar to the CD

diagram in Fig. 6.3 we have the average rank of each method given and draw a horizontal bar between pairs of methods if they are not significantly different.

Unsurprisingly, the simple baselines `MeanValue` and `LastValue` are the worst performing methods. Next, we notice that the versions of AALF that have very high values of p (above $p = 0.8$) perform worse than the comparison methods. The two versions AALF$_{p=0.7}$ and $AALF_{p=0.6}$, however, perform comparably to `DETS` and `ADE`, two popular state-of-the-art online model selection methods. Moreover, $AALF_{p=0.5}$ even significantly outperforms these two selectors, only being surpassed by `OMS-RoC`. This experiment suggests that, depending on the amount of interpretability desired, we are able to achieve comparable performance to current state-of-the-art methods.

Next, we want to analyze the amount of selections of the model f in tandem with the achieved performance. For that, we computed the empirical $p = B/T$ (x axis), as well as the average RMSE (y axis) over all datasets and methods in Fig. 6.6. We show all versions of AALF for all values of p in violet. As can be seen, the exponential trade-off between RMSE and p seen in Fig. 6.4 can also be observed empirically for most datasets. Moreover, we observe an overestimation of p in most of the experiments, i.e., AALF chooses f more than necessary. There is most likely more performance to be gained by tweaking the decision threshold or achieving a better fit of the `RFu` classifier.

As mentioned earlier, `OMS-RoC` significantly outperformed even our best performing selector ($AALF_{p=0.5}$). However, we want to stress that (as can be observed empirically in Fig. 6.6) `OMS-RoC` selects g significantly more often than AALF on most datasets. Thus, in terms of a performance-interpretability trade-off both `OMS-RoC` and AALF perform on different ends of the spectrum. These experiments answer research question **Q3**.

Combining the results of the previous three experiments we are able to provide evidence that an online model selection strategy with a very high interpretability constraint could perform very well in theory (see Section 6.3.2). With choosing g only a fraction of the time we are able to significantly outperform f since the performance increases exponentially with decreasing p (see Fig. 6.4). In our experiments, we are generally able to achieve a good fit to the optimal selection with established classification methods such as Random Forests and Support Vector Machines (see Section 6.3.3). Our proposed upsampling of the minority class further helps to improve the performance in some cases. In other cases (especially for low values of p) the increase in performance is likely due to the fact that we ensembled the classifiers. In comparison to the current state-of-the-art model selection methods (see Section 6.3.4) we performed competitively with a focus on being more interpretable than all other competitors. While being interpretable is not a specific goal of these comparison methods we want to stress the fact that our multiobjective goal of high performance and good interpretability is (to the best of our knowledge) novel in online model selection for time-series forecasting.

6.4. Concluding remarks

In this chapter we presented a novel framework for online model selection with a selection constraint which we called AALF: Almost Always Linear Forecasting. It computes the optimal solution to a constraint optimization problem on held-out validation data, which is then used to train a classifier to perform online model selection. We investigated the theoretically best selection performance, as well as the empirically achieved performance on over 3500 real-world time-series, showing competitive performance against state-of-the-art online model selection methods while being more interpretable overall. More importantly, we were able to provide substantial evidence that a simple, linear model performs very competitively. By predicting when the model will perform bad (and using a more complex, deep, black-box model instead) we could show that we can significantly increase the overall interpretability without sacrificing predictive performance.

7. Interpretable Predictive Maintenance

We have seen in the previous chapter that interpretable methods for time-series analysis can achieve high predictive performance in forecasting. Specifically, we observed that we can learn from data when we should use an uninterpretable model over an interpretable one in an online manner. Additionally, we have discussed in Section 2.5.2 that higher levels of domain-knowledge can enable the use of more interpretable models in general since existing human expert knowledge can be incorporated into the modelling process. While the validity of combining noninterpretable and interpretable models was shown on real-world benchmark sets in the context of forecasting we will proceed with showing the validity on a real-world application scenario.

This chapter will focus on the application scenario of *predictive maintenance*, and how ML methods can be used to help identify and explain potential failures in machines and tools used in real-world scenarios. In general, approaches for tackling maintenance in industry can be classified broadly into three different categories. The most straightforward approach, *corrective maintenance*, replaces machine parts whenever they break. Depending on the industry and machine, this will result in unexpected outages or even harm to people, which is highly undesirable. An alternative is to install a *preventive maintenance* protocol where, periodically, old parts are exchanged for new ones, even if they are not broken. One downside of this approach is the high cost (and potential little benefit) if the old part was not broken. Even worse, exchanging the old part with a new but faulty part (due to a production error) introduces unnecessary failures. In recent years, a third approach has become more attractive to practitioners due to the adoption of ML methods in industry, namely *predictive maintenance*. Here, the goal is to predict if a failure is about to occur and which part is responsible for the upcoming failure based on historical and real-time data provided by the industry process. This approach has the upside of low cost and a small likelihood of unnecessarily introducing faulty parts if the prediction method is accurate. Since predictive maintenance approaches are supposed to be applied in the real world and might impact people and companies financially and physically, there is a high need to understand why a model predicts a failure to be imminent. As stated in [Pas+23], it is not sufficient to know that something is wrong but what causes the failures so that they can be prevented in the future. This makes predictive maintenance an important application scenario for XML and IML approaches.

We will focus on predictive maintenance in the context failure predictions for metro trains in this chapter. In particular, we will investigate a dataset collecting sensor readings for trains run by the *Metro do Porto* in Porto, Portugal. We are concerned with predicting if one critical unit, namely the central Air Production Unit (APU) will experience imminent

failures since the trains are not equipped with a backup unit, and the unit is crucial for the train to be able to operate. We approach this task in two stages, where we use Deep Autoencoders (AEs) that learn characteristics of non-failure data in the first stage. Using the reconstruction error they can alert the practitioners if newly acquired data significantly deviates from the learned characteristics. The second stage consists of an online rule learning approach that can extract high-level, interpretable rules explaining the failure in a more understandable manner compared to related methods.

This chapter is based on the publication [JVG25]. The author of this thesis devised the main idea, specifically the AE architecture and the online rule-learning approach. Moreover, he implemented the code for the experiments and wrote the majority of the original paper.

This chapter is structured as follows. We start by giving an overview over the application setting of the *Metro do Porto* trains, as well as the used MetroPT2 dataset in Section 7.1. Next, we present some proposed methods for failure prediction on MetroPT2 (and some related explainability approaches) in Section 7.2. Our online rule-learning approach for extracting interpretable rules is presented in Section 7.3. We conduct experiments on MetroPT2 in Section 7.5 where we also investigate extracted local and global rules. The chapter will conclude with some final remarks.

7.1. Metro do Porto and MetroPT2

The *Metro do Porto* is the metro system operating in the city of Porto in northern Portugal. At the time of writing it consists of six lines connecting the city and the larger metropolitan area, serving as a crucial mode of transportation. According to the official website the metro can serve up to 9000 passengers per line and hour[10]. It is thus crucial to ensure that the trains are operating on time and that potential downtimes due to failures are minimized. The Air Production Unit (APU) installed onto the trains is of central importance in this regard as it is the main unit responsible for air-related tasks in the train. For example, the APU controls the opening and closing of the train doors and raising or lowering the train at the station's platform. It is important to mention that there is no backup solution for the APU installed. This means that failures of the APU will result in the train being unable to continue operation, inconveniencing passengers in various ways and leading to long delays. Since it takes a train approximately two hours to be decommissioned safely the predictive maintenance task in this case is to predict failures in the APU at least two hours in advance.

In this chapter we will utilize one particular dataset of collected data from trains of *Metro do Porto* called MetroPT2 [Vel+23]. It contains sensor readings of 16 analogue and digital sensors placed onto the APU of a train between April 28th and July 28th 2022, recorded at

[10] https://en.metrodoporto.pt/pages/407

a frequency of 1Hz. The analogue measurements are the following (descriptions adapted from [Vel+23]):

- `TP2` - Pressure on the compressor (bar)
- `TP3` - Pressure generated at the pneumatic panel (bar)
- `H1` - Valve that is activated when the pressure read by the pressure switch of the command is above the operating pressure of 10.2 bar (bar)
- `DV_pressure` - Pressure exerted due to pressure drop generated when air dryers towers discharge the water. When it equals zero, the compressor works under load (bar).
- `Reservoirs` - Pressure inside the air tanks installed on the trains (bar)
- `Oil_temperature` - Temperature of the oil present on the compressor ($^{\circ}C$)
- `Flowmeter` - Airflow was measured on the pneumatic control panel (m^3/h)
- `Motor_Current` - Motor's current, which should present the following values: (i) close to 0A when the compressor turns off; (ii) close to 4A when the compressor is working offloaded; and (iii) close to 7A when the compressor is operating under load (A)

In addition to the analogue sensors there are digital sensors providing a stream of binary values. These sensors are (again, description adapted from [Vel+23]):

- `COMP` - Electrical signal of the air intake valve on the compressor. It is active when there is no admission of air on the compressor, meaning that the compressor turns off or is working offloaded.
- `DV_electric` - Electrical signal that commands the compressor outlet valve. When it is active, it means that the compressor is working under load; when it is not active, it means that the compressor is off or offloaded.
- `Oil_level` - The oil level on the compressor is active (equal to one) when the oil is below the expected values.
- `LPS` - A low-pressure signal, activated when the pressure is below 7 bars. This signal is used for evaluation only and not as a feature.

In addition to the sensor readings the dataset contains two failures which occurred during the recording phase. The first failure is an air leak, while the second is an oil leak. Table 7.1 shows these failures, their duration, and the timestamp at which the `LPS` signal turned on. The `LPS` signal is a failure warning already built into the train conductor control panel. The goal of predicting failures on this dataset is to find the failures two hours before the `LPS` signal turns on so the train can be safely removed from the tracks in time as mentioned earlier.

Failure	Start time	End time	LPS Signal
Air Leak	2022-06-04 10:19:24	2022-06-04 14:22:39	2022-06-04 11:26:01
Oil Leak	2022-07-11 10:10:18	2022-07-14 10:22:08	2022-07-13 19:43:52

Table 7.1.: Failures present in MetroPT2 dataset.

7.2. Predictive Maintenance for MetroPT2

Predictive maintenance, as we will discuss it in this chapter, is closely related to the research field of anomaly detection. In anomaly detection, the goal is to detect data points from a larger set of samples that are significantly different to the other samples. It is a long-studied problem domain that, especially in recent years, is approached more and more with machine learning and deep learning methods [CBK09; Pan+21]. The connection to predictive maintenance is given via the assumption that anomalous data is indicative of irregular behavior. Thus, if some part of the observed machine or process is faulty it is assumed that we can measure a deviation from some initial distribution. In this context we can see that predictive maintenance is a form of *concept drift detection* (see Definition 2.8) as it is usually modelled as detecting a change in data distribution over time.

Due to the assumption that changes in data characteristics can predict failures in the predictive maintenance setting many approaches in the literature focus on learning the characteristics of *failure-free* data using AEs (see Section 2.2). The reconstruction error of an AE can serve as a similarity measure to failure-free data: The smaller the reconstruction error the more likely it is that newly observed data is similar to the training data (which is assumed to be failure-free). This approach of failure prediction using AEs has been applied with success to the MetroPT2 dataset before. In [SVG23] the authors evaluated multiple AE architectures trained on a portion of failure-free data. Specifically, they split the time-series data of the sensor readings into windows of 30 minutes and encoded each window separately. The best model, a Wasserstein AE GAN architecture, achieved a perfect F_1 score, meaning no false alarms were signalled, and both failures in the MetroPT2 dataset were found. In [Gam+24], the authors had a similar approach but found an LSTM-based AE to perform best. Moreover, the authors utilized full compressor cycles, given by the COMP sensor. Each compressor cycle window was discretized into non-overlapping bins, on which summary statistics such as the mean, min and max values were calculated [Dav+21]. In both works, the failures were predicted two hours before the LPS signal.

As mentioned earlier, it is important to not only know if a failure is about to happen but why (and how to prevent it in the future). Thus, both previously mentioned works on failure prediction on the MetroPT2 dataset utilize rule-based explainability approaches [SVG23; Gam+24] to investigate the underlying reason for the predicted failures. Rules have the advantage of being very easy to interpret, as long as they are not excessively

long and as long as the antecedents (e.g., the features) are interpretable themselves. The authors of [SVG23; Gam+24] utilized an explainability layer including AMRules [DGB16; Rib+23]. AMRules is an online rule-learning approach for regression streaming data. The authors use it to learn the AE's reconstruction error based on the raw data from the individual time steps. Moreover, Chebyshev sampling [ARG21] is used to account for the fact that failures are rare events. Afterwards, the rules that have the highest support during the failure duration are returned as an explanation for that failure. We argue that this rule-based approach could be improved by addressing the following two issues: First, the rules are trained on individual time steps from raw data to predict the reconstruction error, i.e., each one hertz vector of sensor values is an individual training example. This results in many rules with varying levels of support since the resolution is very fine-grained. Second, choosing the reconstruction error as a target for rule learning results in overcomplicating the rules unnecessarily since the regression target is a real number. This leads to a very high number of rules, further reducing the interpretability of the approach. We argue that choosing the output of the failure detection pipeline, i.e., $p(\text{failure}) > 0.5$ as a binary label, will lead to more straightforward to comprehend rules since the target is binary.

7.3. Failure detection

We proceed by formalizing the approach for failure detection using AE architectures as this will serve as our basis for detecting failures in this chapter. Since the sensors readings are given as a multivariate time-series let $\boldsymbol{x}_t \in \mathbb{R}^{C \times T}$ be a window of length L consisting of C sensors at time t. We define an AE as a NN architecture consisting of two parts: The encoder $E : \mathbb{R}^{C \times L} \to \mathbb{R}^{m \times L}$ learns a (non-linear) projection of the input channels to an m-dimensional latent vector, preserving the time dimension. The decoder $D : \mathbb{R}^{m \times L} \to \mathbb{R}^{C \times L}$ learns to reconstruct the input from the latent representation. Thus, we get a reconstruction for a window $\boldsymbol{x}_t$ via $\hat{\boldsymbol{x}}_t = D(E(\boldsymbol{x}_t))$. The training objective is to minimize the reconstruction error between each training window and its reconstruction $\hat{\boldsymbol{x}}_i$., which we will define to be the mean squared error:

$$\ell_{\text{MSE}}(\boldsymbol{x}_t, \hat{\boldsymbol{x}}_t) = (LC)^{-1} \sum_{l=1}^{L} \sum_{c=1}^{C} (x_{t,c,l} - \hat{x}_{t,c,l})^2.$$

As mentioned earlier, the reconstruction error for windows of non-anomalous data is assumed to be significantly smaller than the reconstruction error for windows of data during (or shortly preceding) failures. Thus, the reconstruction error can serve as a proxy for the degree of anomaly of a data point.

In terms of the architecture we utilized a simple CNN-based AE architecture based on blocks similar to the residual blocks from Fig. 4.2. Our architecture is shown in Fig. 7.1. The architecture consists of N encoder and decoder blocks. Each block utilizes convolutional layers, batch normalization layers, ReLU activation functions, and Dropout layers. Optionally, an up- or downsample layer (realized with a fully connected layer) is used

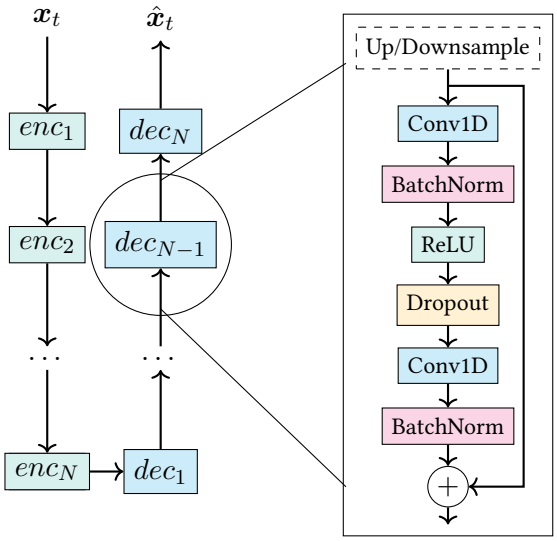

Figure 7.1.: The AE used in this work consists of N encoder and decoder blocks (left) with identical block structures (right).

where needed to ensure correct shapes. We also utilize skip-connections proposed by [He+16] to improve training and generalization.

After fitting the AE on failure-free data we need to decide at which point a reconstruction error is to be considered high enough to indicate that a failure is imminent. By using the boxplot method on the training reconstruction error distribution [RPG16], we define the *anomaly threshold* as

$$\tau_{\text{anom}} = \beta q_{99}$$

where q_{99} is the 99th percentile of the distribution of training reconstruction errors and β is a hyperparameter. Thus, if the reconstruction error for a window of data exceeds τ_{anom}, the slice is considered anomalous. As discussed earlier (and in Section 2.3.1) a single anomalous window is not enough to determine a true drift in concept (or, in our case, to predict a failure). To reduce the number of false-positive alarms, a low-pass filter is applied to this binary output according to Eq. (7.1), where y_t is the observed testing slice prediction (i.e., anomalous or not) at time t and α is a smoothing hyperparameter:

$$z_t = z_{t-1} + \alpha(y_t - z_{t-1}),\ z_0 = y_0. \tag{7.1}$$

Lastly, a failure is detected if the smoothed output exceeds a threshold of $\tau_{\text{fail}} = 0.5$. This reduces the false-positive rate because only prolonged occurrence of anomalous data points should be considered a failure. From now on, we will refer to the smoothed output (at time t) as the probability of failure $p_t(\text{failure})$, i.e.,

$$p_t(\text{failure}) = z_t.$$

7.4. Online rule-learning approach

With our model able to detect failures by monitoring the reconstruction error, we will present our approach for extracting interpretable rules for each detected failure. We propose to compute aggregated values for each window $\boldsymbol{x}_t$ to use as features during the rule-learning approach to get more interpretable rules. Let $\varphi : \mathbb{R}^{L \times C} \to \mathbb{R}^{M \times C}$ be a transformation which aggregates the time dimension of length L using M aggregation functions of type $\phi : \mathbb{R}^{L} \to \mathbb{R}$. In this work, we utilize each windows' variance, minimum, maximum and mean value for higher interpretability. Finally, let $\varphi(\boldsymbol{x}_i)$ be a transformed window. We will see that aggregating over the time dimension will make the rules very general, which helps in interpreting the cause of failure more easily.

Our online rule-learning algorithm is shown in Algorithm 7.1. We will start with an empty set of rules $\boldsymbol{r} = \{\}$ as no failure is assumed prior to starting the algorithm. We assume that there are T windows of data to process and denote the set of indices with $[T] := \{1, 2, \ldots, T\}$. Inspired by the approach presented in [GK14], we define three different states of operation during inference. First, if $p_t(\text{failure}) < \tau_{\text{warn}}$, where τ_{warn} is a hyperparameter, we are in the `Normal` state, i.e., the reconstruction error at time t is considered small. In this state, we add the current transformed window $\varphi(\boldsymbol{x}_t)$ to a history of data H. The history will hold onto non-anomalous data as predicted by our AE model. Once $p_t(\text{failure}) > \tau_{\text{warn}}$, we change to the `Warning` state. During the warning state, we add $\varphi(\boldsymbol{x}_t)$ to a buffer B of anomalous data instead. If we reach the `Failure` state ($p_t(\text{failure}) > \tau_{\text{fail}}$ with $\tau_{\text{fail}} > \tau_{\text{warn}}$) we create a labeled dataset from B and H. Each example in B is labelled as *failure*, while all examples in H are labelled as *no failure*. Using all rules in $\boldsymbol{r}$ we test whether they correctly classify all labelled data points. If a rule $r \in \boldsymbol{r}$ does not correctly classify all labelled data points it is removed from the rule set. If the rule set is empty, we train decision trees on the labelled dataset to perfect accuracy and save them to the rule set. We train multiple trees since (especially for a few anomalous examples) there might be multiple rules that cover all labelled examples due to the greedy nature of the decision tree learning algorithm (see Section 2.4.2). Also, note that we assume that the failure probability is increasing. In contrast, we consider one failure completed if the probability of failure starts decreasing again. Recall that $p(\text{failure})$ results from low-pass filtering the time series of binary failure / non-failure decisions. Thus, if $p(\text{failure})$ stops monotonically increasing, the underlying binary decisions are consecutive non-failure predictions, and we assume the failure to have stopped. See Fig. 7.2 for a schematic overview of this approach.

7.5. Experiments

Next, we will present the results of our AE architecture for failure detection and the extracted rules for each failure from our proposed online rule-learning approach. The code to reproduce all experiments is available online[11].

[11] https://github.com/MatthiasJakobs/metro-xai

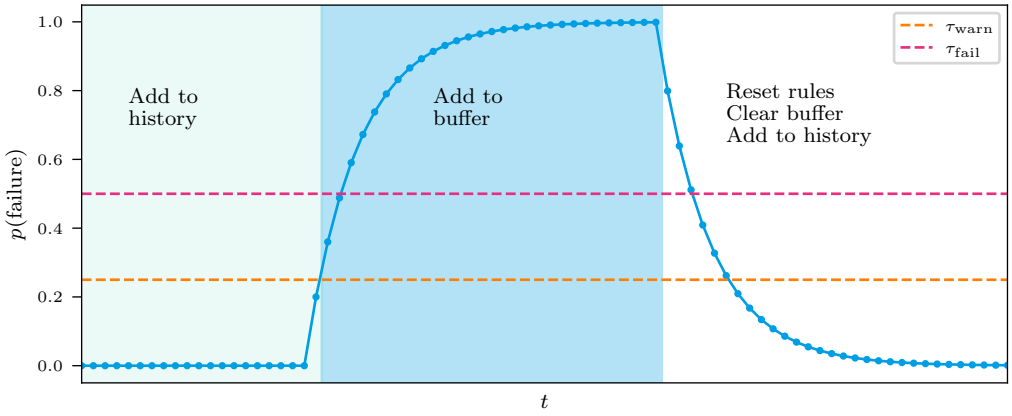

Figure 7.2.: Visualization of our online rule-learning approach. During the green section, we add non-anomalous data to our history. During the blue section, we add the data to the buffer instead and learn fitting rules. Once $p(\text{failure})$ stops monotonically increasing, we reset and return the learned rules, clear the buffer of anomalous data and add the data points to the history again.

Algorithm 7.1 Online rule-learning on failure probability

$B, H, \boldsymbol{r} = \{\}$ ▷ Initialize buffer and history
for $t \in [T]$ **do**
  `isIncreasing` $= p_t(\text{failure}) > p_{t-1}(\text{failure})$
  `isWarning` $= p_t(\text{failure}) > \tau_{\text{warn}}$
  `isFailure` $= p_t(\text{failure}) > \tau_{\text{fail}}$
  if `isWarning` $\wedge$ `isIncreasing` **then**
    $B = B \cup \{\varphi(\boldsymbol{x}_t)\}$ ▷ Enrich buffer
  else
    $B = \{\}$ ▷ Reset buffer
    $H = H \cup \{\varphi(\boldsymbol{x}_t)\}$ ▷ Enrich history instead
  end if
  if `isFailure` **then**
    if `isIncreasing` **then**
      $\mathcal{X}, \mathcal{Y} = \text{CollectExamples}(B, H)$
      $\boldsymbol{r} = \text{UpdateRules}(\mathcal{X}, \mathcal{Y}, \boldsymbol{r})$
    else
      $\boldsymbol{r} = \{\}$ ▷ Return and reset rules
    end if
  end if
end for

Failure	Rule
Air Leak	`Flowmeter_max` > 16.05
Oil Leak	`Flowmeter_max` > 16.18

Table 7.2.: Extracted rules using all available features.

We formulate the following research questions:

- **Q1**: Can we detect both failures two hours before the `LPS` signal activates?
- **Q2**: Which sensors of the APU are most useful in detecting the failures?
- **Q3**: Can we defer simple, interpretable rules for both failures?

7.5.1. Failure detection

We use the following approach based on [SVG23] for preprocessing the dataset. The dataset is first split into two parts: A training period (until 2022-06-01) that is considered free of anomalies and failures and a testing period (from 2022-06-01). Next, the data is transformed into overlapping windows of 30 minutes ($L = 1800$) with a stride of 5 minutes ($d = 300$) to simulate that we receive new sensor values during a real-world application every 5 minutes. The last 30% of training windows are used to measure the validation error during training and estimate τ_{anom} to prevent overfitting to the training data. We normalized each sensor channel by predicting mean and standard deviation over the train set and using it to center the remaining data windows.

In terms of AE architecture, we parameterize the architecture from Section 7.3 in the following way: We utilize $N = 10$ temporal blocks for both the encoder and decoder. All convolution layers have a kernel size of 3 and are made up of 30 kernels each. To increase the receptive field of the encoder and decoder, we increase the dilation of each temporal block exponentially, i.e., each encoder blocks $i \in [N]$ is assigned a dilation of 2^i (and analogous for the decoder). The embedding dimension was experimentally set to 32. Additionally, we set the Dropout probability to 0.2 for all Dropout layers in the AE to prevent overfitting. We train our model using the `Adam` optimizer [KB15] with a learning rate of 10^{-4} for a total of 200 epochs.

After fitting the model to the training data, we found that a smoothing factor $\alpha = 0.15$ and an anomaly threshold of $\tau_{\text{anom}} = 3q_{99}$ lead to a perfect F_1 score, i.e., both failures are detected (more than) 2 hours before the respective `LPS` signal turned on and there were no false positives. We show the results of our method in Fig. 7.3. In the top row, our model $p(\text{failure})$ is very small except for both failure cases (and one slightly anomalous section during late June), suggesting that we highly fit the training data without overfitting it. In the bottom row, we show the two failure periods in more detail. For the Air leak (left), we note that $p(\text{failure}) > 0.5$ starts at 8:43 AM while the `LPS` signal does not turn on until

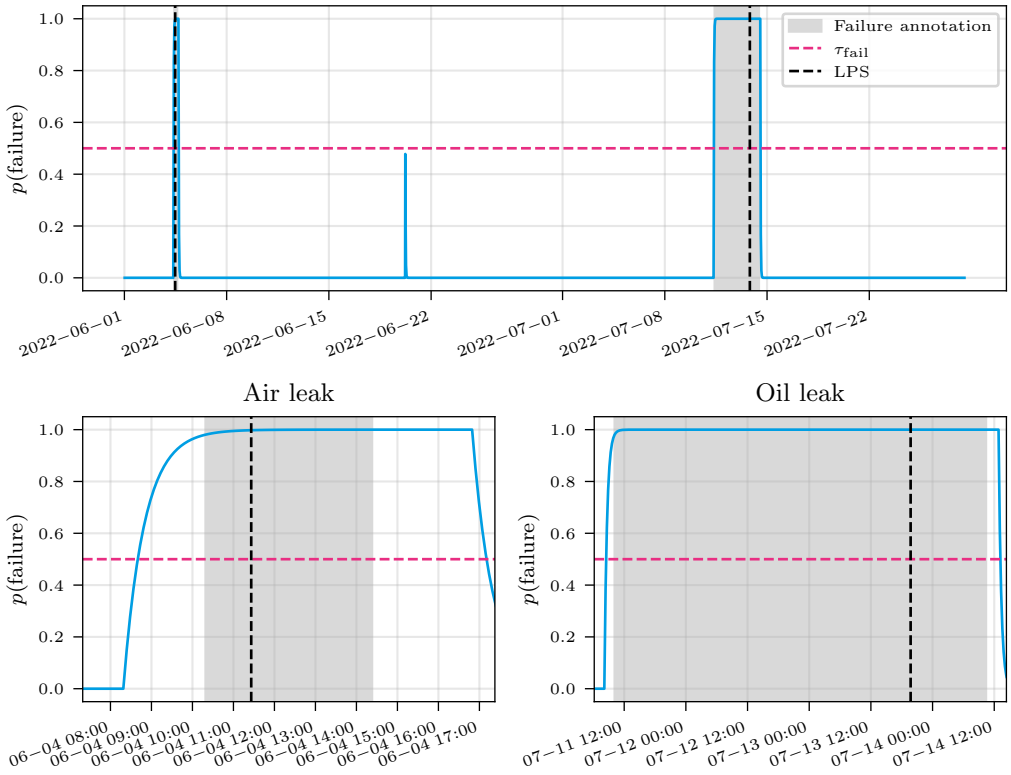

Figure 7.3.: Our models predicted $p(\text{failure})$ (in blue) over the testing period of MetroPT2 (top row) and for both failures in more detail (bottom row). The actual failure periods are shown in gray, as well as the activation of the `LPS` signal as the dotted black lines. The threshold τ_{fail} at which $p(\text{failure})$ is reported as a failure is set to $\tau_{\text{fail}} = 0.5$ and shown in red.

11:26 AM, meaning that we detect the failure over 150 minutes before the `LPS` signal. For the Oil leak (right), we detected the failure at 08:32 AM on July 11th, over two full days before the LPS signal turned on. This answers research question **Q1**.

7.5.2. Explaining detected failures

Next, we apply our online rule-learning algorithm on $p_t(\text{failure})$ estimated by our AE model to investigate probable causes of both detected failures. We are able to recover one rule for each failure, covering the entire failure period. The rules are shown in Table 7.2. Interestingly, there seems to be one highly predictive sensor, namely `Flowmeter`, which is enough to predict whether or not the train will experience a failure. Specifically, if the value in a 30 minute slice of data at any point exceeds a threshold of roughly $16m^3/h$ then this slice is part of a failure. Notice that this is true for both failures. Due to our rule-learning algorithm, no false positives are associated with preceding data, i.e., the found rules do not produce false positives for all windows preceding the respective failure.

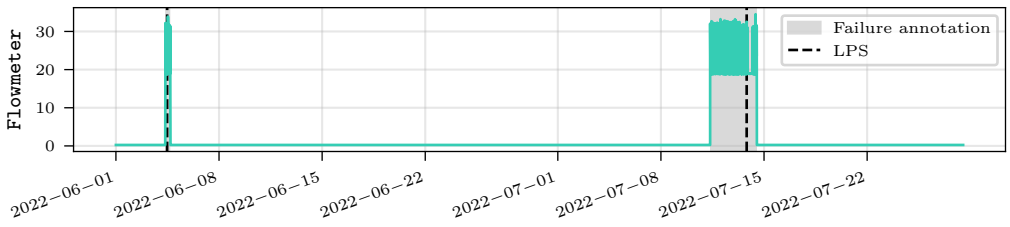

Figure 7.4.: The `Flowmeter` sensor during the entirety of the test data range. Notice that, most of the time, the values of `Flowmeter` are minimal and only increase during the annotated failures. For a more detailed view of both failures, see Fig. 7.5.

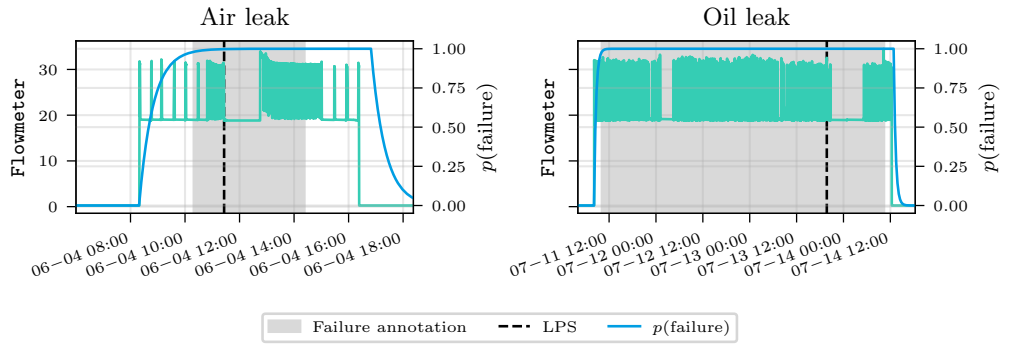

Figure 7.5.: The `Flowmeter` sensor during the Air and Oil leaks. `Flowmeter` values are shown in green while the $p(\text{failure})$ is shown in blue. A clear correlation between failures and high values of `Flowmeter` makes this sensor highly predictive of these failures.

The strong correlation between high values of `Flowmeter` and the annotated failures is also apparent when investigating the raw data visually, as can be seen in Fig. 7.4 and Fig. 7.5. We can observe that `Flowmeter` usually has small values, with sporadic peaks aligning perfectly with the observed failures. We also want to stress that multiple valid thresholds could be used in this case since the difference between failure and no failure value ranges is very large. Thus, as long as this sensor is present in the APU, early failure detection appears to be a relatively simple process that does not require complex deep-learning methods. Indeed, as can be observed in Fig. 7.5, the probability of failure (shown in blue) increases at the same time as the raw values of `Flowmeter`, suggesting a high reliance of our model on this sensor (in particular, a high reconstruction error of that sensor channel).

The correlation between high values of the `Flowmeter` sensor with both failures answer research questions **Q2** and **Q3**. However, we want to note that `Flowmeter` is a fairly expensive sensor to install, and not all trains of the *Metro do Porto* in the real world can be equipped with this sensor. Thus, we also intended to find suitable rules not dependent upon `Flowmeter`. After removing the `Flowmeter` sensor feature from the data used for

Failure	Rule
Air Leak	$\texttt{Oil\_temperature\_max} > 72.16$
Oil Leak	$\texttt{TP2\_max} > 10.61 \lor (\texttt{TP2\_max} \leq 10.61 \land \texttt{Motor\_current\_min} \leq 0.01)$

Table 7.3.: Extracted rules using all available features except for `Flowmeter`.

rule-learning, we were able to find rules that focus on `Oil_temperature`, `Motor_current` and `TP2` (see Table 7.3). These sensors are significantly cheaper to produce and install and thus might be more suitable to be rolled out into every train of *Metro do Porto*.

To further investigate the rules shown in Table 7.3, we investigated the purity of each Decision Tree node that generated these rules. We found for the Oil leak that the `Motor_current` feature is only necessary to separate one window from the other $53,486$ windows, meaning that the simple rule $\texttt{TP2\_max} > 10.61$ could be considered a suitable explanation for the Oil leak. The ability to also detect both failures using the `Motor_current`, `Oil_temperature` and `TP2` sensors gives more insights into research question **Q2**.

7.5.3. Global rules

Lastly, we want to investigate whether we can find interpretable global rules that work for both failures to understand better what constitutes anomalous and non-anomalous data. We slightly change the Algorithm presented in Algorithm 7.1 by adding the buffer windows after each failure is completed to a separate global buffer. Thus, at the end of the available test data, we train a Decision Tree to classify all failure windows (found in the global buffer) from all non-failure windows (found in the history).

As seen in Fig. 7.4, finding a global rule for both failures is effortless when considering the `Flowmeter` sensor. Our algorithm found the following rule that covers all failure windows and did not produce any false positives:

$$\texttt{Flowmeter\_max} > 9.58 \Rightarrow \text{Failure}.$$

When omitting `Flowmeter` we learned the Decision Tree shown in Table 7.3. Similar to the local rule found for the Oil leak, we want to stress that the split on `TP2` is only necessary to separate two windows from the remaining $57,148$ windows. Thus, we argue that `Motor_current` and `Oil_temperature` can be considered sufficient to find all failures, answering research question **Q3** with regards to simple, globaly applicable rules for MetroPT2.

7.6. Concluding remarks

After investigating the extracted rules as well as the raw data itself, we conclude that the MetroPT2 dataset, as presented in [Vel+23], is not very challenging for failure and nov-

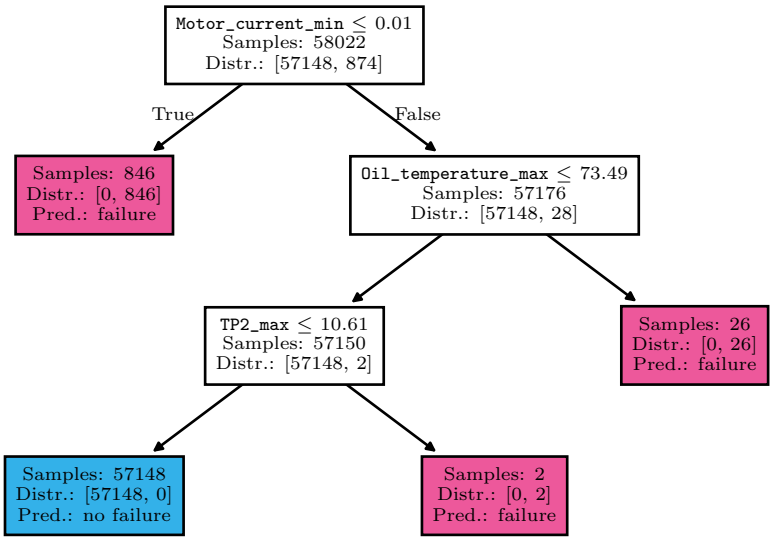

Figure 7.6.: Decision Tree that separates both failures in MetroPT2 from non-failure data without relying on the `Flowmeter` sensor.

elty prediction due to the high correlation between values of `Flowmeter` and the presence of failures in the APU. Thus, we suggest that future work which targets the MetroPT2 dataset remove this sensor from the data before applying their methods. Initial experiments from our side suggested that modelling $p(\text{failure})$ using AEs without providing `Flowmeter` to the model is a substantially more complex problem. While we could find rules that do not involve `Flowmeter` that still fit all slices of failure windows, it is tough to learn the notion of thresholding maximum and minimum values with Convolutional AE architectures. The dependence on `Flowmeter` is also why we struggled to fit a good AE on the related MetroPT3 dataset [Vel+22]. First, MetroPT3 does not contain the `Flowmeter` sensor. Second, our initial experiments showed many false-positive failure predictions, suggesting that different methods than our Convolutional AE to model $p(\text{failure})$ would be necessary. Thus, we recommend that future work for predictive maintenance on *Metro do Porto* data should also focus on MetroPT3 due to its more challenging nature.

One explicit limitation of our work is that an infinite number of rules could be retrieved from the data. For instance, choosing as a global threshold for `Flowmeter` the value 9.59 instead of 9.58 (as shown previously) will still result in a perfect F_1 score. Thus, we cannot confidently gain knowledge about potential breaking points of parts of the APU from these rules. Another limitation is the correct optimization of hyperparameters such as the smoothing parameter α and boxplot threshold βq_i. While we observe that a range of values for these hyperparameters produces satisfactory results, optimizing with validation data containing actual failures would be a more thorough way of setting these values.

Lastly, in the current iteration, the amount of history saved by our online rule-learning algorithm is unbounded. While this results in rules that fit all previous non-failure windows perfectly, this solution will not scale in real-world scenarios where months and years of data could be gathered. We suggest that future work investigate different known strategies to keep a maximum history size, such as random subsampling or removing windows from further in the past with higher probability. Additionally, since this history is merely there to serve as examples for non-failure data windows, clustering methods could be highly efficient in finding representative windows that summarize large amounts of data from history.

8. Conclusion

In this thesis, we discussed how XML and IML methods in the time-series domain (specifically for forecasting and predictive maintenance) can help to gain insights into the decision of complex, DL models, as well as help in building more interpretable methods capable of being applied to safety-critical application settings.

We have seen in Chapter 3 that Explainable Online Model Selection for forecasting can be achieved using a pool of pre-trained NN methods, specifically using Convolutional Neural Networks. By using a drift detection mechanism based on the Hoeffding bound we showed that we can successfully adapt to evolving time-series characteristics, known as concept drifts. Moreover, by creating a Region of Competence of most salient subsequences for each forecaster we can visually inspect each models expertise and answer the question of why a specific model was selected at each time step. We have also shown in the next chapter (Chapter 4) how this approach can be extended to Online Ensemble Pruning. As before, we were able to successfully adapt to concept drifts by enriching the RoCs with newly observed patterns and (if necessary) re-prune the ensemble using our proposed two-stage pruning approach. In Chapter 5 we presented an approach which limits the pool of models to interpretable, tree-based models. By doing that we are able to achieve even more interpretable forecasting results due to the transparent nature of trees. We showed that we can investigate also how robust each lagged value is to noise, giving experts even more insight into the decision-making process. Overall, this second part of the thesis successfully showed that Online Model Selection and Ensembling can be a powerful tool for adapting to concept drifts in the time-series forecasting setting and that we are able to make the selection process and the forecasts itself vastly more explainable than they were before.

In Chapter 6 we investigated the hypothesis that interpretable forecasting models are good enough for most prediction tasks, and that complex, DL methods might not be necessary all the time to achieve high prediction performance. Specifically, we formalized a constraint version of binary online model selection where we require the use of an interpretable model whenever possible. We could show that, in theory, we only need to use a high performing DL model for only 15 percent of data points we wanted to forecast. Moreover, we are able to learn when to use which model from data, leading to an overall interpretable approach using very tiny, transparent models. After investigating this way of combining interpretable and non-interpretable models we investigated a specific, real-world application scenario for interpretable time-series analysis, namely predictive maintenance in Chapter 7. Specifically, we showed that we can use a DL autoencoder network to successfully predict when failures in the studied Metro trains from Porto, Portugal are

about to occur. Moreover, we showed that there are very simple rules predicting these failures, which we learned using a simple online rule-based learning method. Overall, we were able to show in this third part of the thesis that the combination of interpretable and non-interpretable methods is a worthwhile pursuit for future studies and can lead to an overall increase in trustworthiness and understanding while not sacrificing the predictive performance.

We have argued in this thesis that time-series data analysis is a crucial part of many high-stakes decision-making settings such as in medical, financial and industrial domains. Also, we have seen in recent years that ML approaches can perform comparable to human experts and outperform traditional methods, suggesting that novel insights and increased automation could lead to positive outcomes in the aforementioned high-stakes domains. It is therefore of high importance that ML are either simple enough to be understood fully or that they can be investigated sufficiently in order to be applied to those crucial settings. We have seen that visually investigating model expertise via Regions of Competence is one way that more transparency can be achieved. Additionally, they allow to investigate the changes in data via the analysis of model expertise before and after time-series concept drifts, which is also a crucial part of achieving more trustworthiness. Moreover, in the last two chapters of this thesis we made the observation that a combination of opaque DL approaches and interpretable time-series models can achieve comparable performance to complex ML models while being much more interpretable overall. We believe that this approach of combining the strengths of both model types is a crucial avenue for future work to ensure the adoption of ML methods for the greater good.

8.1. Limitations and Future Work

Lastly, we want to highlight some limitations to our works and how they might be addressed in future work. In Chapters 3 to 6 we focused our attention on univariate time-series data. However, many real-world application settings are inherently multivariate. Since a straight-forward extension to multivariate data is not possible with the methodology presented in this thesis there is room for improvement in future work about how a RoC might be visualized, for example, for multivariate data. Moreover, many interpretability enhancements we were able to gain by visualizing data and calculating attribution-based explanations are complicated when using multivariate time-series data. Thus, we suggest that future work addresses these challenges to further increase the interpretability and explainability of time-series forecasting in real-world application scenarios.

While the approach presented in Chapter 6 is straight-forward to formalize we are aware that choosing between multiple different but equally interpretable models would likely increase the predictive performance further. For example, tree-based models or Exponential Smoothing models might help to reduce the use of DL methods even further. We see the research direction of Online Model Selection with (mostly) interpretable base models as promising to achieve high performance and greater insight in the future. Moreover, while

proposing an interpretability budget is a crucial first step it is not a theoretical guarantee. Future work into the direction of guaranteeing a certain percentage of interpretable model decisions in expectation could be vital for adoption in high stakes decision-making contexts.

For Chapter 7, we want to note that our approach of extracting interpretable rules is very sensitive to the specific hyperparameters used. Specifically, we noticed that the smoothing and thresholding parameters contribute largely to how many failures are detected. We could envision future work addressing this using, for example, approaches from the AutoML literature where the goal is to automate hyperparameter optimization. This could not only increase the ease of using our method in more real-world scenarios but also reduce any bias in tuning these sensitive hyperparameters by hand. Moreover, estimating and then analyzing the underlying stochastic process (as defined in Section 2.3) into the modeling process could increase the interpretability further. For example, considering different types of concept drift in parallel sensors might lead to a deeper insight into the reasons for a faulty part of failure.

Bibliography

[MR22] Katharina Morik and Wolfgang Rhode. *Machine Learning under Resource Constraints - Discovery in Physics*. 2022. DOI: 10.1515/9783110785968 (cit. on p. 3).

[Ull+20] Zaib Ullah et al. "Applications of Artificial Intelligence and Machine Learning in Smart Cities". In: *Computer Communications* 154 (2020), pp. 313–323. DOI: 10.1016/j.comcom.2020.02.069 (cit. on pp. 3, 15).

[Rai+21] Rahul Rai et al. "Machine Learning in Manufacturing and Industry 4.0 Applications". In: *International Journal of Production Research* 59 (2021), pp. 4773–4778. DOI: 10.1080/00207543.2021.1956675 (cit. on p. 3).

[Per+21] Mathias Perslev et al. "U-Sleep: Resilient High-Frequency Sleep Staging". In: *npj Digital Medicine* 4 (2021), pp. 1–12. DOI: 10.1038/s41746-021-00440-5 (cit. on pp. 3, 5).

[SVG23] Miguel E. P. Silva, Bruno Veloso, and João Gama. "Predictive Maintenance, Adversarial Autoencoders and Explainability". In: *Machine Learning and Knowledge Discovery in Databases: Applied Data Science and Demo Track*. 2023. DOI: 10.1007/978-3-031-43430-3_16 (cit. on pp. 3, 118, 119, 123).

[JVG25] Matthias Jakobs, Bruno Veloso, and João Gama. "Interpretable Rules for Online Failure Prediction: A Case Study on the Metro Do Porto Dataset". In: *arXiv:2502.07394 [cs]* (2025). DOI: 10.48550/arXiv.2502.07394 (cit. on pp. 3, 7, 116).

[Kra+17] Bartosz Krawczyk et al. "Ensemble Learning for Data Stream Analysis: A Survey". In: *Information Fusion* 37 (2017), pp. 132–156. DOI: 10.1016/j.inffus.2017.02.004 (cit. on p. 3).

[Ser+22] Oscar Serradilla et al. "Deep Learning Models for Predictive Maintenance: A Survey, Comparison, Challenges and Prospects". In: *Applied Intelligence* 52 (2022), pp. 10934–10964. DOI: 10.1007/s10489-021-03004-y (cit. on pp. 3, 15).

[SPM19] Amal Saadallah, Florian Priebe, and Katharina Morik. "A Drift-Based Dynamic Ensemble Members Selection Using Clustering for Time Series Forecasting". In: *Machine Learning and Knowledge Discovery in Databases*. 2019, pp. 678–694. DOI: 10.1007/978-3-030-46150-8_40 (cit. on pp. 3, 45, 49, 60, 71).

[Gam+14] João Gama et al. "A Survey on Concept Drift Adaptation". In: *ACM Computing Surveys* 46 (2014), pp. 1–37. DOI: 10.1145/2523813 (cit. on pp. 3, 18, 19).

[LZ21] Bryan Lim and Stefan Zohren. "Time-Series Forecasting with Deep Learning: A Survey". In: *Philosophical Transactions of the Royal Society A: Mathematical,*

Physical and Engineering Sciences 379 (2021). DOI: 10.1098/rsta.2020.0209 (cit. on pp. 4, 15, 23).

[Zho+21] Haoyi Zhou et al. "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35 (2021). DOI: 10.1609/aaai.v35i12.17325 (cit. on p. 4).

[Wu+21] Haixu Wu et al. "Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting". In: *Advances in Neural Information Processing Systems*. 2021, pp. 22419–22430 (cit. on p. 4).

[HH19] Ehsan Hoseinzade and Saman Haratizadeh. "CNNpred: CNN-based Stock Market Prediction Using a Diverse Set of Variables". In: *Expert Systems with Applications* 129 (2019), pp. 273–285. DOI: 10.1016/j.eswa.2019.03.029 (cit. on pp. 4, 52, 69).

[KC19] Tae-Young Kim and Sung-Bae Cho. "Predicting Residential Energy Consumption Using CNN-LSTM Neural Networks". In: *Energy* 182 (2019), pp. 72–81. DOI: 10.1016/j.energy.2019.05.230 (cit. on pp. 4, 43).

[Shi+15] Xingjian Shi et al. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: *Advances in Neural Information Processing Systems*. 2015, pp. 802–810 (cit. on p. 4).

[RHD17] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. "Right for the Right Reasons: Training Differentiable Models by Constraining Their Explanations". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. 2017, pp. 2662–2670. DOI: 10.24963/ijcai.2017/371 (cit. on p. 4).

[Sch+20] Patrick Schramowski et al. "Making Deep Neural Networks Right for the Right Scientific Reasons by Interacting with Their Explanations". In: *Nature Machine Intelligence* 2 (2020), pp. 476–486. DOI: 10.1038/s42256-020-0212-3 (cit. on p. 4).

[Mol20] Christoph Molnar. *Interpretable Machine Learning*. 2020 (cit. on pp. 4, 37).

[Rud19] Cynthia Rudin. "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead". In: *Nature Machine Intelligence* 1 (2019), pp. 206–215. DOI: 10.1038/s42256-019-0048-x (cit. on pp. 4, 39, 40).

[Bec+23] Katharina Beckh et al. "Harnessing Prior Knowledge for Explainable Machine Learning: An Overview". In: *Proceedings of the IEEE Conference on Secure and Trustworthy Machine Learning*. 2023. DOI: 10.1109/SaTML54575.2023.00038 (cit. on pp. 4, 8).

[Aga+21] Rishabh Agarwal et al. "Neural Additive Models: Interpretable Machine Learning with Neural Nets". In: *Advances in Neural Information Processing Systems*. 2021, pp. 4699–4711 (cit. on p. 4).

[BB19] Wieland Brendel and Matthias Bethge. "Approximating CNNs with Bag-of-local-Features Models Works Surprisingly Well on ImageNet". In: *Proceedings of the International Conference on Learning Representations*. 2019 (cit. on p. 4).

[Zen+23] Ailing Zeng et al. “Are Transformers Effective for Time Series Forecasting?” In: *Proceedings of the AAAI Conference on Artificial Intelligence* (2023). DOI: 10.1609/aaai.v37i9.26317 (cit. on p. 4).

[FH17] Nicholas Frosst and Geoffrey Hinton. “Distilling a Neural Network Into a Soft Decision Tree”. In: *arXiv:1711.09784 [cs, stat]* (2017). DOI: 10.48550/arXiv.1711.09784 (cit. on pp. 5, 38).

[Xu+20] Junjie Xu et al. “Anomaly Detection on Electroencephalography with Self-supervised Learning”. In: *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine*. Dec. 2020. DOI: 10.1109/BIBM49941.2020.9313163 (cit. on p. 5).

[Che+20] Guangyuan Chen et al. “Anomaly Detection in EEG Signals: A Case Study on Similarity Measure”. In: *Computational Intelligence and Neuroscience* 2020 (2020), p. 6925107. DOI: 10.1155/2020/6925107 (cit. on p. 5).

[Sel+19] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: *International Journal of Computer Vision* 128 (2019), pp. 336–359. DOI: 10.1007/s11263-019-01228-7 (cit. on pp. 5, 37, 44, 47).

[LL17] Scott M. Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4765–4774 (cit. on pp. 5, 37, 38, 81, 82).

[Lun+20] Scott M. Lundberg et al. “From Local Explanations to Global Understanding with Explainable AI for Trees”. In: *Nature Machine Intelligence* 2 (Jan. 2020), pp. 56–67. DOI: 10.1038/s42256-019-0138-9 (cit. on pp. 5, 37, 83).

[Pri19] Florian Priebe. “Dynamic Model Selection for Automated Machine Learning in Time Series”. PhD thesis. TU Dortmund University, 2019 (cit. on pp. 6, 43, 45, 46, 51, 111).

[JL25] Matthias Jakobs and Thomas Liebig. “AALF: Almost Always Linear Forecasting”. In: *arXiv:2409.10142 [cs]* (2025). DOI: 10.48550/arXiv.2409.10142 (cit. on pp. 7, 100).

[JS23] Matthias Jakobs and Amal Saadallah. “Explainable Adaptive Tree-based Model Selection for Time-Series Forecasting”. In: *Proceedings of the International Conference on Data Mining*. 2023. DOI: 10.1109/ICDM58522.2023.00027 (cit. on pp. 7, 79).

[SJM22] Amal Saadallah, Matthias Jakobs, and Katharina Morik. “Explainable Online Ensemble of Deep Neural Network Pruning for Time Series Forecasting”. In: *Machine Learning* (2022). DOI: 10.1007/s10994-022-06218-4 (cit. on pp. 7, 61).

[SJM21] Amal Saadallah, Matthias Jakobs, and Katharina Morik. “Explainable Online Deep Neural Network Selection Using Adaptive Saliency Maps for Time Series Forecasting”. In: *Machine Learning and Knowledge Discovery in Databases: Research Track*. 2021. DOI: 10.1007/978-3-030-86486-6 (cit. on pp. 7, 43).

[Hee+25] Raoul Heese et al. “Explaining Quantum Circuits with Shapley Values: Towards Explainable Quantum Machine Learning”. In: *Quantum Machine Intelligence* 7 (2025). DOI: 10.1007/s42484-025-00254-8 (cit. on p. 8).

[Cas+24] Bruno Casella et al. “Federated Time Series Classification with ROCKET Features”. In: *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2024. DOI: 10.14428/esann/2024.es2024-61 (cit. on p. 8).

[SJ23] Amal Saadallah and Matthias Jakobs. “Online Deep Hybrid Ensemble Learning for Time Series Forecasting”. In: *Machine Learning and Knowledge Discovery in Databases: Research Track*. 2023, pp. 156–171. DOI: 10.1007/978-3-031-43424-2_10 (cit. on p. 8).

[Hee+23] Raoul Heese et al. “Shapley Values with Uncertain Value Functions”. In: *Advances in Intelligent Data Analysis*. 2023. DOI: 10.1007/978-3-031-30047-9_13 (cit. on p. 8).

[Fis+23] Raphael Fischer et al. “A Unified Framework for Assessing Energy Efficiency of Machine Learning”. In: *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. 2023, pp. 39–54. DOI: 10.1007/978-3-031-23618-1_3 (cit. on p. 8).

[Mül+23] Sebastian Müller et al. “An Empirical Evaluation of the Rashomon Effect in Explainable Machine Learning”. In: *Machine Learning and Knowledge Discovery in Databases: Research Track*. 2023. DOI: 10.1007/978-3-031-43418-1_28 (cit. on p. 8).

[FJM23] Raphael Fischer, Matthias Jakobs, and Katharina Morik. “Energy Efficiency Considerations for Popular AI Benchmarks”. In: *arXiv:2304.08359 [cs]* (2023). DOI: 10.48550/arXiv.2304.08359 (cit. on p. 8).

[Jak+22] Matthias Jakobs et al. “SancScreen: Towards a Real-world Dataset for Evaluating Explainability Methods”. In: *Proceedings of the Conference ”Lernen, Wissen, Daten, Analysen”*. 2022 (cit. on pp. 8, 81).

[WJM22] Rahel Wilking, Matthias Jakobs, and Katharina Morik. “Fooling Perturbation-Based Explainability Methods”. In: *Workshop on Trustworthy Artificial Intelligence as Part of the ECML-PKDD 22*. 2022 (cit. on p. 8).

[Mor+22] Katharina Morik et al. “Yes We Care!-Certification for Machine Learning Methods through the Care Label Framework”. In: *Frontiers in Artificial Intelligence* 5 (2022). DOI: 10.3389/frai.2022.975029 (cit. on p. 8).

[Fis+20] Raphael Fischer et al. “Solving Abstract Reasoning Tasks with Grammatical Evolution”. In: *Proceedings of the Conference ”Lernen, Wissen, Daten, Analysen”*. 2020 (cit. on p. 8).

[Dij59] Edsger W. Dijkstra. “A Note on Two Problems in Connexion with Graphs”. In: *Numerische Mathematik* 1 (1959), pp. 269–271. DOI: 10.1007/BF01386390 (cit. on p. 9).

[Bel58] Richard E. Bellman. “On a Routing Problem”. In: *Quarterly of Applied Mathematics* 16 (1958), pp. 87–90. DOI: 10.1090/qam/102435 (cit. on p. 9).

[For56] Lester R. Ford. *Network Flow Theory*. Tech. rep. RAND Corporation, 1956 (cit. on p. 9).

[Vap91] Vladimir Vapnik. “Principles of Risk Minimization for Learning Theory”. In: *Advances in Neural Information Processing Systems*. 1991 (cit. on p. 10).

[HTF17] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, NY, 2017. DOI: 10.1007/978-0-387-84858-7 (cit. on pp. 11–13, 32, 33).

[Fis36] Ronald A. Fisher. "The Use of Multiple Measurements in Taxonomic Problems". In: *Annals of Eugenics* 7 (1936), pp. 179–188. DOI: 10.1111/j.1469-1809.1936.tb02137.x (cit. on p. 11).

[GBC13] Natasa Gisev, J. Simon Bell, and Timothy F. Chen. "Interrater Agreement and Interrater Reliability: Key Concepts, Approaches, and Applications". In: *Research in Social and Administrative Pharmacy* 9 (2013), pp. 330–338. DOI: 10.1016/j.sapharm.2012.04.004 (cit. on p. 13).

[Llo82] Stuart P. Lloyd. "Least Squares Quantization in PCM". In: *IEEE Transactions on Information Theory* 28 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489 (cit. on pp. 14, 15, 63).

[Ism+19] Hassan Ismail Fawaz et al. "Deep Learning for Time Series Classification: A Review". In: *Data Mining and Knowledge Discovery* 33 (2019), pp. 917–963. DOI: 10.1007/s10618-019-00619-1 (cit. on pp. 15, 36).

[Bag+17] Anthony Bagnall et al. "The Great Time Series Classification Bake off: A Review and Experimental Evaluation of Recent Algorithmic Advances". In: *Data Mining and Knowledge Discovery* 31 (2017), pp. 606–660. DOI: 10.1007/s10618-016-0483-9 (cit. on pp. 15, 52, 69).

[BKM15] Krisztian Buza, Júlia Koller, and Kristóf Marussy. "PROCESS: Projection-Based Classification of Electroencephalograph Signals". In: *Proceedings of the Conference on Artificial Intelligence and Soft Computing*. 2015. DOI: 10.1007/978-3-319-19369-4_9 (cit. on p. 15).

[Tan+21] Chang Wei Tan et al. "Time Series Extrinsic Regression". In: *Data Mining and Knowledge Discovery* 35 (2021), pp. 1032–1060. DOI: 10.1007/s10618-021-00745-9 (cit. on p. 15).

[ASW15] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. "Time-Series Clustering – A Decade Review". In: *Information Systems* 53 (2015), pp. 16–38. DOI: 10.1016/j.is.2015.04.007 (cit. on p. 15).

[Bor17] Andrei N. Borodin. *Stochastic Processes*. Springer International Publishing, 2017. DOI: 10.1007/978-3-319-62310-8 (cit. on pp. 16, 17).

[Ham94] James D. Hamilton. *Time Series Analysis*. Princeton University Press, 1994. DOI: 10.1515/9780691218632 (cit. on pp. 16, 17).

[SG86] Jeffrey C. Schlimmer and Richard H. Granger. "Incremental Learning from Noisy Data". In: *Machine Learning* 1 (1986), pp. 317–354. DOI: 10.1007/BF00116895 (cit. on p. 18).

[WK96] Gerhard Widmer and Miroslav Kubat. "Learning in the Presence of Concept Drift and Hidden Contexts". In: *Machine Learning* 23 (1996), pp. 69–101. DOI: 10.1023/A:1018046501280 (cit. on p. 18).

[Hol04] Charles C. Holt. "Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages". In: *International Journal of Forecasting* 20 (2004), pp. 5–10. DOI: 10.1016/j.ijforecast.2003.09.015 (cit. on pp. 19, 22).

[Win60] Peter R. Winters. "Forecasting Sales by Exponentially Weighted Moving Averages". In: *Management Science* 6 (1960), pp. 324–342. DOI: 10.1287/mnsc.6.3.324 (cit. on pp. 19, 22).

[HA21] Robert J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practices*. 3rd ed. OTexts: Melbourne, Australia, 2021 (cit. on pp. 19, 21, 22).

[APV12] George Athanasopoulos, Don S. Poskitt, and Farshid Vahid. "Two Canonical VARMA Forms: Scalar Component Models Vis-à-Vis the Echelon Form". In: *Econometric Reviews* 31 (2012), pp. 60–83. DOI: 10.1080/07474938.2011.607088 (cit. on p. 22).

[Hyn+02] Rob J. Hyndman et al. "A State Space Framework for Automatic Forecasting Using Exponential Smoothing Methods". In: *International Journal of Forecasting* 18 (2002), pp. 439–454. DOI: 10.1016/S0169-2070(01)00110-8 (cit. on p. 22).

[FS24] Raphael Fischer and Amal Saadallah. "AutoXPCR: Automated Multi-Objective Model Selection for Time Series Forecasting". In: *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2024. DOI: 10.1145/3637528.3672057 (cit. on pp. 23, 43).

[NH10] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the International Conference on Machine Learning*. 2010 (cit. on pp. 24, 28).

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]* (2015). DOI: 10.48550/arXiv.1502.03167 (cit. on p. 26).

[Hin+12] Geoffrey E. Hinton et al. "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors". In: *arXiv:1207.0580 [cs]* (2012). DOI: 10.48550/arXiv.1207.0580 (cit. on p. 26).

[LeC+89] Yann LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1 (Dec. 1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541 (cit. on p. 26).

[Elm90] Jeffrey L. Elman. "Finding Structure in Time". In: *Cognitive Science* 14 (1990), pp. 179–211. DOI: 10.1207/s15516709cog1402_1 (cit. on pp. 27, 28).

[Hoc91] Sepp Hochreiter. "Untersuchungen Zu Dynamischen Neuronalen Netzen". Diploma. Technische Universität München, 1991 (cit. on p. 28).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cit. on pp. 28, 70).

[MSA22] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "M5 Accuracy Competition: Results, Findings, and Conclusions". In: *International Journal of Forecasting* 38 (2022), pp. 1346–1364. DOI: 10.1016/j.ijforecast.2021.11.013 (cit. on p. 29).

[GOV22] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. "Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?" In: *Advances in Neural Information Processing Systems*. 2022, pp. 507–520 (cit. on p. 29).

[Bre+84] Leo Breiman et al. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984. DOI: 10.1201/9781315139470 (cit. on p. 29).

[Ped+11] Fabian Pedregosa et al. "Scikit-Learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 29).

[Bre96] Leo Breiman. "Bagging Predictors". In: *Machine Learning* 24 (1996), pp. 123–140. DOI: 10.1007/BF00058655 (cit. on p. 31).

[Bre01] Leo Breiman. "Random Forests". In: *Machine Learning* 45 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324 (cit. on pp. 31, 32).

[Fri01] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *Annals of Statistics* 29 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451 (cit. on pp. 31, 32).

[Fri37] Milton Friedman. "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance". In: *Journal of the American Statistical Association* 32 (1937), pp. 675–701. DOI: 10.1080/01621459.1937.10503522 (cit. on p. 35).

[BCM16] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. "Should We Really Use Post-Hoc Tests Based on Mean-Ranks?" In: *Journal of Machine Learning Research* 17 (2016), pp. 152–161 (cit. on p. 35).

[Wil45] Frank Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1 (1945), pp. 80–83. DOI: 10.2307/3001968 (cit. on p. 35).

[Dem06] Janez Demšar. "Statistical Comparisons of Classifiers over Multiple Data Sets". In: *Journal of Machine Learning Research* 7 (2006), pp. 1–30 (cit. on p. 35).

[Gui+18] Riccardo Guidotti et al. "A Survey of Methods for Explaining Black Box Models". In: *ACM Computing Surveys* 51 (2018), pp. 1–42. DOI: 10.1145/3236009 (cit. on pp. 37–39).

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016. DOI: 10.1145/2939672.2939778 (cit. on p. 37).

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic Attribution for Deep Networks". In: *Proceedings of the International Conference on Machine Learning*. Aug. 2017, pp. 3319–3328 (cit. on pp. 37, 38).

[Ate+20] Emre Ates et al. "Counterfactual Explanations for Machine Learning on Multivariate Time Series Data". In: *arXiv:2008.10781 [cs, stat]* (2020) (cit. on p. 38).

[DGK20] Eoin Delaney, Derek Greene, and Mark T. Keane. "Instance-Based Counterfactual Explanations for Time Series Classification". In: *arXiv:2009.13211 [cs, stat]* (2020) (cit. on p. 38).

[AB18] Amina Adadi and Mohammed Berrada. "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)". In: *IEEE Access* 6 (2018), pp. 52138–52160. DOI: 10.1109/ACCESS.2018.2870052 (cit. on p. 38).

[Ade+18] Julius Adebayo et al. "Sanity Checks for Saliency Maps". In: *Advances in Neural Information Processing Systems*. 2018, pp. 9505–9515 (cit. on pp. 38, 47).

[Kön+21] Gunnar König et al. “Relative Feature Importance”. In: *Proceedings of the International Conference on Pattern Recognition*. Jan. 2021, pp. 9318–9325. DOI: 10.1109/ICPR48806.2021.9413090 (cit. on p. 38).

[Smi+17] Daniel Smilkov et al. “SmoothGrad: Removing Noise by Adding Noise”. In: *arXiv:1706.03825 [cs, stat]* (2017) (cit. on pp. 38, 39).

[Liu+21] Changliu Liu et al. “Algorithms for Verifying Deep Neural Networks”. In: *Foundations and Trends® in Optimization* 4 (2021), pp. 244–404. DOI: 10.1561/2400000035 (cit. on p. 39).

[Rom+13] Pablo Romeu et al. “Time-Series Forecasting of Indoor Temperature Using Pre-trained Deep Neural Networks”. In: *Artificial Neural Networks and Machine Learning*. 2013, pp. 451–458. DOI: 10.1007/978-3-642-40728-4_57 (cit. on p. 43).

[Tai+12] Souhaib Ben Taieb et al. “A Review and Comparison of Strategies for Multi-Step Ahead Time Series Forecasting Based on the NN5 Forecasting Competition”. In: *Expert Systems with Applications* 39 (2012), pp. 7067–7083. DOI: 10.1016/j.eswa.2012.01.039 (cit. on p. 43).

[LPP20] Ioannis E. Livieris, Emmanuel Pintelas, and Panagiotis Pintelas. “A CNN–LSTM Model for Gold Price Time-Series Forecasting”. In: *Neural Computing and Applications* 32 (2020), pp. 17351–17360. DOI: 10.1007/s00521-020-04867-x (cit. on p. 43).

[DIA17] Konstantinos Demertzis, Lazaros Iliadis, and Vardis-Dimitris Anezakis. “A Deep Spiking Machine-Hearing System for the Case of Invasive Fish Species”. In: *Proceedings of the International Conference on Innovations in Intelligent Systems and Applications*. 2017, pp. 23–28. DOI: INISTA.2017.8001126 (cit. on p. 43).

[US02] Paul E. Utgoff and David J. Stracuzzi. “Many-Layered Learning”. In: *Neural Computation* 14 (2002), pp. 2497–2529. DOI: 10.1162/08997660260293319 (cit. on p. 43).

[Hon+20] Tao Hong et al. “Energy Forecasting: A Review and Outlook”. In: *IEEE Open Access Journal of Power and Energy* 7 (2020), pp. 376–388. DOI: 10.1109/OAJPE.2020.3029979 (cit. on p. 43).

[SGO20] Omer B. Sezer, Mehmet U. Gudelek, and Ahmet M. Ozbayoglu. “Financial Time Series Forecasting with Deep Learning : A Systematic Literature Review: 2005–2019”. In: *Applied Soft Computing* 90 (2020). DOI: 10.1016/j.asoc.2020.106181 (cit. on p. 43).

[LA23] Sichen Li and Andreas Adelmann. “Time Series Forecasting Methods and Their Applications to Particle Accelerators”. In: *Physical Review Accelerators and Beams* 26 (2023). DOI: 10.1103/PhysRevAccelBeams.26.024801 (cit. on p. 43).

[Gam17] John Gamboa. “Deep Learning for Time-Series Analysis”. In: *arXiv:1701.01887 [cs]* (2017). DOI: 10.48550/arXiv.1701.01887 (cit. on p. 43).

[WM97] David H. Wolpert and William G. Macready. “No Free Lunch Theorems for Optimization”. In: *IEEE Transactions on Evolutionary Computation* 1 (1997), pp. 67–82. DOI: 10.1109/4235.585893 (cit. on p. 43).

[Cer+17a] Vítor Cerqueira et al. "Arbitrated Ensemble for Time Series Forecasting". In: *Machine Learning and Knowledge Discovery in Databases*. 2017, pp. 478–494. DOI: 10.1007/978-3-319-71246-8_29 (cit. on pp. 43, 45, 51, 52, 60, 69, 110).

[Cer+19] Vítor Cerqueira et al. "Arbitrage of Forecasting Experts". In: *Machine Learning* 108 (2019), pp. 913–944. DOI: 10.1007/s10994-018-05774-y (cit. on pp. 43, 45, 51, 55, 60).

[AS19] Roy Assaf and Anika Schumann. "Explainable Deep Neural Networks for Multivariate Time Series Predictions". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. 2019, pp. 6488–6490. DOI: 10.24963/ijcai.2019/932 (cit. on pp. 44, 47).

[BM01] Lucien Birgé and Pascal Massart. "Gaussian Model Selection". In: *Journal of the European Mathematical Society* 3 (2001), pp. 203–268. DOI: Gaussianmodelselection (cit. on p. 44).

[AGP10] Raffaele Argiento, Alessandra Guglielmi, and Antonio Pievatolo. "Bayesian Density Estimation and Model Selection Using Nonparametric Hierarchical Mixtures". In: *Computational Statistics & Data Analysis* 54 (2010), pp. 816–832. DOI: 10.1016/j.csda.2009.11.002 (cit. on p. 44).

[RP99] Isabelle Rivals and Léon Personnaz. "On Cross Validation for Model Selection". In: *Neural Computation* 11 (1999), pp. 863–870. DOI: 10.1162/089976699300016476 (cit. on pp. 45, 51).

[Wol92] David H. Wolpert. "Stacked Generalization". In: *Neural Networks* 5 (1992), pp. 241–259. DOI: 10.1016/S0893-6080(05)80023-1 (cit. on pp. 45, 53).

[Jac+91] Robert A. Jacobs et al. "Adaptive Mixtures of Local Experts". In: *Neural Computation* 3 (1991), pp. 79–87. DOI: 10.1162/neco.1991.3.1.79 (cit. on pp. 45, 53).

[Cer+17b] Vítor Cerqueira et al. "Dynamic and Heterogeneous Ensembles for Time Series Forecasting". In: *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics*. 2017, pp. 242–251. DOI: 10.1109/DSAA.2017.26 (cit. on pp. 45, 110).

[Zho+16] Bolei Zhou et al. "Learning Deep Features for Discriminative Localization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2921–2929 (cit. on p. 47).

[BC94] Donald J. Berndt and James Clifford. "Using Dynamic Time Warping to Find Patterns in Time Series". In: *Proceedings of the International Conference on Knowledge Discovery and Data Mining*. 1994, pp. 359–370. DOI: 10.2307/2282952 (cit. on p. 48).

[Hoe63] Wassily Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58 (1963), pp. 13–30 (cit. on p. 49).

[DG17] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017 (cit. on pp. 52, 69).

[SA81] Thomas Stoffel and Afshin Andreas. "NREL Solar Radiation Research Laboratory (SRRL): Baseline Measurement System (BMS); Golden, Colorado (Data)". In: (July 1981). DOI: 10.7799/1052221 (cit. on p. 52).

[MSA20] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "The M4 Competition: 100,000 Time Series and 61 Forecasting Methods". In: *International Journal of Forecasting* 36 (2020), pp. 54–74. DOI: 10.1016/j.ijforecast.2019.04.014 (cit. on pp. 52, 99).

[GES01] Felix A. Gers, Douglas Eck, and Jürgen Schmidhuber. "Applying LSTM to Time Series Predictable through Time-Window Approaches". In: *Proceedings of the International Conference on Artificial Neural Networks*. 2001, pp. 669–676. DOI: 10.1007/3-540-44668-0_93 (cit. on pp. 51, 70).

[MD97] Dragos D. Margineantu and Thomas G. Dietterich. "Pruning Adaptive Boosting". In: *Proceedings of the International Conference on Machine Learning*. 1997, pp. 211–218 (cit. on p. 59).

[LYZ12] Nan Li, Yang Yu, and Zhi-Hua Zhou. "Diversity Regularized Ensemble Pruning". In: *Machine Learning and Knowledge Discovery in Databases*. 2012, pp. 330–345. DOI: 10.1007/978-3-642-33460-3_27 (cit. on pp. 59, 60).

[BWT05] Gavin Brown, Jeremy L. Wyatt, and Peter Tiňo. "Managing Diversity in Regression Ensembles". In: *Journal of Machine Learning Research* 6 (2005), pp. 1621–1650 (cit. on pp. 59, 62).

[MHS08] Gonzalo Martínez-Muñoz, Daniel Hernández-Lobato, and Alberto Suárez. "An Analysis of Ensemble Pruning Techniques Based on Ordered Aggregation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31 (2008), pp. 245–259. DOI: 10.1109/TPAMI.2008.78 (cit. on pp. 59, 60).

[Car+04] Rich Caruana et al. "Ensemble Selection from Libraries of Models". In: *Proceedings of the International Conference on Machine Learning*. 2004 (cit. on p. 59).

[ZWT02] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. "Ensembling Neural Networks: Many Could Be Better than All". In: *Artificial Intelligence* 137 (2002), pp. 239–263. DOI: 10.1016/S0004-3702(02)00190-X (cit. on p. 60).

[Zha02] Tong Zhang. "Covering Number Bounds of Certain Regularized Linear Function Classes". In: *Journal of Machine Learning Research* 2 (2002), pp. 527–550 (cit. on p. 60).

[CTY09] Huanhuan Chen, Peter Tiňo, and Xin Yao. "Predictive Ensemble Pruning by Expectation Propagation". In: *IEEE Transactions on Knowledge and Data Engineering* 21 (2009), pp. 999–1013. DOI: 10.1109/TKDE.2009.62 (cit. on p. 60).

[PTV12] Ioannis Partalas, Grigorios Tsoumakas, and Ioannis Vlahavas. *A Study on Greedy Algorithms for Ensemble Pruning*. Technical Report. Aristotle University of Thessaloniki, Greece, 2012 (cit. on p. 60).

[TPV09] Grigorios Tsoumakas, Ioannis Partalas, and Ioannis Vlahavas. "An Ensemble Pruning Primer". In: *Applications of Supervised and Unsupervised Ensemble Methods*. 2009, pp. 1–13 (cit. on p. 60).

[KK15] Alexey V. Krikunov and Sergey V. Kovalchuk. "Dynamic Selection of Ensemble Members in Multi-Model Hydrometeorological Ensemble Forecasting". In: *Procedia Computer Science* 66 (2015), pp. 220–227. DOI: 10.1016/j.procs.2015.11.026 (cit. on p. 60).

[ZDY21] Jing Zhang, Qun Dai, and Changsheng Yao. "DEP-TSP Meta: A Multiple Criteria Dynamic Ensemble Pruning Technique Ad-Hoc for Time Series Prediction". In: *International Journal of Machine Learning and Cybernetics* 12 (2021), pp. 2213–2236. DOI: 10.1007/s13042-021-01302-y (cit. on p. 60).

[STM21] Amal Saadellah, Maryam Tavakol, and Katharina Morik. "An Actor-Critic Ensemble Aggregation Model for Time-Series Forecasting". In: *Proceedings of the International Conference on Data Engineering*. 2021. DOI: 10.1109/ICDE51399.2021.00233 (cit. on p. 60).

[GG16] Pierre Gaillard and Yannig Goude. *Opera: Online Prediction by Expert Aggregation*. Manual. 2016 (cit. on p. 60).

[SM21] Amal Saadallah and Katharina Morik. "Online Ensemble Aggregation Using Deep Reinforcement Learning for Time Series Forecasting". In: *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics*. 2021. DOI: 10.1109/DSAA53316.2021.9564132 (cit. on p. 60).

[KV95] Anders Krogh and Jesper Vedelsby. "Neural Network Ensembles, Cross Validation, and Active Learning". In: *Advances in Neural Information Processing Systems*. 1995 (cit. on p. 62).

[God+21] Rakshitha Godahewa et al. "Monash Time Series Forecasting Archive". In: *Neural Information Processing Systems Track on Datasets and Benchmarks*. 2021 (cit. on pp. 68, 69, 87, 88, 104, 105).

[He+16] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on pp. 69, 120).

[MA14] Ahmad Mozaffari and Nasser L. Azad. "Optimally Pruned Extreme Learning Machine with Ensemble of Regularization Techniques and Negative Correlation Penalty Applied to Automotive Engine Coldstart Hydrocarbon Emission Identification". In: *Neurocomputing* 131 (2014), pp. 143–156. DOI: 10.1016/j.neucom.2013.10.030 (cit. on p. 71).

[Saa+20] Amal Saadallah et al. "BRIGHT—Drift-Aware Demand Predictions for Taxi Networks". In: *IEEE Transactions on Knowledge and Data Engineering* 32 (2020), pp. 234–245. DOI: 10.1109/TKDE.2018.2883616 (cit. on pp. 72, 92).

[Zin03] Martin Zinkevich. "Online Convex Programming and Generalized Infinitesimal Gradient Ascent". In: *Proceedings of the International Conference on Machine Learning*. 2003 (cit. on p. 72).

[CL06] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511546921 (cit. on p. 72).

[Win17] Olivier Wintenberger. "Optimal Learning with Bernstein Online Aggregation". In: *Machine Learning* 106 (2017), pp. 119–141. DOI: 10.1007/s10994-016-5592-6 (cit. on p. 72).

[Sha53] Lloyd S. Shapley. "A Value for N-Person Games". In: *Contributions to the Theory of Games* 2 (1953), pp. 307–317. DOI: 10.1515/9781400881970-018 (cit. on pp. 79–81).

[You85] Hobart P. Young. "Monotonic Solutions of Cooperative Games". In: *International Journal of Game Theory* 14 (1985), pp. 65–72. DOI: 10.1007/BF01769885 (cit. on p. 81).

[AJL21] Kjersti Aas, Martin Jullum, and Anders Løland. "Explaining Individual Predictions When Features Are Dependent: More Accurate Approximations to Shapley Values". In: *Artificial Intelligence* 298 (2021), p. 103502. DOI: 10.1016/j.artint.2021.103502 (cit. on p. 81).

[ŠK14] Erik Štrumbelj and Igor Kononenko. "Explaining Prediction Models and Individual Predictions with Feature Contributions". In: *Knowledge and Information Systems* 41 (2014), pp. 647–665. DOI: 10.1007/s10115-013-0679-x (cit. on pp. 81, 82).

[PB97] R. Kelley Pace and Ronald Barry. "Sparse Spatial Autoregressions". In: *Statistics & Probability Letters* 33 (May 1997), pp. 291–297. DOI: 10.1016/S0167-7152(96)00140-X (cit. on p. 82).

[LP22] Gabriel Laberge and Yann Pequignot. "Understanding Interventional TreeSHAP : How and Why It Works". In: *arXiv:2209.15123 [cs]* (2022). DOI: 10.48550/arXiv.2209.15123 (cit. on pp. 83, 84).

[KKK16] Been Kim, Rajiv Khanna, and Oluwasanmi O. Koyejo. "Examples Are Not Enough, Learn to Criticize! Criticism for Interpretability". In: *Advances in Neural Information Processing Systems* 29 (2016) (cit. on p. 93).

[Smy20] Slawek Smyl. "A Hybrid Method of Exponential Smoothing and Recurrent Neural Networks for Time Series Forecasting". In: *International Journal of Forecasting* 36 (2020), pp. 75–85. DOI: 10.1016/j.ijforecast.2019.03.017 (cit. on p. 99).

[HK19] Zahra Hajirahimi and Mehdi Khashei. "Hybrid Structures in Time Series Modeling and Forecasting: A Review". In: *Engineering Applications of Artificial Intelligence* 86 (2019), pp. 83–106. DOI: 10.1016/j.engappai.2019.08.018 (cit. on p. 99).

[Zha03] G. Peter Zhang. "Time Series Forecasting Using a Hybrid ARIMA and Neural Network Model". In: *Neurocomputing* 50 (2003), pp. 159–175. DOI: 10.1016/S0925-2312(01)00702-0 (cit. on p. 99).

[BC23] Marius Brehler and Lucas Camphausen. "Combining Decision Tree and Convolutional Neural Network for Energy Efficient On-Device Activity Recognition". In: *Proceedings of the IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip*. Dec. 2023. DOI: 10.1109/MCSoC60832.2023.00035 (cit. on p. 100).

[DPP22] Francesco Daghero, Daniele Jahier Pagliari, and Massimo Poncino. "Two-Stage Human Activity Recognition on Microcontrollers with Decision Trees and CNNs". In: *Proceedings of the Conference on Ph.D Research in Microelectronics and Electronics*. June 2022. DOI: 10.1109/PRIME55000.2022.9816745 (cit. on p. 100).

[Bus24] Sebastian Buschjäger. "Rejection Ensembles with Online Calibration". In: *Machine Learning and Knowledge Discovery in Databases: Research Track*. 2024. DOI: 10.1007/978-3-031-70365-2_1 (cit. on p. 100).

[SFG19] David Salinas, Valentin Flunkert, and Jan Gasthaus. "DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks". In: *arXiv:1704.04110 [cs]* (2019). DOI: 10.48550/arXiv.1704.04110 (cit. on p. 105).

[FS10] George Forman and Martin Scholz. "Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement". In: *ACM SIGKDD Explorations Newsletter* 12 (2010), pp. 49–57. DOI: 10.1145/1882471.1882479 (cit. on p. 108).

[Saa23] Amal Saadallah. "Online Explainable Model Selection for Time Series Forecasting". In: *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics*. 2023. DOI: 10.1109/DSAA60987.2023.10302609 (cit. on p. 111).

[Pas+23] Sepideh Pashami et al. "Explainable Predictive Maintenance". In: *arXiv:2306.05120 [cs]* (2023). DOI: 10.48550/arXiv.2306.05120 (cit. on p. 115).

[Vel+23] Bruno Veloso et al. *MetroPT2: A Benchmark Dataset for Predictive Maintenance*. 2023. DOI: 10.5281/zenodo.7766691 (cit. on pp. 116, 117, 126).

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Computing Surveys* 41 (2009), pp. 1–58. DOI: 10.1145/1541880.1541882 (cit. on p. 118).

[Pan+21] Guansong Pang et al. "Deep Learning for Anomaly Detection: A Review". In: *ACM Computing Surveys* 54 (2021), pp. 1–38. DOI: 10.1145/3439950 (cit. on p. 118).

[Gam+24] João Gama et al. "From Fault Detection to Anomaly Explanation: A Case Study on Predictive Maintenance". In: *Journal of Web Semantics* 81 (2024), p. 100821. DOI: 10.1016/j.websem.2024.100821 (cit. on pp. 118, 119).

[Dav+21] Narjes Davari et al. "Predictive Maintenance Based on Anomaly Detection Using Deep Learning for Air Production Unit in the Railway Industry". In: *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics*. 2021. DOI: 10.1109/DSAA53316.2021.9564181 (cit. on p. 118).

[DGB16] João Duarte, João Gama, and Albert Bifet. "Adaptive Model Rules From High-Speed Data Streams". In: *ACM Transactions on Knowledge Discovery from Data* 10 (2016), pp. 1–22. DOI: 10.1145/2829955 (cit. on p. 119).

[Rib+23] Rita P. Ribeiro et al. "Online Anomaly Explanation: A Case Study on Predictive Maintenance". In: *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. 2023. DOI: 10.1007/978-3-031-23633-4_25 (cit. on p. 119).

[ARG21] Ehsan Aminian, Rita P. Ribeiro, and João Gama. "Chebyshev Approaches for Imbalanced Data Streams Regression Models". In: *Data Mining and Knowledge Discovery* 35 (2021), pp. 2389–2466. DOI: 10.1007/s10618-021-00793-1 (cit. on p. 119).

[RPG16] Rita P. Ribeiro, Pedro Pereira, and João Gama. "Sequential Anomalies: A Study in the Railway Industry". In: *Machine Learning* 105 (2016), pp. 127–153. DOI: 10.1007/s10994-016-5584-6 (cit. on p. 120).

[GK14] João Gama and Petr Kosina. “Recurrent Concepts in Data Streams Classification”. In: *Knowledge and Information Systems* 40 (2014), pp. 489–507. DOI: 10.1007/s10115-013-0654-6 (cit. on p. 121).

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *Proceedings of the International Conference on Learning Representations*. 2015 (cit. on p. 123).

[Vel+22] Bruno Veloso et al. “The MetroPT Dataset for Predictive Maintenance”. In: *Scientific Data* 9 (2022). DOI: 10.1038/s41597-022-01877-3 (cit. on p. 127).