Explainable Adaptation of Time Series Forecasting

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Amal Saadallah

Dortmund

2022

Tag der mündlichen Prüfung: 12. Dezember 2022

Dekan/Dekanin: Prof. Dr.-Ing. Gernot Fink

Gutachterinnen:

- Prof. Dr. Katharina Morik
- Prof. Dr. Barbara Hammer

Abstract

A time series consists of data points measured over time. This type of data is omnipresent in many fields, such as healthcare, manufacturing, and transportation, to name but a few. This is due to the dynamic nature of numerous real-world phenomena, where related events naturally happen and evolve over time. This also explains why this data type has always attracted the attention of both academic and industrial communities. Many of these phenomena are described by multiple inter-dependent time series forming thus a Multivariate Time Series (MTS). Accurately predicting the future behavior of a time series is crucial for many applications as predictions are usually required for a decision-making process. The process of predicting the future behavior of a time series is referred to as time series forecasting. Forecasting allows anticipating the behavior of a time series by professionals which enables them to take proactive measures in different tasks such as resource planning and management, security measures setting, process adjustment, etc. Several Machine Learning (ML) models have been developed or applied to solve the forecasting task by learning from historical time series data.

In an online-learning setting, time series observations are incrementally acquired and the distributions from which they are drawn may keep changing over time. This phenomenon is widely known as concept drift. Such changes may affect the generalization of learned ML models to future data. To cope with this dynamic behavior, online adaptive forecasting methods are required. In this context, the aim of this thesis is to extend the State-of-the-Art (SoA) in the ML literature for time series forecasting. More specifically, our research goal can be divided into two main parts: (i) forecasting the future numerical values of time series; (ii) performing model-based quality prediction in a timely manner. To reach these goals, we develop novel methods for time series forecasting in an online adaptive manner to changes in time series data as well as forecasting models' performance. In the first part, we focus on the task of online time series forecasting. First, we devise a framework for the online selection of time series variables composing MTS data that should be considered as input to a given forecasting model. The selection procedure is updated adaptively following concept drift detection in the dependence structure between these variables that evolve over time and in the performance of the MTS forecasting model. The framework is fully automated using a meta-learning schema carefully devised for this task. In addition to being affected by changes in the time series, it has also been proven that none of the ML forecasting models is universally valid for every application. This can be explained by the fact that different models have varying regions of expertise or so-called Regions of Competence (RoCs) over the time series. We then suggest a novel method for online single model selection. We also extend the RoCs assumption and highlight the usefulness of combining several models into one single model, i.e. an ensemble model. The ensemble model building includes three main stages. First, ensemble members' generation, i.e. individual models composing the ensemble, is performed. Second, some of these models get selected. This stage is known in the ML literature as ensemble pruning. Third, the selected model are aggregated into one single model. We aim to make ensemble construction online and adaptive to the changes in the time series and forecasting models' performance over time. Therefore, we develop two methods for online ensemble pruning and one novel method for online ensemble aggregation using the Deep Reinforcement Learning paradigm. The methods are updated in an informed manner following concept drift detection under different perspectives. In addition, we present two additional methods for both online single model selection and ensemble pruning that are specifically devised for Deep Neural Networks. We also promote the explainability of both model selection and ensemble pruning processes as well as models' performance. We show the usefulness of the developed methods empirically, using a large set of time series from different application domains.

Regarding the second part, we contribute to the literature on online ML model-based quality prediction for three different Industry 4.0 applications, namely a machining process using NC-milling, a bolt installation process in the automotive industry, and Surface Mount Technology (SMT) in electronics manufacturing. In NC-milling process monitoring, in addition to real-time quality prediction using our developed methods on time series sensor data, we show how process simulation can be used to generate additional knowledge and how such knowledge can be integrated efficiently into the ML process. Based on the results in this case study, we conclude that our developed methods are competitive with SoA approaches. Then, we present two applications of explainable model-based quality prediction and their impact on smart industry practices.

Acknowledgements

This thesis couldn't have been created without support. First of all, I'd like to express my deep gratitude to my supervisor, Prof. Dr. Katharina Morik. Finishing this thesis would not be possible without her continuous support, guidance, valuable input, encouragement, and never-ending trust in my person. A special word to her, for believing in me and taking me as a student.

Also, I'd like to thank the committee members of this thesis, Prof. Dr. Barbara Hammer and Prof. Dr. Thomas Liebig, and the head of the committee Prof. Dr. Petra Wiederkehr. I thank all my colleagues at the Artificial Intelligence Group for the many interesting collaborations, discussions, and all the fun we had together. I am grateful to all of them. I also thank my colleagues in project B3 of the Collaborative Research Center SFB 876, in particular, Felix Finkeldey and Jan Büscher, for the intensive collaboration on data acquisition, storage, and preprocessing for the industrial use cases. I further recognize that my research would not have been possible without the financial support from the Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876, project B3.

I thank also Dr. Luis Moreira-Matias with whom I worked on the first research work on taxi-demand prediction and who encouraged me to dive further into research.

Finally, I acknowledge the love and support from my family and friends, in particular my mother Saloua. I thank you very much for all of the support, the friend, and love you have given to me!

Table of Contents

List of Figures

List of Tables

Part I

Introduction

1 Introduction

1.1 Context

In several scientific fields, measurements are acquired over time. These measurements most often comprehend some temporal dependency among them. In other words, the currently observed measurement value may depend on the previously captured values. In addition, they lead to a collection of chronologically ordered data termed time series. With the increasing development of data collection and storage technologies in this modern era of information, massive amounts of time series data become more and more available [1], [2]. Much of the world's data supply is in the form of time series [3], [4]. Due to this profusion, learning from time series data is one of the most active research topics in the data science community so far [2], [3], [5]. In addition, the time-changing nature of many real-world phenomena has contributed to this generalized interest in time series. For example, weather-related variables are tracked over time, and some are continuously predicted to provide information people and organizations can use to enhance societal benefits and reduce losses, including protecting lives and properties and ensuring safety [6]. Likewise, in finance, economists monitor stock market indices over time to help investors compare current stock price levels with past prices and accurately estimate future market performance to look for appropriate investment opportunities [7]. For some industrial processes, time series sensor data is continuously acquired and analyzed to ensure sustainable zero-defect processes [8].

Uncertainty is a common issue in most scenarios involving learning from time series data, which challenges the accurate understanding of its future behavior [9]. Such understanding requires rigorous time series analysis [5]. One of the main tasks in time series data analysis is forecasting, which denotes estimating the future values of a time series. Forecasting has always attracted the attention of academic and industrial communities as it has always been considered one of the main steps for real-time decision-making and planning in a wide variety of fields [10], [11]. It covers also a big range of applications, including traffic prediction [12], [13], weather forecasts [14], and stock market prices prediction [7], [15], to name but a few. Nonetheless, forecasting has also been revealed to be one of the most challenging tasks in time series analysis due to the dynamic behavior of time series data, which may involve non-stationary and complex processes and are therefore most often subject to concept drifts [9], [11], [16]. In other words, the underlying generating process of the time series observations may change significantly over time. Thus, forecasting models trained using old historical data may become obsolete. This task becomes more and more arduous when the time series data is represented by multiple time-dependent time series variables, creating thus a Multivariate Time Series (MTS) data. This is often the case in rapidly growing digital environments, and Internet-of-Things systems [17], [18]. The evolution of MTS is spatio-temporal, along with the different variables and over time, respectively. On the one hand, this data represents an enriched form of information about the application. On the other hand, the number of these variables can increase drastically and might include irrelevant and redundant ones. This may heighten the curse of dimensionality. Therefore, it is necessary to carefully select the most important time series variables as a preparation step before forecasting. Again, this spatio-temporal data may involve multiple non-stationary processes, and the dependencies along its composing variables may also follow a non-stationary process. As a result, the relation between time series variables might change significantly over time, and the offline input variable selection procedures may also become invalid over time. Therefore, selecting time series variables should cope with the evolving nature of the spatio-temporal dependencies in the MTS data.

Several Machine Learning (ML) models have been successfully applied to solve the time series forecasting task. In general, ML uses data generated by some process to learn the parameters of a function or model that depends on this data. Learning is driven by some specified loss function that measures in a well-defined way how well the parameters fit the data. A "good" choice of these parameters is often determined using a numerical optimization which denotes the process of searching the loss function's minimum by exploiting information from its derivatives. ML models can be grouped into different types depending on their outcome. The main three most popular types include Supervised Learning, where the model maps an input to an output based on a sample of input-output pairs forming a labeled dataset, Unsupervised Learning, where no label is given to the model and only a set of inputs is modeled, and Reinforcement Learning where the model learns a policy of how to act given an observation of the world and every action has some impact in the environment and the environment provides feedback that guides the learning algorithm. Time series forecasting can be framed as a Supervised Learning problem. Hence, an ML model solves the forecasting task either by considering as input the whole or some window of an ordered sequence of time series observations in an offline or a streaming fashion or by using time series embedding into a high-dimensional Euclidean space that reformulates forecasting as a classical regression task by treating each time series observation as the target or the dependent variable and its corresponding subsequence of a given number of lagged observations as the explanatory variables [9], [19]. Note that time series forecasting should be distinguished from time series regression, where the goal is to predict continuous scalar values from a time series, the prediction does not necessarily depend on recent values, and the target may not necessarily come from the same series. The time series regression task is closely related to time series classification, which aims to learn the relationship between a set of time series and a categorical class label [20]. Note also that we are tackling the forecasting task using an online learning schema, i.e., time series data is assumed to become available in sequential order and can be used immediately to update the ML model or the forecasting framework in general for future data at each time step.

Nevertheless, it is generally accepted that none of these ML models is universally valid for every forecasting application. This seems to be a particular case of the No Free Lunch theorem by Wolpert [21], which means no learning algorithm is best suited for all learning tasks. Even if we consider a single application, forecasting models' performance is time-dependent [9], [10], [22]. This can be explained by the fact that different forecasting models have different areas of expertise or so-called Regions of Competence (RoCs) placed over different parts of the input time series [9], [22]. These regions evolve over time. Therefore, forecasting model selection should be able to cope with changes in the time series and adapt to new concepts automatically and efficiently. In other words, online and adaptive model selection is often required to cope with the time-evolving nature of time series and the fact that forecasting models have a certain expected level of competence in predicting a particular region in the time series.

While some works have focused on the online selection of a single model [23], others have extended on the assumption that no single model is expert all the time and have proposed to adaptively combine multiple forecasting models into a single model using an ensemble technique [9], [10], [16], [22]. The individual models that make up the ensemble are often called ensemble members. Ensemble members must also be carefully selected in an online adaptive manner to account for changes in both the time series and the time-changing forecasting performance of the members. This selection is referred to as online ensemble pruning, in which only a subset of these members is retained at each time instant [24], [25].

For online single model selection, estimating the error densities of candidate models, either using approximations [26] or empirical estimates [27] is most often employed. These methods are impractical and sometimes ineffective in the context of forecasting. This is primarily because the composite densities for the error function of the target and estimated time series values must be continuously approximated over time in the case of using approximations. Moreover, the results depend largely on the quality of the approximation. When empirical assessments of the errors are used, the fact that the estimated empirical error of a model is usually lower than its true error can lead to an erroneous selection. Model selection can also be performed using meta-learning to learn an appropriate selection strategy [9], [16], [22] .

Online model selection becomes more and more crucial when selecting many models to appear in an ensemble, not only to adapt to the changes in the time series and models' performance but also to optimize resources adequately for the corresponding application. Hence, an ensemble with a huge number of models may add a lot of computational overhead due to the large memory requirements of some of its members, e.g. decision trees [28]. The optimization of run-time overhead is imperative for some specific applications where real-time requirements must be met, such as in online forecasting with a short time duration between two subsequent time steps. Nonetheless, selecting ensemble members (i.e., pruning) is inherently challenging. Hence, given a set of trained forecasters, it is difficult to select the sub-ensemble with the best generalization performance because it is not easy to estimate the generalization error of the sub-ensemble similarly to a single model. Moreover, finding the optimal subset of models is a combinatorial search problem with exponential computational complexity. Therefore, it is impossible to compute the exact solution by exhaustive search, and an approximate search is required [29], [30]. Ensemble pruning has been widely studied in the ML literature for classification problems [29], [30]. Few works tackled this task for time series forecasting, especially in an online dynamic fashion [10]. Once ensemble members are selected in a timely manner, a decision about how to aggregate them into one single model, i.e., the ensemble model, must also be made in real-time. Ensemble aggregation strategies can be categorized into three main families. The first group is based on voting schemes that use either a majority or a (weighted) average of the votes to decide the final output (e.g., bagging [31]). The second family is based on cascading, where several forecasting models are concatenated using all information collected from the output from a given forecasting model as additional information for the next model in the cascade [32]. The third category is based on a stacking technique [33], and meta-learning is frequently used to learn the aggregation rule of actual outputs from prior experiences. A machine-learning algorithm is trained on the ensemble members' outputs as explanatory variables joined the target time series values to learn an aggregation rule [34]. However, there is no single best way to estimate the weights of each model in a linearly weighted ensemble. Thus, learning the optimal ensemble aggregation strategy remains an open research question [9], [35], [36].

Similar issues apply to MTS forecasting. Therefore, in addition to selecting time series variables that should cope with the time-evolving nature of the spatio-temporal dependencies in the MTS data, adequate model selection is also required to cope with the time-changing characteristics of the MTS.

As explained above, several decisions must be made in forecasting, ranging from time series variables selection and online model selection to the online aggregation of many models. Machine Learning is used to develop frameworks that are able to automate these decisions fully. However, to maximize the benefits of these ML-based frameworks while simultaneously mitigating or even preventing their risks and dangers, the concept of *trustworthy ML* has appeared to promote the idea that all the parts involved in an ML application, can benefit from the full potential of ML if trust can be established in its development, deployment, and usage [37], [38]. Several requirements have to be fulfilled to build ML-based trustworthy solutions. According to the European Commission High-Level Expert Group on Artificial intelligence (AI) [38], seven key requirements AI systems should meet in order to be deemed trustworthy. Among them, three are related to the explainability of AI systems. Explainable Artificial Intelligence (XAI) describes methods that generate interpretations for decisions and provide human-understandable representations for solutions developed by AI [39]. More particularly, explainability in ML is the process of explaining to a human why and how an ML model made a decision [40]. In this context, explainability means the algorithm and its decision or output can be understood by a human. This is an important concept to make ML models transparent and solves the black-box problem. A Black-box model refers to ML models that are sufficiently complex such that they can not be straightforwardly understood by humans [41]. In the context of model selection, from a mathematical viewpoint, it can be seen that while selection made by "simple" rules, such as deciding between two forecasting models based on a single threshold on the time series values, provides high explainability but possibly, limited forecasting accuracy, "more complex" model selection methods, such as meta-learning using neural networks provide high predictive accuracy at the expense of limited explainability. To solve this trade-off, it is necessary to devise adequate model selection explainability tools.

Combining online adaptive methods for time series forecasting with adequate explainability tools helps in making a step toward building trustworthy ML solutions necessary in a wide variety of fields such as Industry 4.0, i.e., the fourth industrial revolution characterized by "interconnectedness, data, and intelligence" [42]. In this revolution, adaptable, smart, and reconfigurable manufacturing processes are developed by utilizing computer-integrated manufacturing, and digital information technologies. AI is also incorporated to allow manufacturers to learn from data and create more connected and intelligent industrial practices [43]. One of the most important tasks in the context of smart manufacturing is the real-time process quality prediction using ML models [44]. This task is also known as model-based quality prediction where an ML model is trained to map the available quality-related input information and data, e.g., master data, operating states, or process parameters, to the resulting product quality [45], [46]. Nevertheless, there are still some limitations in the way of ML adaption and application in real-world industrial environments. The most challenging obstacle is the lack of reliable labeled data sources in many manufacturing scenarios. One solution to this issue is using synthetic data sources such as process simulations. In addition, simulations represent expert knowledge; hence, integrating simulation into ML may help achieve better predictions. This integration can be achieved by fusing simulation and real-world data, e.g., sensor data, for ML applications. The combination of simulation and ML has been successfully used for optimizing many industrial processes such as Numerically Controlled (NC)-milling [47]. However, deciding the optimal way to fuse real-world and simulation data is still an open research question.

1.2 Goals and Research Questions

The previous section highlighted how time series forecasting plays a significant role in decision-making across several domains. Then, we outlined the main challenges faced when solving the forecasting task. More precisely, we pointed up the non-stationary behavior of time series data that can be subject to concept drifts, the necessity of online adaptive time series variables selection in the case of MTS forecasting, the consequences of the No Free Lunch theorem, inducing the need for the online management of many forecasting models through either online single model selection or the adaptive aggregation of many models into an ensemble model, and the requisite of explainable ML-based solutions. We also noted the importance of forecasting in the context of Industry 4.0 applications, especially for online quality predictive analytics, and the challenges this task entails. In this context, the main goal of this thesis is *to develop new ML methods for forecasting the future values of a time series in an **online**, **adaptive**, and **informed** manner following changes in the time series and the forecasting models' performance over time.*

These methods aim to improve the accuracy of the prediction process in time-dependent domains, thereby improving the quality of decision-making by professionals in a wide range of organizations and institutions. To do so, the objective of these methods is to minimize the difference between the predicted and observed values of time series within the scope of predicting its future values. Our work includes the development of methods for time-dependent time series variables selection for MTS forecasting and the adaptive online management of many forecasting models. We develop equally explainability tools for forecasting models' selection process. Regarding the prediction of the quality of industrial processes, we aim to monitor these processes in a timely manner while overcoming the lack of labeled data.

Therefore, we decompose these research goals into five research questions:

- **RQ1** Given the time-evolving spatio-temporal dependencies among time series variables in MTS, what is the most appropriate way of selecting the most important variables in a timely manner to enhance the predictive performance of MTS forecasting models?
- **RQ2** How can we dynamically select one or many forecasting models and cope with non-stationary sources of variation in time series and the time-varying models' performance?
- **RQ3** How can we provide suitable timely explanations for the reason behind model selection at a given time instant or interval?
- **RQ4** In selecting many models simultaneously, how can we dynamically and adaptively combine them to cope with the above-mentioned variations?
- **RQ5** How can the developed methods be transferred efficiently to industrial case studies to perform online model-based quality prediction? And how can the online management of many models be exploited to integrate sensor and simulation data for ML applications efficiently?

Before developing novel methods for addressing the online management of forecasting models, we first need an approach for adaptively selecting MTS variables in a timely manner. In this context, to answer the first question, we devised a fully-automated framework for both adaptive input MTS variables selection. The adaptation is performed in an informed manner following concept-drift detection in the spatio-temporal dependencies among these variables and the MTS forecasting model's error over time. In addition, a well-designed meta-learning scheme is used to automate the selection of appropriate dependencies measures and the MTS forecasting model (Chapter 3). The second and third research questions were addressed by developing four methods for dynamically selecting either one single or a set of forecasting models. Two of them are specific to Deep Neural Networks (DNNs). These methods exploit gradient-based techniques for generating saliency maps with a coherent design to make them able to specialize the DNNs across different regions in the input time series using a performance-based ranking. The two remaining methods are model-agnostic and can be applied to any family of forecasting models. We assume that different forecasting models have different Regions of expertise or Competence (RoCs) and varying relative performance. Therefore, performance drift detection mechanisms and adaptive model clustering approaches are proposed. To answer the third research question, we exploit the methods that use the concept of RoCs for model selection to provide appropriate visual explanations for the reason for selecting a particular model(s) in a specific time interval or instant (Chapter 4-5). While several approaches have been proposed in the literature for the fourth question, there is no consensus regarding the most appropriate one. In this context, we offer a novel meta-learning approach for aggregating linearly weighted ensembles for the task of time-series forecasting using the Deep Reinforcement Learning (DRL) paradigm. We outline a DRL framework with a coherent design of the components of the environment and the objective function as an aggregation method in our task. In this framework, the combination policy in an ensemble is modeled as a sequential decision-making process that captures the temporal behavior in time series. Furthermore, an actor-critic model aims to learn the optimal weights in continuous action space (Chapter 6).

The proposed methods are adapted to a particular domain of application: real-time quality prediction for Industry 4.0 use cases. One of them consists of online monitoring of an NC-milling process. Furthermore, process simulations are used as synthetic data sources due to the shortage of labeled data. An automated fusion framework is developed to optimize the integration of sensor and simulation data (Chapter 8). Finally, we apply our developed methods for the online forecasting of the NC-milling cutting forces. Two additional case studies for model-based quality prediction using DNNs are presented, and saliency maps are exploited to produce visual explanations for the behavior of the models (Chapter 9).

The research questions presented above are addressed empirically. According to an experimental design developed to this effect, the proposed methods were tested from different perspectives. The significance of the main results was assessed according to the Wilcoxon signed-rank test analysis [48]. In support of reproducible research, the code developed for each set of experiments is available online.

1.3 Thesis Contributions

Learning from time series data is at the core of this thesis. By proposing novel methods, this thesis contributes to the field of time series forecasting. Our contributions reach several fields, including dynamic time series variables selection, online model selection, online ensemble learning, meta-learning, deep learning, deep reinforcement learning, explainability, forecasting, and model-based quality prediction for Industry 4.0 applications. The contributions of this thesis can be then summarized as follows:

- A novel method for online drift-aware input time series variables selection for MTS forecasting using relevance and redundancy analysis. The drift detection mechanism is devised to operate on spatial and temporal dimensions, i.e., the two dimensions of MTS data.
- A meta-learning approach for fully automating the choice of relevance and redundancy measures for MTS variables selection, as well as the selection of the forecasting model.
- A novel model-agnostic method for online ensemble pruning using a combination of a drift-aware performance ranking and models' clustering.
- A novel model-agnostic method for online single model and ensemble members selection for time series forecasting by computing the RoCs of a set of candidate forecasting models using an adaptive clustering procedure.
- Generation of appropriate explanations for the reason for outputting a particular forecast value at a particular time instant and selecting particular models at a particular time instant or interval using clusters' visualization.

- A novel method for online DNNs selection for time series forecasting by computing the RoCs for a set of candidate DNNs using an adaptation of saliency maps to produce Performance Gradient-based Saliency Maps (PGSMs). The PGSMs are updated in an informed manner following concept drift detection in the time series data.
- A novel theoretically-based method for the online adaptive ensemble of DNNs pruning for time series forecasting using pre-computed PGSMs, i.e., the pruning is updated in an informed manner following concept drift detection in both candidate DNNs' performance and the time series data.
- Generation of suitable explanations for the reason behind selecting a specific DNN at a specific time instant or interval using the developed PGSMs.
- An adaptive meta-learning technique for dynamic ensemble aggregation for the task of time-series forecasting using Deep Reinforcement Learning.
- An extensive set of experiments comparing the proposed approaches to State-of-the-Art methods for each task. These experiments also include an analysis of the impact of introducing variations to the proposed methods, run-time comparisons, and a statistical significance analysis of the results.
- Application and adaptation of some of the developed methods to the real-time quality prediction of real-world industrial processes.
- A novel framework for the automated fusion of time series sensor and simulation data for the online monitoring of an NC-milling process.

1.4 Thesis Outline

This thesis is organized into four parts. The first part presents the introduction and includes the current chapter. In this chapter (Chapter 1), we presented and spurred the topic of this thesis. We moreover expressed the main goal and decomposed it into five research questions. At last, we summarised the contributions of the thesis. To include the fundamental concepts used in this thesis and the related works on the evoked topics, Chapter 2 is introduced. In this Chapter, further details on the problems addressed in this thesis, including time series analysis and forecasting, are provided. We also present formal definitions for the main ML tasks performed in this thesis within the scope of time series forecasting.

The main parts of this thesis are part II and part III. Part II includes three Chapters (Chapter 3, 4, 5, and 6), and handles the problem of forecasting, in which the objective is to predict the subsequent value of a time series concurring with its past values. Chapter 3 introduces a method for online adaptive input time series variable selection for MTS forecasting. Chapter 4 is dedicated to presenting the methods we devised for online adaptive model selection and ensemble pruning. In Chapter 5, we present a novel method for online DNNs selection for time series forecasting by computing their RoCs using an adaptation of saliency maps to the time series domain, followed by Chapter 6 in which we propose a novel method for an ensemble of forecasting models aggregation.

Part III is composed of Chapter 7 and 8, and is devoted to the application and adaptation of some of our methods to real-world industrial case studies. Chapter 7 includes learning from heterogeneous data sources, precisely, sensors and simulation to forecast different quality-related variables of an NC-milling process in real-time. Model selection methods are adapted to integrate these data sources efficiently. In Chapter 8, we propose an adaptation of the proposed DNN-based models to provide an explainable quality prediction of two industrial use cases.

The final part of the thesis (Part IV) includes Chapter 9, which concludes the thesis. In this chapter, we summarize our answers to the research questions raised in this thesis and highlight some open issues as well as future work. The thesis also contains an Appendix, where the time series data sets and the learning algorithms are described.

2 Background

In this Chapter, we present the fundamental background related to the topics handled by our thesis. Section 2.1 describes time series data and its main characteristics. We define the possible learning tasks on this data type. Then, we put light on the main topics that will be addressed in the next Chapters. In Section 2.4, we formally define the task of time series forecasting, evoke the related challenges and describe the existing ML approaches to solve this task. Section 2.5 details the task of model selection in the context of time series forecasting. A specific focus is given in Section 2.6 to ensemble learning, which defines an important establishment of the work in this thesis. In Section 2.7, we describe the development of ML explainability topic in general and in the context of time series analysis in particular. Finally, in Section 2.8, we illustrate the current trend in applied ML for model-based quality prediction of industrial processes. In addition, we highlight the importance of learning from heterogeneous data sources, more precisely process simulation and sensors, under some particular circumstances.

To recap, this chapter aims to equip the reader with the fundamentals of the main topics studied in this thesis that are assumed to be necessary for the understanding of the remainder of the thesis.

2.1 Time Series

When formally defining a time series, we need to go beyond the foremost straightforward definition and include the general context as well. Therefore, characterizing the time series as a sequence of observations does not reflect much of the context and does not help to define target tasks on this special data type clearly. A more formal and general definition is given by Mierswa, and Morik [49], who first introduced the notion of value series that also covers time series. In a value series, each element x_n of the value series consists of an index and a value vector. The value vector is an m-dimensional vector and corresponds to a point in the value space.

Definition 1 ***Value Series*** *A value series is a mapping $X : \mathcal{N} \to \mathcal{R} \times \mathcal{C}^m$. The point of the series with index n is noted as x_n. $X_{1:k} = \{x_1, \cdots, x_k\}$ denotes the sequence of points starting at index 1 and ending at index k with length k.*

In the following, we focus on time series. Therefore, the index component corresponds to a measurement of time.

Definition 2 ***Time Series*** *A time series variable X, is a temporal sequence of values, where $X_{1:T} = \{x_1, x_2, \cdots, x_T\}$ denotes the sequence of X recorded until time instant T and x_t is the value of X at a time instant t.*

In this thesis, we focus exclusively on numerical time series, i.e., $x_t \in \mathcal{R}, \forall x_t \in X$, and assume that the observations are captured over regular time intervals, e.g., every minute.

Time series data and its analysis are increasingly substantial due to its massive generation through different technologies in a wide range of fields, including, for example, the Internet-of-Things (IoT) devices [50], the digitalization of industrial and healthcare systems [51], [52], and the evolution of the concept of smart cities [12]. This explains the rapid growth in quantity and quality, and the importance of time series data [9], [53]. In most cases, time series data is continuously acquired. In addition, it reveals a dynamic behavior and is often subject to uncertainty, complicating the exact understanding of its behavior. Therefore, the need for vigorous time series analysis using statistical and ML tools becomes more and more crucial. The aim of time series analysis is to extract meaningful knowledge and statistical information about data points recorded in chronological order. It is carried out to analyze the past behavior of the series as well as to forecast its future behavior.

2.1.1 Time Series Components

Time series can be decomposed into four main components: trend, seasonal, cyclic, and irregular. The time-evolving nature of this data can then be expressed and understood using these components.

The trend component corresponds to a long-term change in the mean level of the time series. When the long-term variations are of no interest to the application, the trend component can be removed. We describe this transformation in detail and its effect on the time series in Section 2.1.3. However, when the time series contains periodical and predictable changes over fixed periods, it is said to include a seasonal component. The seasonal change can also be removed if it is judged of no interest.

In addition to seasonal changes, sometimes time series dispose of some other predictable variations, which occur at no fixed periods. This type of change is called, most often, a cyclic component. The standard illustration of a cyclic pattern can be found in economic cycles, where periods of growth and recession are encountered. After removing the three above components from a time series, the remaining part is referred to as the residual or noise component. This component does not have any explainable behavior and can not be approximated by any trend, seasonal or cyclic pattern, but can have a huge impact on the dynamics of the time series [54].

At a given time instant t, a time series can be decomposed in an additive manner into the above four components. The additive decomposition can be expressed as follows [54]:

$$x_t = Trend_t + Seasonality_t + Cyclic_t + Residuals_t \tag{2.1}$$

where $Trend$, $Seasonality$, $Cyclic$, and $Residuals$ represent the trend, seasonal, cyclic, and residual components of the time series at that point, respectively. Other forms of decomposition can be applied to time series, including multiplicative decomposition or a mixture of both multiplicative and additive. The multiplicative decomposition is given by:

$$x_t = Trend_t \times Seasonality_t \times Cyclic_t \times Residuals_t \tag{2.2}$$

The choice of adequate decomposition depends on the time series data and the problem at hand. For instance, if the seasonal component increases as the trend increase, a multiplicative decomposition can be more appropriate in this case. In addition, if a multiplicative factor models the residual component and the component is positive, an additive decomposition model for $log(x_t)$ can be used in this case:

$$log(x_t) = Trend_t + Seasonality_t + Cyclic_t + Residuals_t \tag{2.3}$$

Figure 2.1 shows an example of a multiplicative decomposition of the time series of Electricity production data [55]. In this example, the multiplicative decomposition seems more appropriate than the additive since the original time series variance and its trend are increasing over time. However, it can also be seen that the residual/random component also has an increasing variance, which implies that a log-transformation (Equation 2.3) can be, in this case, more appropriate for this data. It should be noted that the random series obtained from the decomposition is not precisely a realization of the random process that generated the series but rather an estimate of that realization.

2.1.2 Stationarity

A time series is said to be stationary if there are no significant changes in its mean or variance and if periodic changes have also been removed [53]. In other words, time series properties are preserved over time, and they are independent of the time when the data is observed. However, it should be noted that different levels of stationarity have been defined in the literature [53]. For instance, we can distinguish between strict stationarity and weak or second-order stationarity.

Definition 3 ***Strictly Stationary Time Series*** *A time series is said to be strictly stationary if the joint distribution of* $\{x_1, x_2, \cdots, x_t\}$ *is identical to the joint distribution* $\{x_{1+j}, x_{2+j}, \cdots, x_{t+j}\}, \forall t, j \in \mathcal{N}$.

This definition means that if we shift the time window in which we observe the time series, this does not affect the joint distributions. This condition of strict stationarity is often relaxed in practice and replaced by second-order stationarity or weak stationarity.

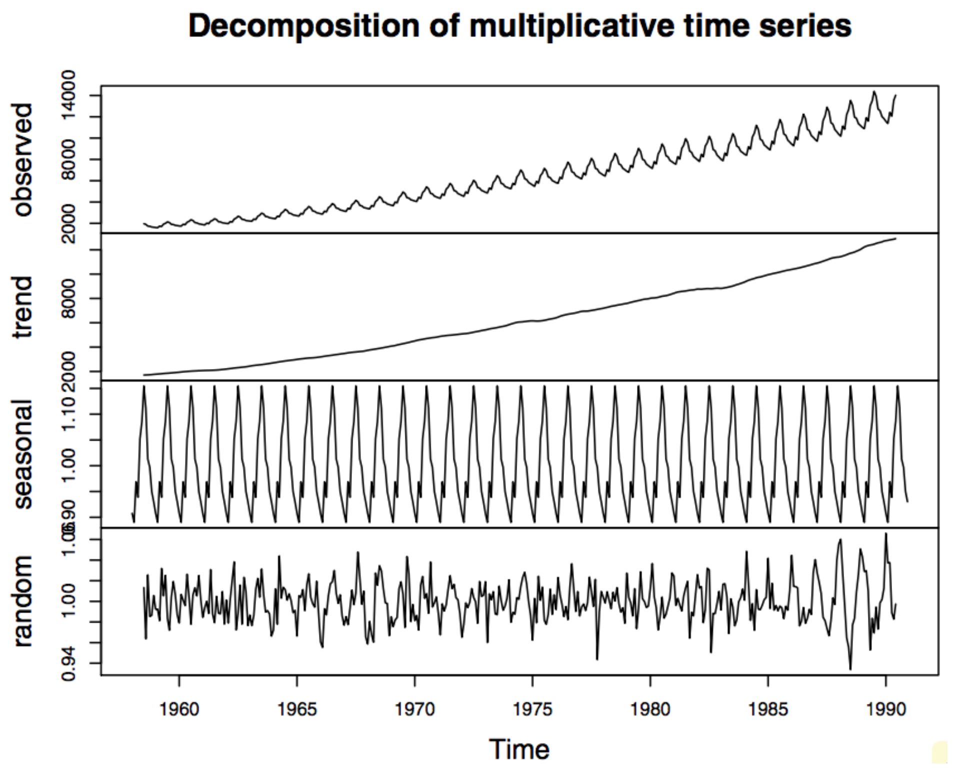


Figure 2.1: Decomposition of Electricity time series production data [55].

Definition 4 ***Weak Stationary Time Series*** *A time series is said to be a second-order or weak stationary series if it has a constant mean and a constant variance and its auto-covariance does not depend on time.*

The definition of second-order stationarity can be formally expressed using the following conditions:

Condition 1 *Constant Mean and Variance.*

$$\mu_j = \mu \wedge \sigma_j = \sigma, \forall j \in \{1..t\} \tag{2.4}$$

Condition 2 *Time-invariant Covariance.*

$$\gamma(t, t-j) = E[(x_t - \mu) - (x_{t-j} - \mu)] = \gamma_j, \forall j \in \mathbb{Z} \tag{2.5}$$

where μ is the mean of the time series X and μ_j is the mean of the sequence $X_{1:j}$, i.e. X recorded till time instant j. Similarly, σ is the variance of time series X and σ_j is the variance of the sequence $X_{1:j}$ In the remainder of this thesis, when we refer to a time series as stationary, we use the definition of second-order stationarity.

A time series can also be considered as trend-stationary when the mean of its trend is deterministic. In this case, the trend can be removed to obtain a stationary residual time series.

2.1.2.1 Stationarity Tests

Stationarity is an influential property in time series modeling, and several tests were devised to verify its presence. In this section, we recapitulate three widely used parametric tests and point the reader toward non-parametric ones.

The Dickey-Fuller test [56] was the first statistical test developed to test the null hypothesis that a unit root is present in an Auto-Regressive (AR) model (See Section 2.4) of a given time series and that the generating process is thus non-stationary. The original test treats the case of a simple lag-1 AR model. The test has three versions that differ in the model of the unit root process they test for. The choice of the version can significantly affect the size and power of the test. Using prior knowledge or structured strategies for a series of ordered tests allows the discovery of the most appropriate version. Extensions of this test were developed to adapt more complex models and data. These extensions include the Augmented Dickey-Fuller (ADF) [57](using AR model of any order p and supporting the modeling of time trends), the Phillips-Perron test (PP) [58] (adding robustness to unspecified autocorrelation and heteroscedasticity) and the ADF-GLS test [59] (locally de-trending data to deal with constant and linear trends).

The second method is presented by Kwiatkowski et al. and is called the Kwiatkowski-Phillips-SchmidtShin (KPSS) test [60]. This method tests the null hypothesis that a given time series is stationary in trend, whereas the alternative hypothesis is the presence of a unit root. Unit roots are typical origins of non-stationarities in trend. A time series containing a unit root is assumed to be non-stationary in trend. Nevertheless, after removing the trend component, the remaining residuals are stationary. The KPSS test is typically used for trend inclusion in forecasting models, for example, ARIMA (Auto-Regressive Integrated Moving Average) [61].

The aforementioned tests do not allow for the possibility of a structural break which means an abrupt change in the time series inducing a change in the mean or other parameters of the time series generating process. Considering the time of the break as an exogenous phenomenon, it has been shown that the power to reject a unit root decreases when the stationary alternative is true and a structural break is ignored [58]. Therefore, The Zivot and Andrews Test [62] is proposed as a unit root test in which the exact time of the break-point is assumed to be unknown. To do so, Zivot and Andrews [62] proceed with three models to test for a unit root. The first model allows for a one-time change in the level of the series. The second model permits a one-time change in the slope of the trend function. The third model combines one-time changes in the level and the slope of the trend function of the series.

To face the limitations of the above-mentioned parametric tests that have been proved to cover only a narrow sub-class of possible cases encountered in real-world time series data, non-parametric tests for stationarity have appeared in the time series analysis literature. These tests offer a promising avenue for examining time series data. There is no longer the need to assume very simple parametric models to apply to the data in order to capture non-stationarities. They also minimize the risk of not discovering a complex form of non-stationarity. The reality of it, however, is more complex. Therefore, at the moment, there are no widely-applicable non-parametric tests that enclose all real-world scenarios generating time series data. Rather, these tests restrict themselves to specific types of data or processes. For example, the non-parametric test in continuous-time Markov processes [63] was suggested for univariate time-homogeneous Markov processes. The idea is to construct a kernel-based statistical test and conduct Monte-Carlo simulations to study the test's finite-sample size and power properties. Delft et al. [64] presented a non-parametric stationarity test limited to functional time series, meaning that data is obtained by separating continuous time records into consecutive intervals. Due to the limitations and lack of implementations of a handful of non-parametric tests, the time series analysis process is forced toward making strong assumptions about the data to use parametric tests.

2.1.3 Time Series Transformations

In this section, we present transformations that map the time series data from the given vector space into the same or another space, i.e., time series are original elements of the vector space $\mathbb{R}^2$ (See Definition 1). The basis $\mathcal{B}$ of a vector space V is a set of vectors that can represent all vectors in V by their linear combination. The scalar product is the only required operation on vector spaces for the transformations. Since the most common transformations performed on time series data are based on the transformation into the infinite space of harmonic oscillations, we suppose *Hilbert* spaces.

Definition 5 ***Hilbert Space*** *Let H be a vector space with an inner product $< f, g >$. H is called a Hilbert space if the norm defined by $|f| = \sqrt{< f, f >}$ turns H into a complete metric space, i.e., any Cauchy sequence of elements of the space converges to an element in that space.*

The assumption of *Hilbert* spaces generalizes all finite-dimensional spaces with a scalar product, such as Euclidean space with an ordinary scalar product. additionally, we use *Hilbert* spaces with an infinite number of dimensions to present the concept of Fourier and Wavelet transformations. Therefore, we need an infinite-dimensional *Hilbert* space of functions.

Definition 6 ***Function Space*** *Let P be a Hilbert space. If the elements $f \in P$ are functions, P is called a Function Space.*

Following Definition 6, the set of all functions $f : \mathbb{R} \rightarrow \mathbb{R}$ with a finite integral $\int_{-\infty}^{\infty} f(x)^2 dx$ together with the inner product $< f, g >= \int_{-\infty}^{\infty} f(x)g(x)dx$ form a well-known function space $\mathcal{L}^2$.

2.1.3.1 Temporal Space

Time series typical components, such as trend or seasonality, may violate stationarity conditions. Therefore, some transformations have been suggested to make the time series in question stationary [54]. This section presents some of the commonly used transformations, namely differencing, model fitting, logarithms, and root squares. Differencing consists of subtracting the previous value of the time series from its current value:

$$x'_t = x_t - x_{t-1} \tag{2.6}$$

where x'_t is the transformed t-th value of the time series. This operation is generally used to account for a trend component. For instance, we differentiate the original time series data to remove the trend component. Differencing can be applied multiple times to a time series. For example, the well-known automatic forecasting procedure *auto.arima* [65] uses the KPSS test described above to determine the number of times the series needs to be differentiated to reach trend-stationarity. Differencing can also be used to account for seasonality by computing the difference between the current time series value and the previous value from the same season:

$$x'_t = x_t - x_{t-n_{sea}} \tag{2.7}$$

where n_{sea} corresponds to the seasonality period.

Fitting an appropriate model to the time series data is another manner to account for the trend. In this case, the trend component is removed by modeling the residuals resulting from that fit [54].

It may also happen that the time series has a time-varying variance, which also breaks stationarity. In this case, applying logarithm or root square operations most often helps stabilize the variance [66]. However, resulting negative data values thwart the use of these approaches. One solution to mitigate this issue is to add a suitable constant to make the data positive before applying these transformations. These approaches are particular instances of the Box-Cox transformation [67].

2.1.3.2 Frequency Space

Fourier Transform was developed to inspect periodicity in time series, i.e., whether a time series contains a pattern, which repeats itself after a specific amount of time, called a period. The inverse of the period is named frequency. For instance, a time series with a period of 1 *second*, its frequency is 1 *Hertz (Hz)*. The main idea behind the transformation is to decompose the time series into a series of its periodic components. Therefore, for a periodic time series, the definition for the continuous Fourier transform F of the time-domain description of a time series X is given by:

$$F(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} X(t)dt \tag{2.8}$$

where ω is the frequency and an evaluation of X at time t, denoted by $X(t)$, is given by x_t. After applying the Fourier transform on a time series, information about its composing frequencies is obtained. At this step, we usually speak about the space of frequency or frequency-domain description. The aim of applying the Fourier transform is to decompose the original time series into a sum of its simpler signals, no matter how complex the original one looks. An example of Fourier transform on synthetic data set with a number of individual frequencies $[2, 5, 3]$ is shown in Figure 2.2. In Figure 2.2, the frequency spectrum that composes the time series is obtained after applying Fourier transformation.

To change from frequency back to the time domain, we apply the inverse transform given by:

$$X(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega t} F(\omega) d\omega \tag{2.9}$$

If the time series represents a non-stationary or transient process, no information related to changes in the frequency is available.

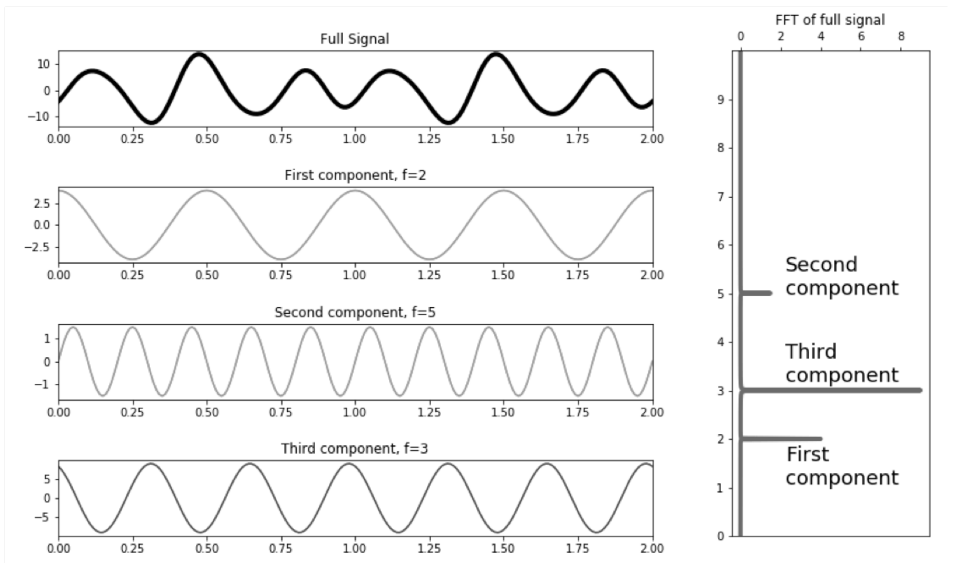


Figure 2.2: Synthetic data, in the first horizontal box the original time series, next boxes the individual components, corresponding to frequencies of 2, 5 and 3 respectively. In the vertical box, the result of the Fourier transform.

Windowed Fourier Transform If the time series X changes its characteristics over time, a frequency spectrum at time $t = \tau$ can be obtained by multiplying the time series by a window-function $g(t - \tau)$ which is non-vanishing in the region $t = \tau$ only. The relevant part of the time series can then be retained and analyzed. The WTF can be written as:

$$F(\omega, \tau) = \int_{-\infty}^{\infty} X(t) g^*(t - \tau) e^{-i\omega t} dt \tag{2.10}$$

where $*$ denotes the complex conjugate. The inverse transform is given by:

$$X(t) = \frac{1}{C} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\omega, \tau) g(t - \tau) e^{i\omega t} d\omega d\tau \tag{2.11}$$

where C is a context is given by:

$$C = \int_{-\infty}^{\infty} |g(t)|^2 dt \tag{2.12}$$

The window used in WFT analysis may be rectangular, Cosine (Hanning), or Gaussian (Gabor) shaped functions. The transform function $F(\omega, \tau)$ yields information related to the frequency spectrum at time τ. The frequency spectra obtained may, however, be strongly dependent on the shape and, in particular, the width of the function. The window functions used in the WFT analysis are independent of the frequency, and the choice of a particular window reflects the trade-off between resolution in time and frequency. Hence, a wide (narrow) window gives good (poor) frequency resolution and poor (good) time resolution.

The precision with which a signal could be simultaneously localized in time Δt and frequency $\Delta\omega$ is given by:

Condition 3

$$\Delta t \Delta\omega = const \tag{2.13}$$

This uncertainty relation is a consequence of the fact that in order to identify a certain frequency, the duration of the time series must at least be of the same order as the period corresponding to that frequency.

2.1.3.3 Wavelet Space

Subject to the uncertainty relation presented by Condition 3, the use of wavelet transforms yields optimum time and frequency resolutions for the time series. The basis of wavelet analysis is the (daughter) wavelets which are constructed from a basic mother wavelet Ψ by dilation and translation, i.e., $t \rightarrow \frac{t-\tau}{a}$, where a is a scaling parameter. The daughter wavelet ψ can then be written as:

$$\psi(t) = \frac{1}{\sqrt{a}}\Psi(\frac{t-\tau}{a}) \tag{2.14}$$

The translation of the analyzing window in time is given by τ, as in the WFT. However, its size can be adjusted to the frequency analyzed by the scaling parameter a. Short windows are used for high frequencies, while long windows are used for low frequencies. The wavelet transform is defined by:

$$W(a,\tau) = \int_{-\infty}^{\infty} x(t)\psi^*(t)dt \tag{2.15}$$

The inverse transform or reconstruction formula is given by:

$$x(t) = \frac{1}{C}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} W(a,\tau)\psi(t)\frac{dad\tau}{a^2} \tag{2.16}$$

where $C = \int_{-\infty}^{\infty} \frac{|\Psi(\omega)|^2}{|\omega|}d\omega$ must be finite (the admissibility condition). Many wavelets types are possible to use, provided they fulfill the following requirements:

Condition 4 *Admissibility Condition (Zero Average)*

$$\int_{-\infty}^{\infty} \Psi(t)dt = 0$$

which implies that the wavelet must be oscillatory.

Condition 5 *Finite Energy*

$$\int_{-\infty}^{\infty} |\Psi(t)|^2 dt < \infty$$

Condition 6 *Existence of the Fourier Transform*

$$\Psi(\omega) = \int_{-\infty}^{\infty} \Psi(t)e^{-i\omega t}dt$$

In terms of the Fourier transform $F(\omega)$ given by Equation 2.8 and $\Psi(\omega)$, the wavelet transform in Equation 2.15 can be written as:

$$W(a,\tau) = \frac{\sqrt{a}}{2\pi}\int_{-\infty}^{\infty} \Psi^*(a\omega)F(\omega)e^{i\omega\tau}d\omega \tag{2.17}$$

Next, we give an example of the Fourier transform of a time series extracted from the ElectroCardioGram (ECG) dataset [1]. An ECG is a graphical representation of the electrical

[1] This data is sourced from https://www.physionet.org/.

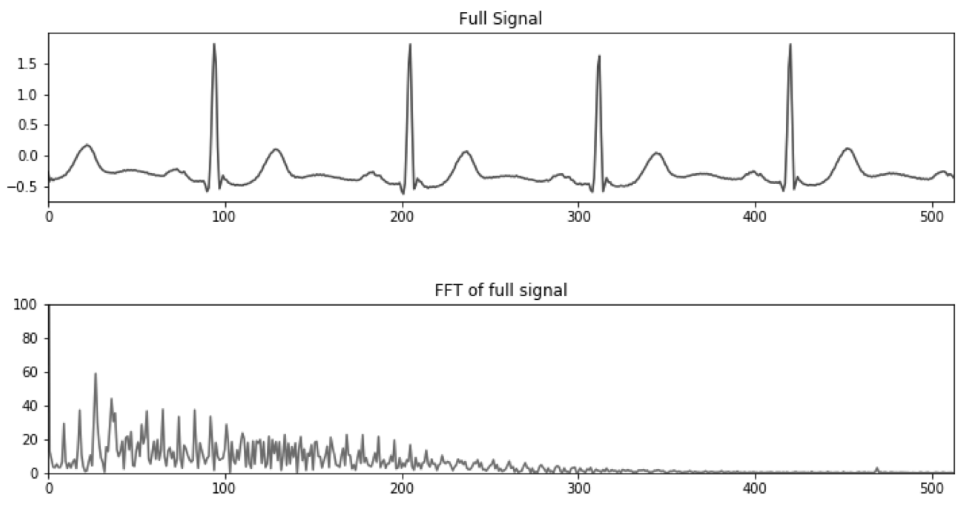


Figure 2.3: ARR ECG time series (above), FT transform (bellow).

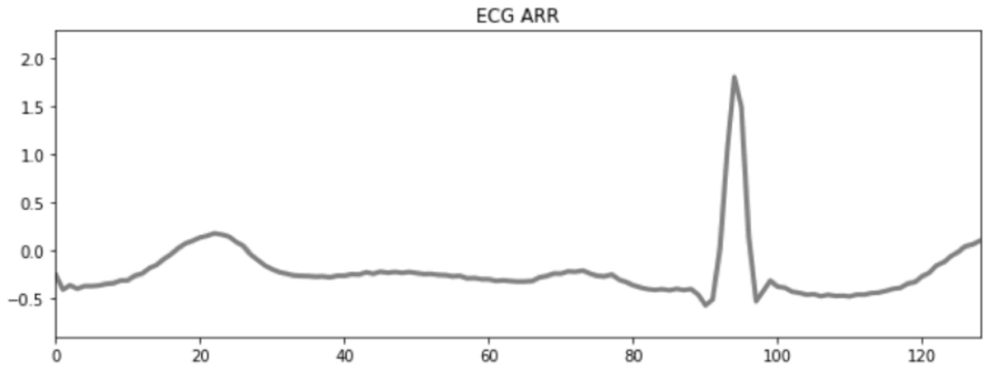


Figure 2.4: Wavelet transform of the ARR ECG sequence.

activity of the heart. It consists of an electric signal that varies over time. This signal is used for analyzing heart activity and predicting the presence of some diseases. Earlier, ECG data was restricted to healthcare environments, but nowadays, smart devices like smartphones and smartwatches can produce an electrocardiogram. Therefore, ECG data analysis is becoming more and more demanded. One ECG signal can belong to one of three different classes: cardiac ARRhythmia (ARR), Congestive Heart Failure (CHF), and Normal Sinus Rhythm (NSR). In Figure 2.3, we plot as a time series sequence of an ARR example followed by its Fourier transform.

In contrast to the very well-defined set of frequencies obtained with synthetic data, in Figure 2.2, the Fourier transform in Figure 2.3 is less clear. Hence, some peaks can be identified, but they are not sharp in some intervals. This is explained by the fact that the synthetic time series is composed of three frequencies fixed in time opposingly to the time-varying frequencies in the ECG time series, which violates Condition 3. A wavelet transform applied to the same time series sequence from the ECG dataset is shown in Figure 2.4.

For better visualizing the transformation, we can use a scaleogram, a tool that builds and displays the $2D$ spectrum for the continuous wavelet transform. A scaleogram takes the

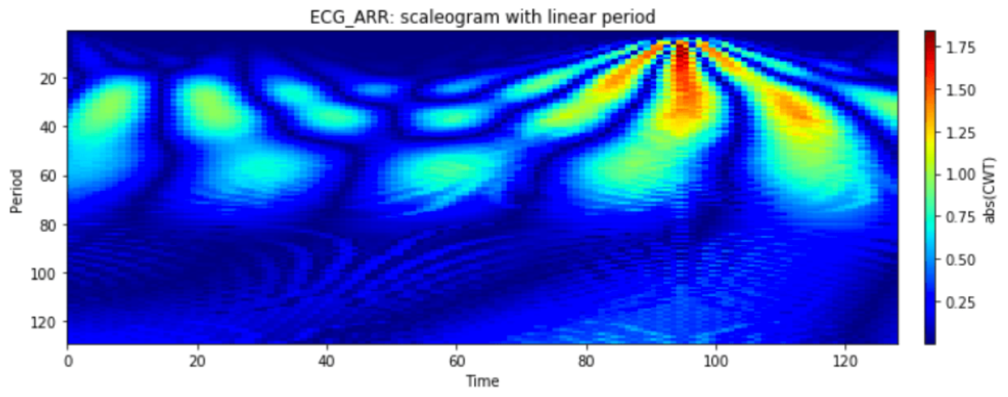


Figure 2.5: A scaleogram of the ARR ECG sequence wavelet transform.

absolute value of the wavelet transform coefficients of a time series and plots it. The scaleogram is shown in Figure 2.5.

Each horizontal characteristic in the scaleogram can be interpreted as a frequency of the total signal. The fact of not seeing a continuous line in Figure 2.5 corresponds to non-continuous frequencies in time.

2.1.3.4 Correlation Space

A correlation measures the extent of a linear relationship between two variables; autocorrelation measures the linear relationship between lagged values of a time series. Its value r_p between x_t and x_{t-p} can be written as:

$$r_p = \frac{\sum_{t=p+1}^{T}(x_t - \bar{x})(x_{t-p} - \bar{x})}{\sum_{t=p+1}^{T}(x_t - \bar{x})^2} \tag{2.18}$$

where T is the length of the time series and $\bar{x}$ its mean value. The calculation of autocorrelation values between two points in time, t and $t-p$, produce the correlation space, where for each lag p, the correlation coefficient r_p in $[-1, +1]$ is indicated. This transformation is commonly used for audio data since it eases the recognition of the speed of the music, measured in beats per minute [49].

In the case of the p-th order, the correlation between x_t and x_{t-p} can in part be due to the correlation these observations have with the intervening lags x_{t-1}, x_{t-2},$\cdots$, x_{t-p+1}. To adjust for this correlation, the Partial Auto-Correlation Function (PACF) is calculated by:

$$PACF_p = \frac{\gamma(t, t-p)}{\sigma_t \sigma_{t-p}} \tag{2.19}$$

where γ and σ are the covariance and the variance, respectively. For details, see Equations 2.5 and 2.4.

2.1.3.5 Symbolic Representation

Non-numerical representations of time series, such as Symbolic Aggregate approXimation (SAX) which bins continuous time series into intervals, transforming each time series independently into a sequence of symbols, usually letters [68], have recently been used in the task of time series forecasting [69], [70]. Note that such representations would potentially allow availing of the wealth of data structures and algorithms from the text processing and bio-informatics communities [68], [71], and possibly convert the real-valued data in a streaming fashion for online tasks of time series [72].

2.1.3.6 Filters

A filter is an operator that maps the time series X into another series X' within the same space. A linear filter satisfies the following:

$$x'_t = \sum_{-\infty}^{\infty} \beta_j x_{t-j} \tag{2.20}$$

The collection $\boldsymbol{\beta} = \{\beta_j\}$ is called a linear filter. X' is a linear function of X and it is a filtered version of X. Linear filtering, where $\boldsymbol{\beta}$ is assumed to be a known collection of numbers, is often used to identify patterns in a noisy time series.

Filtering involves a convolution between the two series X and $\boldsymbol{\beta}$. The convolved series is then X'. At time instant t, the convolution is the sum of the product between the $\boldsymbol{\beta}$ series going forward and the series X centered at time t going backward.

$$\begin{array}{ccccccccc} \cdots & x_{t+3}, & x_{t+2}, & x_{t+1}, & x_t, & x_{t-1}, & x_{t-2}, & x_{t-3} & \cdots \\ & \times & \times & \times & \times & \times & \times & \times & \\ \cdots & \beta_{-3}, & \beta_{-2}, & \beta_{-1}, & \beta_0, & \beta_1, & \beta_2, & \beta_3 & \cdots \end{array}$$

with $\sum_{-\infty}^{\infty} |\beta_j| < \infty$.

2.1.4 Multivariate Time Series

Looking at the time series from a univariate perspective means that only the present and past values of the time series in question are available for fitting the forecasting model. However, in nowadays rapidly growing digital environments and Internet-of-Things systems, the representation of time series data involves most often multiple interdependent variables, creating thus a Multivariate Time Series (MTS) data [17]. This data represents an enriched form of information about the application.

Definition 7 ***Multivariate Time Series*** *An MTS* $\mathbf{X}$ *is composed of multiple time series variables, i.e.* $\mathbf{X} = \{X^1, X^2, \cdots, X^N\}$*, that are interdependent. The MTS* $\mathbf{X}_{1:T}$ *recorded until a time instant* T *can be formally described as* $N \times T$*-dimensional matrix, with* $\mathbf{X}_t = \{x_t^1, x_t^2, \cdots, x_t^N\}, \forall t \in [1, T]$ *which represent the spatial dimension of* $\mathbf{X}$ *at a fixed time instant* t *and* $X_{1:T}^i$ *represents the evolution of* $\mathbf{X}$ *over the temporal dimension on the time series variable* X^i*.*

$$\mathbf{X}_{1:T} = \begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_{T-1}^1 & x_T^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^i & \cdots & \cdots & x_{T-1}^i & x_T^i \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^N & \cdots & \cdots & x_{T-1}^N & x_T^N \end{pmatrix} = \begin{pmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_t \\ \vdots \\ \mathbf{X}_T \end{pmatrix}^{\top}$$

In this work, the time series variables of an MTS are assumed to be recorded simultaneously with the same frequency.

2.1.5 Time Series Distance Measures

In the context of MTS, it is most often required to measure the similarity between its composing time series [16]. It is also possible to demand a computation of similarity between a set of candidate univariate time series, i.e., individual time series for classification or clustering tasks. Several similarity/distance measures have been proposed in the ML literature [73]. In this section, we present some of the commonly used measures. Denote by X^i and X^r two time series recorded until time T.

Minkowski Distance D_{Mink} [74] is a generalization of both Euclidean D_E and Manhattan D_{Man} distances. The Minkowski distance between X^i and X^r, is given by:

$$D_{Mink} = (\sum_{t=1}^{T} |x_t^i - x_t^r|^q)\frac{1}{q} \tag{2.21}$$

If $q = 1$, Equation 2.21 defines the Manhattan distance D_{Man}. If $q = 2$, D_{Mink} gives then the Euclidean distance D_E, which is known to be more sensitive to outlier values in the time series sequence due to its non-linear character. The main advantage of Minkowski distances is that they are straightforward to compute and interpret. Minkowski distances do not take into consideration the time interval between measurements, allowing them to be applied generally to unequally spaced observations. Even though Minkowski distances, including Euclidean distance D_E and the Manhattan distance D_{Man}, are mathematically simple, they have some limitations. Most notably, they do not take into account the non-stationarity of variance or temporal cross-correlations in the time series [75]. Consequently, the time series observations with the largest variance will be overlooked by Equation 2.21 as they contribute more strongly to the similarity [74]. To mitigate this issue, the Mahalanobis distance D_{Mah} is proposed [76]:

$$D_{Mah} = \sqrt{(x_t^i - x_t^r)^\top \Sigma^{-1} (x_t^i - x_t^r)} \tag{2.22}$$

where Σ is the covariance matrix of the time series, which needs to be estimated beforehand. The assumption behind the D_{Mah} is that all the data points in a time series have temporal distributions that can be represented by Σ. The major drawback of D_{Mah} is the need for prior information on the covariance matrix.

Correlation measures The Pearson correlation coefficient D_{PC}, also referred to as Pearson's r or the bivariate correlation, is a statistic that measures the linear correlation between X^i and X^r and is given by:

$$D_{PC} = \frac{\mathbb{E}[(X^i - \mu_{X^i})(X^r - \mu_{X^r})]}{s_{X^i} s_{X^r}} \tag{2.23}$$

where μ_{X^i} and μ_{X^r} are the means of X^i and X^r, respectively, and s_{X^i} and s_{X^r} are their corresponding standard deviations.

The Spearman's rank correlation coefficient D_{SC}, named after Charles Spearman, is a non-parametric measure of rank correlation (statistical dependence between the rankings of two variables). It measures how well the relationship between two X^i and X^r can be described using a monotonic relationship. A monotonic relationship checks whether the value of one variable increases, so does the value of the other variable, or if the value of one variable increases, the other variable value decreases. First, X^i and X^r are converted to to ranks $R(X^i)$ and $R(X^r)$.

$$D_{SC} = \frac{\gamma(\mathrm{R}(X^i), \mathrm{R}(X^r))}{s_{\mathrm{R}(X^i)} s_{\mathrm{R}(X^r)}} \tag{2.24}$$

where $\gamma(\mathrm{R}(X), \mathrm{R}(Y))$ is the covariance of the rank variables, and $s_{\mathrm{R}(X^i)} s_{\mathrm{R}(X^r)}$ are their standard deviations.

The main difference between the two correlation coefficients is that Pearson's correlation is better suited when a linear relationship between the two variables exists. In contrast, Spearman's correlation works for monotonic relationships in general. Another difference is that Pearson's correlation is computed on raw data values of the variables, whereas Spearman's correlation is computed with rank-ordered variables.

Dynamic Time Warping Distance DTW has been introduced in the literature by Vintsyuk, T.K. (1968) for audio time series data [77]. DTW can also be used for time series of different lengths. We consider time series of the same length in our case. DTW searches for the temporal alignment that minimizes Euclidean distance between aligned series. A temporal alignment can be defined as a matching between time indices of two time series. More formally, the optimization problem is given by:

$$DTW_q(X^i, X^r) = \min_{\pi \in \mathcal{A}(X^i, X^r)} \left(\sum_{(t,t') \in \pi} D_E(x^i_t, x^r_{t'})^q \right)^{\frac{1}{q}} \tag{2.25}$$

Here, an alignment path π of length L is a sequence of indices pairs $\Big((t_0, t'_0), \dots, (t_{L-1}, t'_{L-1})\Big)$ and $\mathcal{A}(X^i, X^r)$ is the set of all admissible paths. In order to be considered admissible, a path should satisfy the following conditions:

- The beginning (respectively the end) of the time series are matched together:
  - $\pi_0 = (0, 0)$.
  - $\pi_{L-1} = (T-1, T-1)$
- The sequence is monotonically increasing in both t and t' and all time series indices should appear at least once, which can be written:
  - $t_{j-1} \leq t_j \leq t_{j-1} + 1$
  - $t'_{j-1} \leq t'_j \leq t'_{j-1} + 1$

The optimal alignment path, also called the warping path, can be computed using dynamic programming. The matrix in Figure 2.6 shows the optimal warping path on the distance matrix of the two time series. All warping paths start at the lower left corner and go to the upper right corner. The optimal warping path has the lowest cost in this way. The cost of a matrix cell (n, m) is $(x^r_n - x^i_m)^2$, and the cost of a path takes the sum of all matrix cells it uses.

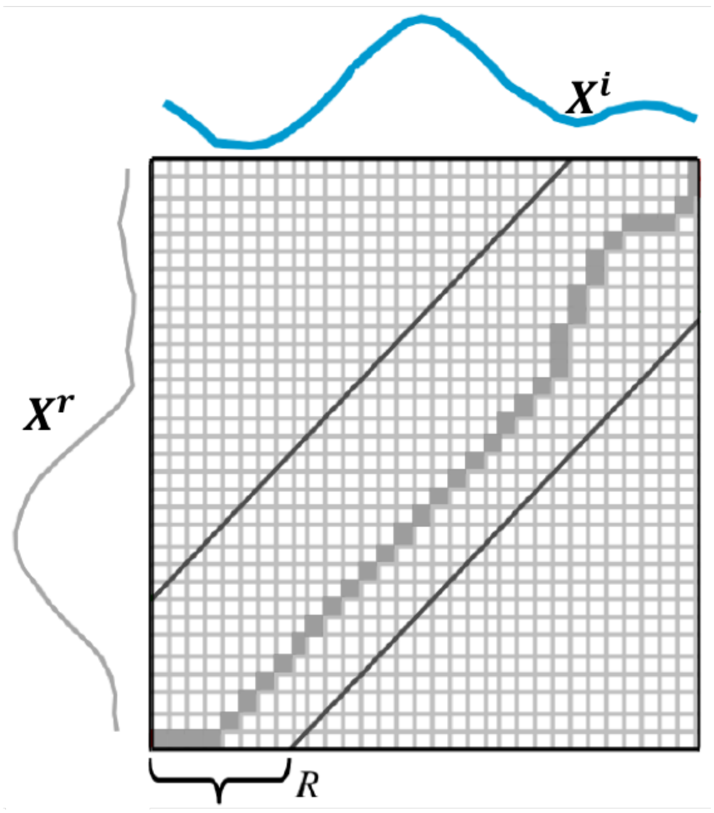


Figure 2.6: The warping path between the two series X^i and X^r on the distance matrix. The bounds with distance R are the Sakoe-Chiba-Band that is used to constrain the warping path.

Solving this dynamic programming problem has a complexity of $O(T^2)$ given T is the length of the time series. This makes DTW expensive to compute compared to Euclidean distance with $O(T)$ complexity. A simple constraint that decreases the computational cost of DTW is the Sakoe-Chiba-Band [78]. A Sakoe-Chiba-Band of size R adds a constraint for the warping paths. The elements of the warping path are not allowed to be further away than R from the diagonal of the matrix. We also see this constraint in the matrix in Figure 2.6. The DTW distance is between two time series using the warping path with minimal cost.

There has been a lot of research works focused on reducing the computational cost of DTW and on its application for time series classification [79]–[81], clustering [82], and indexing [83]. The State-of-the-Art implementation for subsequence search with DTW is called the UCR Suite and is presented in [84]. The implementation utilizes multiple optimizations that compute lower bounds of the DTW distance. If the computed lower bound value is already higher than the current smallest distance, there is no need to calculate the real DTW distance. A set of lower bounds with increasing complexity is computed, and only in the worst case the full DTW computation is required. The computed subsequence match remains identical, while the computational costs are reduced significantly. The introduced optimizations allow the extensive use of similarity search using DTW [84], [85].

Fourier-based Distance The Fourier transform presented in Section 2.1.3 can be used to transform the time series X^i and X^r into a set of scaled cosine waves with unique amplitude

A_j and phase shift ϕ_j:

$$x_t^{i/r} = A_0^{i/r} + \sum_{j=1}^{L-1} A_j^{i/r} cos(2\pi jt + \phi_j^{i/r}) \tag{2.26}$$

where i/r denote the time series with index i or r with:

$$A_j^{i/r} = \sqrt{(F_{j,c}^{i/r})^2 + (F_{j,s}^{i/r})^2} \tag{2.27}$$

and

$$\phi_j^{i/r} = \arctan(\frac{F_{j,c}^{i/r}}{F_{j,s}^{i/r}}) \tag{2.28}$$

j is the frequency of the Fourier transform components (i.e., the number of cosine wave cycles over the time series), F_j^c and F_j^s are the real and imaginary parts in Euler's equation, respectively. Together A_j and ϕ_j describe the j-th frequency FT component as one cosine wave in the frequency domain, whereas the sum of the cosine waves represents the original time series [86]. The Fourier distance measure is given by the Euclidean distance between m selected amplitude and phase components of the discrete Fourier transform:

$$D_{FT} = \sqrt{\sum_{j=0}^{m}(A_j^{i/r} - A_j^{i/r})^2 + \sum_{j=0}^{m}(\phi_j^{i/r} - \phi_j^{i/r})^2} \tag{2.29}$$

Since the Fourier transform contains amplitude and phase, it is sensitive to amplitude scaling, time scaling, and translation.

2.2 Concept Drift

In Section 2.1.2, we presented the notion of stationary time series. Hence, a time series is considered stationary if its mean, variance, and autocorrelation structure are static over time. Time series stationarity is closely linked to the phenomenon of the so-called concept drift [87]. When non-stationary occurs in a time series, it is considered subject to concept drift. In other words, concept drift represents changes in the underlying process generating the time series being followed. This change in the process naturally impacts the data distribution of the time series, yielding it to change.

Even though the number to be predicted is referred to as the concept, it can also refer to phenomena of interest other than the target idea, such as an input. However, it should be noted that in the case of (real) concept drift, we refer to the change in the decision boundary $P(Y = output|X = input)$ [88]. In the case of data drift (or virtual drift), the boundary remains the same even though P(X) has changed [88]. In this thesis, we will tackle data drift for different variables of interest, such as error or correlation time series. Since these variables are of interest for tracking significant changes in the forecasting methods, they are treated as 'concepts', and we use concept drift to refer to drift detection in these variables.

2.2.1 Concept Drift Types

Different types of concept drift can be distinguished [89]. Abrupt concept drift occurs when the process generating the data suddenly changes. This type of drift can occur due to, for example, some accidents. Incremental and gradual concept drifts happen when the data distribution changes smoothly. This occurs due to, for instance, a trend component in the time series. Reoccurring concept drift indicates cyclic changes in the dynamics of the time series. One very common aspect that induces this type of concept drift is seasonality. Concept drift is one of the major challenges when forecasting time series data. It generally happens in dynamically changing environments and complicates the forecasting task [89]. Therefore, forecasting methods should be able to cope with the non-stationary behavior of time series. Particularly, according to Gama et al. (2014) [89], predictive models should be able to detect concept drift as early as possible. It is also important that the models can distinguish between concept drift and noisy time series data points, i.e., points that denote irregular values that fall outside the standard behavior of the data. In other words, predictive models should be adaptive to concept drift but at the same time robust to noise.

2.2.2 Concept Drift Adaptation

A taxonomy of adaptive algorithms designed to cope with concept drift is presented with details in [89]. This taxonomy is organized according to different perspectives: memory, change detection, learning, and loss estimation. In this part, we focus on the learning component since it is per se enclosing the strategies that forecast the future behavior of time series and update the predictive model over time. This is one of the major goals of this thesis. A complete taxonomy overview is presented in [89].

Concept drift adaptation using learning strategies can be grouped into three main families. The first family focuses on the learning mode. Hence, a predictive model may be retrained or incrementally updated. The former regularly, for example, when a new time series observation is available or periodically, e.g., with the collection of a certain number T_p of observations, discards the current model and retrains a new one from scratch. Opposingly, incremental approaches update the current model using incoming observations.

The second family focuses on the model adaptation fashion itself. In this family of methods, we may distinguish two main types of adaptations, blind and informed. The blind adaptation methods update the predictive models without any explicit detection of concept drift. Periodic retaining and incremental algorithms are an example of this approach. Contrariwise, informed adaptation approaches are put into action when some trigger is launched, for example, an alarm that concept drift has occurred. Informed adaptation mechanisms are typically followed by model retraining mode to adapt to the detected drifts. Finally, the third family includes model management approaches that maintain a pool of different predictive models combined to make predictions. This type of approach is commonly known in the ML literature as ensemble learning [90]. The combined prediction is usually a weighted average of the individual predictions. Ensemble models cope with concept drift by changing these weights over time. Ensemble weights can be changed blindly by simply tracking the performance of the predictive models over some validation time windows and setting their values such that the worst-performing models are penalized [12], [16]. These models are known in the literature as dynamic ensembles [91]. In Section 2.6, we dive further into ensemble model learning for time series forecasting.

2.3 Time Series Learning Tasks

Before diving into the details of the forecasting task, we give a brief overview of the learning tasks that occupy the attention of most time series analysis research [72].

- **Indexing:** consists of searching the similar time series from a database to a query time series [92].
- **Clustering:** consists of finding groupings of time series in a database using a similarity measure [93].
- **Classification:** consists of learning the relationship between a time series and a categorical class label [20].
- **Summarization:** consists of determining an approximation of a very long time series, i.e., composed of an extremely large number of data points. This approximation should maintain the essential characteristics of the original time series but fits it to a single executive summary such as a computer screen [94].
- **Anomaly Detection:** consists of detection parts in the time series that contain anomalies, abnormal or novel behavior given some model/ knowledge of “normal” behavior [95], [96].

2.4 Forecasting

One of the major goals of time series analysis is to predict the future behavior of the data. This process is commonly known as forecasting, which consists of predicting the future values of a time series as accurately as possible, given all the information available, including historical data and knowledge of any future events that might impact the forecasts [54]. Forecasting is a typical statistical task for several business structures since it plays a central role in decision-making in many processes, such as scheduling and managing production, transportation, and resources. In addition, forecasting helps to provide a guide for long-term strategic planning, such as decisions on investments.

The forecasting horizon is defined depending on the specific application at hand. We may distinguish between short-term, medium-term, and long-term forecasts. Short-term forecasts are usually needed for scheduling purposes such as the scheduling of transportation, where for instance, forecasts of demand are also most often required. With respect to medium-term forecasts, they are usually related to the planning of future resource requirements, for example, purchasing raw materials, hiring personnel, or buying machines and equipment. Long-term forecasts are required in strategic planning. Such decisions must take into account some other knowledge about the application. Such knowledge can be incorporated into the forecasting model. For example, investment decisions should consider some information about market opportunities, environmental factors, and internal resources.

In practice, it is advised that organizations develop a forecasting system that includes several methods for predicting uncertain events [54], [97]. Such a forecasting system requires the development of expertise in identifying forecasting problems, applying a range of forecasting methods, selecting appropriate methods for each problem over time, and evaluating and updating these methods over time. This emphasizes equally the strong need for online adaptive forecasting methods.

In the early stages of a forecasting project, the problem has to be well-defined so that the correct decision about what should be forecast is made. For example, in manufacturing, where forecasts are required for quality prediction, it is necessary to make clear whether forecasts are needed for every product line or group of products. Should it be based on weekly, monthly, or annual data? It is also necessary to consider the suitable forecasting horizon for the application. The frequency with which the forecasts need to be produced is also application-dependent. Usually, forecasts that need to be produced frequently are better done using an automated system than with methods that require careful manual work. For many applications, exchange with domain experts is also required to ensure an adequate understanding of their needs and how the forecasts are to be used efficiently before launching extensive work in producing the forecasts.

Once the forecasting problem requirements are well-defined, it is then necessary to collect the data that will be used to produce the forecasts. In the current information era, recorded and stored data availability is continuously increasing. The forecaster's task is often to identify where and how the required data is stored. It has been reported that the largest part of a forecast expert's time is spent in locating and collating the available data before developing suitable forecasting methods [54].

2.4.1 Definitions

Definition 8 ***Univariate Time Series Forecasting*** *Given a historical realization of the time Series X process, recorded until time t, $\{x_1, x_2, \cdots, x_t\}$, the goal is to forecast the realization of this process at a horizon h. In other words, predict the value of X at $t+h$, i.e., x_{t+h}.*

Definition 9 ***Multivariate Time Series (MTS) Forecasting*** *Given Historical realizations of time series processes $X^i, \forall i \in [1, N]$ and $\mathbf{X}_{1:t}$ denotes its corresponding Euclidean space defined in Definition 7, the goal is to forecast all the N realizations of $\mathbf{X}$ at time instant $t+h$.*

Note that for MTS forecasting, it is possible to define one target time series X^i with $i \in [1, N]$ and the goal is then to predict the value of X^i at $t+h$, x^i_{t+h} using MTS forecasting techniques.

2.4.2 Basic Steps

A forecasting task usually involves five basic steps.

1. Problem definition: This is the most critical step. Defining the problem precisely requires a correct understanding of the task, who requires the forecasts, and how the forecasts will be used in the application. To do this, a forecaster needs to spend some time talking to

domain experts who can be involved partially in collecting data, maintaining databases, and using the forecasts for future planning.

2. Data and Information Collection: Two types of information are most often required, namely historical data and domain knowledge collected from the accumulated expertise of the experts involved in the application. Sometimes it is difficult to collect enough historical data due to, for example, the high costs related to data collection and storage in many applications or data privacy and legal issues such as in the healthcare field. Old historical data may also be outdated due to structural changes in the data generating process. In this case, one may only use the most recent observations. However, it should be noted that dynamic adaptive time series analysis methods can adjust the time window involved in the learning process to overcome this issue [98], [99].

3. Exploratory Time Series Analysis: This analysis aims to discover the time series' main component, such as the trend or seasonality components. It also investigates the importance of such component and their relation to the dynamics of the time series. It is also important to determine the available time series variables in the case of MTS data and investigate the relationships between them. Various tools are available to support this analysis. Further details are provided in Chapter 3.

4. Model Selection and training: The model selection consists of choosing the best forecasting model to be used given the available historical time series data, the existing relationships between time series variables, if many are available, and any other existing explanatory variable. Model selection usually refers to the selection of the best model parameters for one specific model, e.g., the parameters of the Auto-Regressive Integrated Moving Average (ARIMA) model. It can also denote the selection between many forecasting models. To generate these models initially, different hypotheses about the data and the forecasting task modeling have to be drawn. Further details about forecasting models are provided in Sections 2.4.3, 2.4.4 and 2.4.5.3.

5. Forecasting Model Evaluation: Once a model is selected and its parameters are estimated, it can then be deployed to produce forecasts. However, the evaluation is only done once the true values of the time series become available for that forecast period. Several metrics have been proposed for assessing forecasting accuracy. Some of these metrics are presented in Section 2.4.6.

2.4.3 Simple Forecasting Models

Some forecasting models are designed based on simple modeling assumptions. These models have been shown to be effective in many applications [97]. In this section, four simple forecasting models that are usually used as benchmarks are presented.

2.4.3.1 Mean Model

Using this model, the future forecast values are assumed to be equal to the mean value of the historical data. Then assuming that measurements of X are collected until time t, the forecast value $hatx_{t+h}$ of X at $t+h$ can be written as:

$$\hat{x}_{t+h} = \bar{X}_{1:t} = \frac{(x_1 + \cdots + x_t)}{t}. \tag{2.30}$$

2.4.3.2 Naïve Model

For the Naïve model, we simply consider the forecast value to be exactly equal to the value of the last observation. That is,

$$\hat{x}_{t+h} = x_t \tag{2.31}$$

This method is considered to be quite well-performing for many economic and financial applications [54].

2.4.3.3 Seasonal Naïve Model

A similar model to the Naïve model that is useful for time series data depicting high seasonality is given by the Seasonal Naïve Model. In this case, each forecast is set to be equal to the last observed value from the same season. Formally, the forecast value at time $t+h$ is given by:

$$\hat{x}_{t+h} = x_{t+h-m(k+1)} \tag{2.32}$$

where m is the seasonality period, and k is the integer part of $\frac{h-1}{m}$ (i.e., for example, the number of complete years in the forecast period prior to time $t+h$). For example, with weekly data, the forecast for all future Tuesday values of transportation demand is equal to the last observed Tuesday values. Similar rules are applicable to monthly and quarterly data and for other seasonal periods.

2.4.3.4 Drift Method

A variation of the Naïve model that allows the forecasts to increase or decrease over time, with the amount of change in the time series over time, here is called the Drift method. The forecast value is set to the average change observed in the historical data. Therefore, the forecast value at time $t+h$ is computed by:

$$\hat{x}_{t+h} = x_t + \frac{h}{t-1}\sum_{j=2}^{t}(x_j - x_{j-1}) = x_t + h\left(\frac{x_t - x_1}{t-1}\right). \tag{2.33}$$

This is equivalent to drawing a line between the first and last observations and extrapolating it into the future.

2.4.4 General Auto-Regression Models

Auto-Regressive models are commonly used for time series forecasting. This family of models projects a time series into a Euclidean space according to Taken's method on time-delay embedding [100]. Following the common ML setting for regression tasks, a set of observations (z_t, x_t) is then constructed [9]. In each observation, each time series value x_t is modeled based on its own p past/lagged values, i.e., $z_t = \{x_{t-p}, \cdots, x_{t-2}, x_{t-1}\}$, where, $x_t \in \mathbb{X} \subset \mathbb{R}$, which represents the vector of values we want to predict, and $z_t \in \mathbb{Z} \subset \mathbb{R}^p$ represents the feature vector for x_t. The goal is then to build a regression model $\hat{f}$ for approximating $f : \mathbb{Z} \rightarrow \mathbb{X}$, where f denotes the true function. Hence, this approach is based on modeling the conditional distribution of the t^{th} value of the time series based on its previous p values: $f(z_t|x_t)$. Formally, this approach is a regression problem. The temporal dependency structure is then modeled using past observations as explanatory variables. This auto-regressive modeling is the essence of many important forecasting models in the ML literature, for example, ARIMA [61]. The time-delay embedding approach described above requires the determination of the number of past values or lags p to be used, i.e. the embedding dimension which represents how far back in time the auto-regressive process should go.

Over this section, we present forecasting models, for univariate time series, i.e single time series but most of them can be generalized to MTS, as the time-delay embedding can also be generalized to MTS data $\mathbf{X} = \{X^1, X^2, \cdots, X^N\}$, by maps a set of observations from each time series variable $X^r \in \mathbf{X}$ to a $p \times N$-dimensional feature space corresponding to the p past lagged values of each observation in each time series variable in $\mathbf{X}$. Each observation is then composed of a feature vector $z_t^r \in \mathbb{Z} \subset \mathbb{R}^{p \times N}$, which denotes the previous p values of each variable in $\mathbf{X}$, and a target vector $x_i^r \in \mathbb{X} \subset \mathbb{R}$, which represents the value of the time series variable we want to predict. The goal of MTS forecasting is then to construct a model $f : \mathbb{Z} \rightarrow \mathbb{X}$, where f denotes the regression function.

2.4.4.1 ARIMA Family of Models

In this section, we present a special family of Auto-Regressive models which is based on a combination of Auto-Regressive and Moving Average processes.

Auto-Regressive model X is considered to follow an Auto-Regressive process of order p denoted by AR(p) if it satisfies:

$$x_t = \phi_0 + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + \epsilon_t \tag{2.34}$$

where ϵ_t is a white noise and the parameters $\{\phi_j\}_{j \in [0,p]}$ are the AR model parameters. The forecast value of a series can be interpreted as a slight perturbation of a simple function of the most recent p observations.

Moving Average Series X is called to follow a Moving Average process of order q, MA(q), if it satisfies the following:

$$x_t = \theta_0 + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots \theta_q \epsilon_{t-q} \tag{2.35}$$

where ϵ_t is a white noise and $\{\theta_j\}_{j \in [0,q]}$ are the model parameters. It is easy to distinguish between MA and AR series by analyzing the behavior of their AutoCorrelation Function (ACF): the ACF of MA cuts off sharply while the ACF of AR decays in an exponential fashion. For ACF computation, see Equation 2.18.

Integrated Model In this approach, the differentiated series of X can be modeled using a white noise process. In general, the order of differentiation d is determined as the minimum number of times the series needs to be differentiated to reach stationarity. A random walk process is an example of an Integrated Model $I(d)$ of order d. For $d = 1$:

$$x'_t = x_t - x_{t-1} = \epsilon_t \tag{2.36}$$

where the values of the differentiated time series x'_t are just a function of the random term ϵ.

Auto-Regressive Moving Average Series X is assumed to be an Auto-Regressive Moving Average process of order (p, q), ARMA(p,q) if it satisfies:

$$x_t = \phi_0 + \phi_1 x_{t-1} + \cdots + \phi_p x_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \tag{2.37}$$

where $\{\theta_j\}_{j\in[1,q]}$ and $\{\phi_j\}_{j\in[0,p]}$ are the ARMA model's parameters.

Auto-Regressive Integrated Moving Average Model If a differentiated series of X of order d, is an ARMA(p,q) process, then X is said to be an Auto-Regressive Integrated Moving-Average series ARIMA(p,d,q). To fit the appropriate $ARIMA$ model to the time series X, the model orders p and q, and the differentiation order d need to be identified accurately. First, the integration or differentiation parameter d is set up to stationarise the time series and remove the responsible features of seasonality. For example, if the resulting time series depicts a strong trend, e.g. growth or decline, then the process is clearly non-stationary, and it should be differentiated at least once. The second test that can be used is to examine the estimated autocorrelation of the time series. For a stationary time series, the auto-correlations will typically decay rapidly to 0. For a non-stationary time series, the auto-correlations will typically decay slowly if at all. Suppose that we have identified a particular ARIMA(p,d,q) model which seems to represent a given time series. The second step would be to fit the identified model to the data to estimate its parameters. Fitting the model can be performed using Maximum Likelihood Estimation (MLE) procedures which allow producing both estimates and standard errors for the model parameters. Once a model is fitted on a time series sequence, it is important to assess the quality of fitness. This is because the quality of forecasts depends largely on the appropriateness of the fitted model. The standard way for evaluating the goodness of fit is through the evaluation of the residuals (i.e. deviations of predicted from actual empirical values of the data). A simple investigation is to simply plot the residuals and examine whether they are similar to a white noise series. After selecting the appropriate model, estimating its unknown parameters, and verifying that the model fits well the data, we can proceed to forecast the future values of the time series. Once the forecast $\hat{x}_{t+1}$ is obtained for $t + 1$, we can use it to obtain a forecast for x_{t+2} and then use these two forecasts to generate a forecast for x_{t+3}, and so on. The process can be then continued to obtain forecasts out to any horizon h in the future by computing $\hat{x}_{t+h}$. However, it is important to note that uncertainty increases as long as we predict further and further from the data. Therefore, we can expect the standard errors associated with the predictions to increase.

In practice, the Box-Jenkins approach [61] can be used for identifying, estimating, and evaluating $ARIMA$ models. The approach is based on a three-step iterative cycle of:

- Identification.
- Estimation.
- Verification

For identification, the data may require some pre-processing to make it stationary before the differentiation procedure. To achieve stationarity, some or a combination of the following operations may be required:

- Re-scale the time series data, e.g. by using a logarithmic or exponential transform.
- Remove deterministic components.

Then, the differentiation is carried out until stationarity is reached. In practice, usually, $d = 1$ or 2 should be sufficient. Assuming that the time series is stationary or made stationary. The model identification is performed by evaluating the sample AutoCorrelations (ACF) and Partial AutoCorrelations (PACF) (see Equations 2.18 and 2.19). First, we fit an ARMA(p,q) model to the differentiated series. It is important to note:

- An MA(q) process has negligible ACF after the q-th term.
- An AR(p) process has negligible PACF after the p-th term.
- An ARMA(p,q) process has k-th order sample ACF and PACF decaying geometrically for $k > max(p, q)$.

In the estimation procedure of an ARMA model, it is possible to start estimating the likelihood based on the early recorded time series observations. The identification procedure requires substantial intervention from forecast experts to compute the above statistics. Therefore, several attempts have been made to automate the model identification procedure. The most straightforward method fits a set of models to the time series, then selects the "best" model following some criteria. The most typical criteria to select the best model among a set of candidates are:

- Small Schwarz criterion or BIC criterion [101].
- Small Standard Error (SEE) [102].
- High coefficient of determination R^2 score [103].

The third stage in the Box-Jenkins procedure is to check whether the model fits the observed time series data. There are several ways to do so:

- Check for overfitting by adding extra parameters to the model, e.g. regularisation parameters, and using a likelihood ratio test or t-test to check that they are not meaningful [104].
- Analyzing residuals by computing them and using the autocorrelation functions, ACFs, PACFs, and spectral densities estimates, to check whether they denote a white noise process.

2.4.4.2 Exponential Smoothing

Similar to an AR(p) model, the Exponential Smoothing model [105] expresses the future time series observations as a linear combination of its past observations. In addition, the Exponential Smoothing model attributes weights to the past values, such that they follow an exponential decay giving lower weights, i.e. lower relevance, to older observations [54]. For instance, using a simple exponential smoothing method, the forecast at $t+1$ of X is given by:

$$\hat{x}_{t+1} = \eta_0 x_t + \eta_1 x_{t-1} + \eta_2 x_{t-2} + \cdots \tag{2.38}$$

where the $\{\eta_j\}_{j\geq 0}$ are the weights of past observations. Several exponential smoothing methods are presented in the literature. For a comprehensive reading, we refer to the work by Hyndman and Athanasopoulos (2018) [54].

2.4.4.3 Vector Auto-Regressive Model

In this section, we present the generalization of the AR process to MTS data. This is framed under the Vector Auto-Regressive (VAR) model [17], [106]. The VAR model is one of the most commonly applied models to handle dependencies among multiple random processes for forecasting purposes [67]. Similarly to ARMA models, VAR departs also from a stationarity assumption and possesses order p. It can be described as follows:

$$\begin{aligned} \hat{\mathbf{X}}_t &= {\mathbf{X}_{t-1}}^\top \Phi_1 + {\mathbf{X}_{t-2}}^\top \Phi_2 + \cdots + {\mathbf{X}_{t-p}}^\top \Phi_p + \boldsymbol{\epsilon}_t \\ &= \sum_{j=1}^{p} {\mathbf{X}_{t-j}}^\top \Phi_j + \boldsymbol{\epsilon}_t \end{aligned} \tag{2.39}$$

where $\boldsymbol{\Phi} = (\Phi_1, \ldots, \Phi_p) \in \mathbb{R}^{N\times N}$, where N is the number of variables in the MTS (See Definition 7), are the model's parameters while $\boldsymbol{\epsilon}_t$ is a vector white noise, i.e., $E(\boldsymbol{\epsilon}_t) = \mathbf{0}$.

The traditional VAR models share most of the properties, issues, and training schema as for the above-mentioned ARMA ones. Naturally, the parameter matrices can easily scale up to high dimensionality. Similarly to other regression frameworks, it is expected that the model complexity increases, thus leading (with a high likelihood) to sparse solutions.

2.4.5 Deep Neural Networks

2.4.5.1 Some Deep Learning Notions

Artificial Neural Networks (ANNs) have been successfully applied to a wide variety of learning tasks, including time series forecasting. ANNs are composed of a collection of connected units or nodes called artificial neurons which are conceptually inspired by biological neurons. Each connection is providing the output of one neuron as an input to another neuron and is assigned a weight that represents its relative importance. A given neuron can have multiple input and output connections. Basically, every single neuron takes some inputs, processes them, and computes an output by some function (usually non-linear and called activation function) of the sum of its inputs, weighted by the weights of the connections from the inputs to the neuron (a bias term is added to this sum). The initial inputs are a sample of the external data or its feature values. The ultimate outputs are intended to solve the learning task, such as forecasting the next value of a time series. Typically, neurons are aggregated into layers. The layer that receives external data is called the input layer. The layer that outputs the final prediction is called the output layer. In between these layers, there are zero or more hidden layers. Single-layer and unlayered networks can also be used. Between two layers, multiple connection patterns can exist. They can be fully connected such that every neuron in one layer is connected to every neuron in the next layer. They can be pooling, where a group of neurons in one layer is connected to a single neuron in the next layer, resulting thus in reducing the number of neurons in that layer. They can be convolutional layers which are the major building blocks used in Convolutional Neural Networks (CNNs). As mentioned in Section 2.1.3.6, convolution is the simple application of a filter to an input that results in activation. Repeated application of the same filter to an input results in a map of activations called a feature map or activation map, indicating the locations and strength of a detected feature in the input, such as specific regions in an image. Neurons with connections forming a directed acyclic graph are called feedforward networks. Alternatively, networks that allow connections between neurons in the same or previous layers are known as Recurrent Neural Networks (RNNs). A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers.

2.4.5.2 Learning a Deep Neural Network

Deep Learning incorporates new representations of the raw data in $\mathcal{Z}$ through transformations $\phi : \mathbb{R}^p \to \mathbb{R}^{p'}$ that map the raw data to another space $\phi(\mathcal{Z}) = \mathbb{R}^{p'}$. In the learning process, i.e., $\phi = \phi_\theta$, is fitted to some sample of data. As we mentioned above, DNNs learn ϕ_θ in multiple levels of abstraction. Each level corresponds to one layer of the network. The overall DNN can be viewed as nested layer transformations $f(z_t) = f^{(l)}(f^{(l-1)}(\cdots f^{(1)}(z_t))$ where l is the depth of the network. Each layer $f^{(i)} : \mathbb{R}^{p^{(i-1)}} \to \mathbb{R}^{p^{(i)}}$ applies two operations. First, an affine transformation that consists of a weighted sum or a convolution is computed. Then, an activation function, e.g., tanh or ReLU activation, is applied. Each layer is parametrized by the weights used in the affine transformation, while the activation function is usually fixed. These weights are learned automatically using an optimization procedure that minimizes an objective loss function. In order to estimate the loss/error of some given weights values, a differentiable loss function is then defined. The most commonly used loss function in DNNs for the forecasting task is the Mean Square Error loss [107]:

$$L(x_t) = (x_t - \hat{x}_t)^2 \tag{2.40}$$

with $\hat{x}_t$ is the forecast value x_t. Similarly, we can define the average loss for a whole training composed, for example, of T pairs of (z_t, x_t) by:

$$J(\omega) = \frac{1}{T}\sum_{t=1}^{T} L(x_t) \tag{2.41}$$

with ω is the set of weights to be learned by the network. The loss function is minimized to learn the weights in ω using a gradient descent method:

$$w = w - \alpha\frac{\partial L}{\partial w}|\forall w \in \omega \tag{2.42}$$

with α is the learning rate of the optimization algorithm. By subtracting the partial derivative, the model is actually auto-tuning the parameters $w \in \omega$ in order to reach a local minimum of J. Since the partial derivative with respect to certain parameters $w \in \omega$ can not be in most of the cases computed directly, a chain rule of the derivative is employed using the well-known backpropagation algorithm [108].

2.4.5.3 Deep Learning for Forecasting

DNNs have shown the ability to automatically learn new, complex, and enriched feature representation from input data [109], thus achieving a good performance in solving a wide variety of tasks. Multi-Layer Perceptrons (MLPs) **mlp**, Recurrent-based Neural Networks such as Long Short-Term Memory Networks (LSTMs) [110], as well as Convolutional Neural Networks (CNNs) [111], have been widely used as State-of-the-Art neural networks architectures for solving the forecasting task [112], [113]. Many improvements over these network architectures have been proposed in the literature, ranging from optimizing the architecture structure to combining these networks together in one single forecasting task [113]–[115].

2.4.5.4 Multi-Layer Perceptron

The MLP usually consists of an input layer, one or more hidden layers, and an output layer. The input layer has p variables, i.e., the dimension of z_t, and it is fed to the first hidden layer. The neuron output from the first hidden layer is fed as the input to the following hidden layer, and so on. The neuron output from the last hidden layer is fed to the output layer. The neuron's output is given by:

$$x_t = f(\omega z_t + b) = f(\sum_{j=1}^{p} w_j x_{t-j} + b) \tag{2.43}$$

where z_t is the input variable, f is a nonlinear activation function, $\omega = \{w_j\}_{j \in [1,p]}$ and b are the weight and bias of the linear transformation. Rectifier linear unit (ReLU) is most often applied as an activation function in the hidden layers, as shown in Equation 2.44, and the output layer does not apply any activation function in the forecasting task.

$$f(a) = \{0, \text{if } a < 0 \text{ and } a, \text{if } a \geq 0\} \tag{2.44}$$

The MLP models the relationship between input and output pairs by learning from the recorded time series data. Mean squared error (MSE) is applied as a loss function (see Equation 2.40). Figure 2.7 shows an example of an MLP architecture for the task of time series forecasting. The input layer uses the p-lagged values for each time series data point x_t, represented by the vector of values z_t. These p-lagged values are considered as the input feature and are fed to the first hidden layer that has n neurons. The neuron output from the first hidden layer is fed as input to the next hidden layer, and so on, until the last hidden layer. Finally, the last hidden layer output feeds into the output layer.

It should be noted that there are some problems related to the application of MLPs for time series forecasting. For instance, the length of the input sequence to be fed to the network, i.e. the number of lagged values p to be used, is difficult to determine. In addition, it is difficult

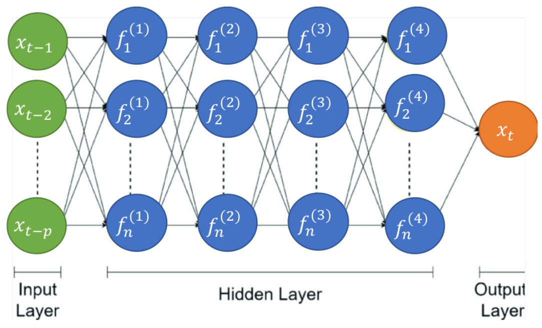


Figure 2.7: MLP architecture for time series forecasting.

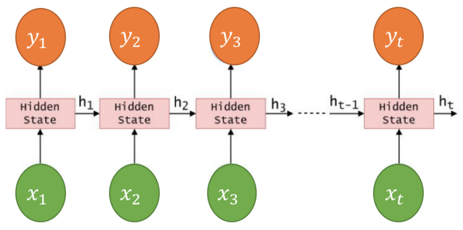


Figure 2.8: Unfolded basic RNN architecture for time series forecasting.

to achieve convergence when learning the network's weights and easy to fall into the local minimum problem [116]. The network structure, e.g., the number of hidden layers, is also hard to determine beforehand.

2.4.5.5 Recurrent Neural Networks

The Recurrent Neural Network (RNN) is a class of neural networks that allow output from some nodes to affect subsequent input to the same nodes. This allows it to exhibit temporal dynamic behavior. Opposingly to MLPs, RNNs do not consume all the input data at once. They display a sort of memory by handling sequential data, accepting the current input data, and previously received inputs. They can memorize previous inputs due to their internal memory. At each step, the RNN does a series of calculations before producing an output. The output, known as the hidden state, is then combined with the next input in the sequence to produce another output. This process continues until the model is programmed to finish or the input sequence ends. An example of unfolded basic RNN architecture is shown in Figure 2.8, x_t is the input at time t that will be fed into the hidden state (h_t), and y_t is the output. The hidden states computations are shown in Equations 2.45 and 2.45.

$$h_t = \tanh(\omega_{hh} h_{t1} + \omega_{xh} x_t) \tag{2.45}$$

$$y_t = \omega_{hy} h_t \tag{2.46}$$

Even though RNNs have a timing concept in their architecture and seem to be more suited for handling time dependencies in a sequence of time series observations for forecasting purposes, they have serious problems, most gravely the rapid gradient degradation, also known as the vanishing gradient problem [116] meaning that in training, the gradient becomes too small, and the parameter updates become insignificant. This makes the learning of long data sequences difficult. LSTMs are a special kind of RNNs capable of learning long-term

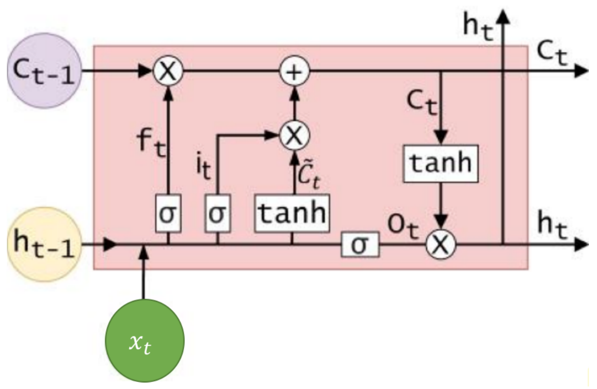


Figure 2.9: Hidden layer neuron structure of an LSTM neural network.

dependencies by remembering information for long periods. To do so, some modifications are introduced to the computation of the hidden states. LSTMs were first introduced by Hochreiter and Schmidhuber (1997) [117] and were refined and popularized by many following works [70], [110], [118], [119]. The Hidden layer neuron structure of an LSTM neural network is described in Figure 2.9.

We can distinguish four gates:

- Forget Gate (f_t):
$$f_t = \sigma(\omega_f.[h_{t-1}, x_t] + b_f) \tag{2.47}$$
where σ is a sigmoid function, t is the current time step, f_t is the forget gate at time t, x_t is the input, h_{t-1} is the previous hidden state, ω_f is the weight matrix between forget gate and input gate, and b_t is the connection bias at time t.

  The forget gate is devised to select which information needs attention and which can be ignored. The information from the current input x_t and the hidden state $h_(t-1)$ are passed through the sigmoid function σ. Sigmoid outputs values between 0 and 1 to decide whether the part of the old output is necessary to keep by generating a value close to 1. This value output by f_t will be used later by the cell for point-by-point multiplication.

- Input Gate (i_t):
$$i_t = \sigma(\omega_i.[h_{t-1}, x_t] + b_i) \tag{2.48}$$
$$\tilde{C}_t = \tanh(\omega_{\tilde{C}}.[h_{t-1}, x_t] + b_{\tilde{C}}) \tag{2.49}$$
where t is the current time step, i_t is the input gate at time t, ω_i is the weight matrix of the sigmoid operator between the input and output gates, and b_i is the bias vector, and $\tilde{C}_t$ is the value generated by tanh, and $\omega_{\tilde{C}}$ is the weight matrix of the tanh operator between cell state information and the network output and $b_{\tilde{C}}$ is the bias at t w.r.t $\omega_{\tilde{C}}$.

  The input gate computes a set of operations to update the cell status. First, the current state x_t and previously hidden state h_{t-1} are passed into the second sigmoid function. The output values are converted to 0 for referring to "important" and 1 for "not important." Second, the same hidden and current state information is forwarded to the tanh function, which outputs the vector $\tilde{C}_t$ that produces values between -1 and 1 to regulate the network. These output values computed by the activation functions will also be used for point-by-point multiplication in the cell gate.

- Cell Gate (C_t):

$$C_t = f_t.C_{t-1} + i_t.\tilde{C}_t \tag{2.50}$$

where t is the current time step, C_t is the cell state information, f_t is the forget gate at time t, i_t is the input gate at time t, C_{t-1} is the previous cell state and $\tilde{C}_t$ is the value generated by tanh at time t. After receiving the computed information from the forget gate and input gate, this step is devised to select and store the information from the new state in the cell state. To do so, the previous cell state C_{t-1} gets multiplied with forget vector f_t. If the output is 0, then values will get dropped in the cell state. Next, the network takes as output the value of the input vector $i(t)$ and computes point-by-point addition, which updates the cell state giving the network a new cell state $C(t)$.

- Output Gate (o_t):

$$o_t = \sigma(\omega_o.[h_{t-1}, x_t] + b_o) \tag{2.51}$$
$$h_t = o_t.\tanh(C_t) \tag{2.52}$$

The output gate computes the value of the next hidden state, which in its turn contains information from previous inputs. The values of the current state and previous hidden state are forwarded to the third sigmoid function. Then, the new cell state generated from the cell state is passed through the tanh function. These two outputs are multiplied point-by-point. Based on the final value, the network makes the decision on which information the hidden state should receive. This hidden state is used for prediction. Finally, the new cell state and new hidden state are brought to the next time step.

To sum up, the forget gate f_t determines which pertinent information from the prior steps is required. The input gate selects which relevant information can be counted in the current step, and the output gate finalizes the next hidden state.

2.4.5.6 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are initially designed for image data by considering feature extraction from two-dimensional data. A similar one-dimensional architecture of CNNs can be used for univariate time series classification, and forecasting [111], [120].

In the case of a univariate time series, convolution can be interpreted as applying and sliding a filter over the time series. Opposingly to the case of image data, the filters operate only on one dimension, i.e., time dimension, instead of two dimensions like the case for images, i.e., width and height. The filter can also be considered as a generic non-linear transformation of the raw time series data. A general form of applying the convolution on the time series X

at a centered time stamp t is given by:

$$C_t = f(K \circ X_{t-\frac{l}{2}:t+\frac{l}{2}} + b) \tag{2.53}$$

with K is a filter of length l, $X_{t-\frac{l}{2}:t+\frac{l}{2}}$ is the subsequence of the time series X centered at time t, $\circ$ is the convolution operator (a discrete convolution of two one-dimensional signals f and g, written as $f \circ g$, is defined as $f \circ g(i) = \sum_{j=-\infty}^{\infty} f(i)g(i-j)$ and if the signals are finite, the infinite convolution may be truncated), b is a bias parameter, and f is a non-linear function such as the Rectified Linear Unit (ReLU). Several filters are usually applied to the time series. An intuition behind applying several filters on an input time series would be to learn multiple new features useful for the forecasting task [121]. In order to learn automatically the values of the filter K, the convolution should be followed by a discriminative classifier. It should be noted that some pooling operations, such as average or maximum computation, can also be applied after the convolution to reduce the time series length by aggregating it over a sliding window. A batch normalization operation can also be performed to accelerate the network's convergence. In the context of time series, batch normalization is performed over each channel, preventing thus the internal covariate shift across one mini-batch training of the time series.

The architecture of a CNN used for time series forecasting usually consists of an input layer, a convolutional layer, a pooling layer, a flattened layer, a fully connected layer, and an output layer. The input features are fed into the convolution layer. A filter is applied to an input feature in the convolution layer to produce a feature map. The activation function is applied to the results. The output from the convolution layer is fed to a pooling layer which performs an aggregation operation to reduce the size of the feature map. The pooled feature map is passed into a flattened layer to convert the data into a one-dimensional array for inputting it into the next layer. The flattened layer output is fed to a fully connected layer. The weights are applied to process the data in the fully connected layer. The fully connected layer output is fed into the output layer. The ReLU is usually applied in the convolution layer as an activation function. The activation function is not applied to the remaining layers. An example of a one-dimensional convolutional neural network (1D-CNN) is shown in Figure 2.10.

2.4.6 Forecasting Evaluation Metrics

Evaluation metrics are required to quantify the predictive performance of a forecasting model. Typically, one way is to measure the deviation between the predicted values output by the forecasting model and the true values of the time series. However, there is no single unified way for measuring this deviation, i.e., error, that performs well under all situations [54]. In fact, it is difficult to define an error measure that works for any type of non-stationarity in the time series [1]. In addition, selecting the appropriate error metric for a given forecasting task is highly domain and application-dependent. This is why evaluating forecasting models' performance is still an active area of research [1], [122]. In the following, we present four error measures that though having some problems, are still commonly used for the evaluation of forecasting models' performance. These measures are the Mean Absolute Scaled Error (MASE) [123], the symmetric Mean Absolute Percentage Error (sMAPE) [124], the Mean Absolute Error (MAE) [125], and the Root Mean Squared Error (RMSE).

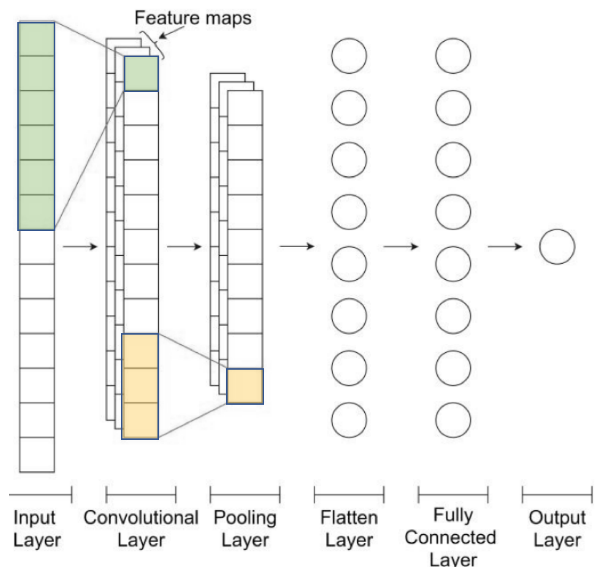


Figure 2.10: One-dimensional convolutional neural network (1D-CNN) architecture for the time series forecasting.

$$MASE = \frac{\sum_{t=1}^{T} |x_t - \hat{x}_t|}{l_{naive}} \tag{2.54}$$

where l_{naive} represents the average loss of the naive method in the training set used to build the respective forecasting model.

$$sMAPE = \frac{100\%}{n} \sum_{t=1}^{T} \frac{|x_t - \hat{x}_t|}{\frac{x_t + \hat{x}_t}{2}} \tag{2.55}$$

$$MAE = \frac{\sum_{t=1}^{T} |x_t - \hat{x}_t|}{T} \tag{2.56}$$

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T} (x_t - \hat{x}_t)^2}{T}} \tag{2.57}$$

For time series datasets that contain zero-valued observations, computing the sMAPE error measure may result in divisions by zero. Therefore, one variant of the sMAPE was presented in [126] to solve the issue of small values and divisions by zero of the original sMAPE by adding an additional component to the denominator of sMAPE. Furthermore, sMAPE breaks symmetry because under-prediction gives higher errors than over-prediction, even though the computed absolute error looks the same. The MASE avoids the problems stated for the sMAPE metrics such as symmetry issues and divisions by zero. Its only issue arises when the naive forecast has very different performance for different parts of the time series (e.g., different time windows). As a general guideline, scale-dependent error measures such as the MAE or the RMSE are advised to be considered first [1]. In fact, they are straightforward to interpret and do not share many of the drawbacks of the other measures. MAE is minimal for a forecast that is the median of the forecasting distribution, while RMSE is minimal for its mean. As such, the only drawback of the RMSE is the high sensitivity toward outliers.

For the MAE, in the case of a series with over 50% zeros, called intermittent time series, the best-performing model will be the one with a forecast heavily biased towards zero.

2.5 Model Selection

Forecasting models' performance estimation is not only used for reporting to the end-user an estimate of the expected generalization ability of a given forecasting model but also for performing the adequate model selection. Model selection is usually performed among a set of possible alternative models that can belong to the same family but have different parameter settings or to different families of models. A significant amount of work in the literature addresses the former for the time series forecasting task [122], [127]–[129]. In this work, we address the latter problem.

The main goal of model selection is to determine which model yields the best predictive performance because the exact model that describes the time series generating process is unknown. Therefore, model selection is usually based on comparing competing models and selecting the best that models accurately the time series. Model selection is a challenging task in time series analysis in general and in time series forecasting in particular. This is due to the time-changing nature of time series, which in turn affects the forecasting models [130]. An improper model selection leads to choosing a poor model with low prediction accuracy. As we mentioned, a forecasting model is an approximation to the time series generative process. Thus, it is crucial to reject a model far from reality and select the model that enhances the quality of such approximation [128].

Several methods for model selection have been presented in the literature over the last decades. For instance, in [131], [132], the authors claim that model selection should be conducted as a result of its predictive performance on an independent time series dataset, e.g., validation or test set. In this way, enhanced model performance can be achieved through adequate performance-based model selection. In this context, forecasting evaluation metrics can be used for model selection [133]–[135]. The Mean Squared Error is used in [133] as a criterion for model selection and claimed that it is more suitable than using the Residual Sum of Squares (RSS) which is also known as the Sum of Squared Residuals (SSR) or the Sum of Squared estimate of Errors (SSE) and is computed using the sum of the squares of residuals, a small RSS indicates a tight fit of the model to the data. Opposingly, authors in [134] suggested the use of RSS for model selection.

In [136], the authors argue that in addition to a good predictive performance on some data samples, model selection has to take into account how well the model approximates the true time series process. Therefore, many works [128], [132], [137] suggest the use of an Information Criterion that measures the quality of a model by taking into account how well it fits the data and its complexity. Akaike Information Criterion (AIC) [138], Bayesian information criterion (BIC) [139] and Structural Risk Minimization (SRM) [140] are the most widely used information criteria for model selection [130], [141]. A comprehensive reading about these criteria for model selection can be found in [132]. The authors of [135] suggested using an unbiased Akaike Information Criterion (AICu) and argued that it outperforms AIC and the biased correction Akaike Information Criterion (AICc). However, in a recent study [142], it has been shown that AIC outperforms several other tools and methods of model selection. In [141], the author shows that the ultimate decision to use the AIC or BIC depends on many factors, including the loss function employed, the study's methodological design, the substantive research question, and the notion of a true model and its applicability to the study at hand.

To sum up, different methods and tools for model selection for the task of time series forecasting have been proposed in the ML literature ranging from domain or application-specific to general. However, it is also a piece of evidence that there is no unified method or way for model selection [141], but it is important to note that some of the procedures recommended in the literature, especially the methods that are based on computing statistical estimates for information criterion or data likelihood, are complex, laborious, and time-consuming, and even though more theoretical than empirical, are less effective [130]. In addition, most of the above-mentioned works present offline model selection methods in the sense that the selection is made once at a time and kept static in the forecasting stage. There have been some recent but very scarce attempts to address the problem of time series forecasting using online learning [143]. Again, most of these methods rely mainly on blind model selection updates in the sense that even though the model selection is performed in an online manner, it does not take into account explicitly the significant changes in the time series data and the models' performance over time [143]–[145]. For instance, In [144], an online approach that relies on regret minimization techniques is adapted to ARMA family of models. A new model is learned (i.e., new ARMA model parameters and coefficients are estimated) each time a new time series observation is acquired. Using the same learning fashion, an online learning method to estimate the parameters of ARIMA models by reformulating it into a full information online optimization task (without random noise terms) is presented in [145]. This is achieved by adopting a game-theory-based approach, where an online player sequentially executes a decision (i.e., ARIMA's coefficient setting) and then suffers from a loss that may be unknown to the decision maker ahead of time. It can be adversarial or even depend on the actions taken by the decision maker. More recently, an online model selection has been performed based on the predicted forecasting error [146]. Candidate models are ranked according to their predicted performance, i.e., model comparison. The model with the lowest predicted error measure gets selected. The selection is updated without any indication of whether the change in the candidate models' ranking, i.e., performance, is significant or not.

2.6 Ensemble Learning for Time Series Forecasting

Despite the wide range of models that have been developed and applied for time series forecasting, it is widely accepted that none of these models is universally valid for every forecasting application and every time series data [1], [9], [10], [53]. This statement is confirmed empirically through the experiments conducted by Aiolfi and Timmermann (2006) [147], which reported the relatively different performance of the forecasting models over different time series. A larger empirical evaluation is presented in [1] where an experimental evaluation of 13 different baseline forecasting models on 58 data sets that include both real-world and competition time series datasets covering various domains is reported. The results of the evaluation showed that none of the models is the best performing in all the cases. Instead, these models have a large variance in performance over different time series datasets. This also seems to be a particular case of the No Free Lunch theorem by Wolpert (1996) [21], which states that no learning algorithm is best suited for all the learning tasks. Furthermore, models have a time-dependent performance even for the same forecasting application. In other words, forecasting models' performance change over time. This has also been confirmed in the study carried out by Aiolfi and Timmermann (2006) [147] that have shown that while some forecasting models denote a varying relative performance over time and some others maintain good (or bad) throughout the whole time series. This behavior of the models is also time series dependent and varies from one case to another. In this context, one reasonable solution is combining forecasts of different models to obtain one desired forecast value. This is formally broached in the ML literature by ensemble learning. Ensemble methods [9], [11], [16], [29]–[31], [148] have been a very popular research topic during the last decades. The success of ensemble methods stems in part from the fact that they offer attractive solutions to a wide variety of learning problems from the past and the present, such as improving predictive performance [16], [29], learning from multiple physically distributed data sources [46], scaling inductive algorithms to large databases [149] and learning from concept-drifting data streams [16].

Ensemble construction can be divided into three main stages:

1. Ensemble members generation: This stage consists of learning single individual models composing the ensemble. To do so, M possible hypotheses are formulated to model a given learning task. This results in M different individual models called base models or ensemble members.

2. Ensemble pruning: In this stage, only a subset of $k < M$ hypotheses is kept. This stage is devised to reduce the ensemble size and corresponding resource consumption while promoting its performance.

3. Ensemble aggregation: The kept hypotheses from the pruning stage are aggregated together into one single model using some aggregation rule, e.g., majority voting or averaging.

In this thesis, we are particularly focusing on linearly aggregated ensembles, i.e., linearly weighted, where the aggregation rule is time-dependent. In other words, the weights are set sequentially following an online learning fashion to cope with the time-changing nature of time series data.

Definition 10 ***Sequential Convex Ensemble Aggregation for Time Series Forecasting***

Denote with $\mathbb{P} = \{f_1, f_2, \cdots, f_M\}$ a pool of M forecasting models trained to approximate a true unknown function f that generated the time series X. Let $\mathbf{\hat{x}_{t+h}} = (\hat{x}^{f_1}_{t+h}, \hat{x}^{f_2}_{t+h}, \cdots, \hat{x}^{f_M}_{t+h})$ be the vector of forecast values of X at a future time instant $t+h, h \geq 1$ (i.e. x_{t+h}) by each of the models in $\mathbb{P}$. An ensemble model $\bar{f}_{\mathbb{P}}$ of $\mathbb{P}$ at time instant $t+h$ can formally be expressed as a convex combination of the forecasts of the models in $\mathbb{P}$.

$$\bar{f}_{\mathbb{P}}(\mathbf{\hat{x}_{t+h}}) = \sum_{i=1}^{M} w_{i,t+h} \hat{x}^{f_i}_{t+h} \tag{2.58}$$

where $\mathbf{w}_{t+h} = \{w_{i,t+h}\}_{i\in[1,M]}$ are the ensemble weights. The weights are constrained to be positive and sum to one. This constraint is necessary for some of the following results. These weights are the result of some adopted aggregation rules.

The aggregation rule can be determined or learned using several approaches. Further details are provided in Section 2.6.3.

2.6.1 Ensemble Error Decomposition

The expected error of the ensemble of the models in $\mathbb{P}$ $e_{\bar{f}}$ at a future data point x_{t+h} can be expressed as follows [150]:

$$\begin{aligned} e_{\bar{f}}(x_{t+h}) &= (x_{t+h} - \bar{f}(\hat{x}_{t+h}))^2 \\ &= \sum_{i=1}^{M} w_{i,t+h}(x_{t+H} - \hat{x}^{f_i}_{t+h})^2 - \sum_{i=1}^{M} w_{i,t+h}(\hat{x}^{f_i}_{t+h} - \bar{f}(\hat{x}_{t+h}))^2 \\ &= \bar{e}(x_{t+h}) - \bar{a}(x_{t+h}) \end{aligned} \tag{2.59}$$

The left term in Equation 2.59 refers to the weighted average error of the base models $\bar{e}$ and the right term to the ensemble ambiguity $\bar{a}$ which is simply the variance of the ensemble around the weighted mean and it measures the disagreement between the models, i.e. ensemble members, on x_{t+h}. The trade-off between these two determines how well the ensemble performs at this data point. The above relations can be averaged over several H time steps and the ensemble generalization error can be written as:

$$\begin{aligned} E_{\bar{f}} &= \frac{1}{H}\sum_{h=1}^{H} e_{\bar{f}}(x_{t+h}) \\ &= \frac{1}{H}\sum_{h=1}^{H}(\bar{e}(x_{t+h}) - \bar{a}(x_{t+h})) \\ &= \bar{E} - \bar{A} \end{aligned} \tag{2.60}$$

Equation 5.9 expresses the decomposition into bias and variance in the ensemble. If the ensemble is strongly biased, the ambiguity is expected to be small. This is mainly due to the single models implementing very similar functions and thus the agreement between them is expected to be big even on data points outside the training set. Therefore, the generalization error of the ensemble will be essentially equal to the weighted average of the generalization errors of its members. However, in the opposite case, if the ambiguity is high, i.e. there is a large variance, and in this case, the generalization error will be smaller than the average generalization error. In addition, it is important to note that $E_{\bar{f}}$ is positive. Therefore, $\bar{A}$ can be viewed as a lower bound for $\bar{E}$. This explains why promoting diversity by increasing ambiguity leads at some point to an increase in the average error of the ensemble members. Therefore, a trade-off between the average of the members' errors and the ambiguity should be established.

In the case of an equally-weighted ensemble with fixed weights over time, i.e., $w_{i,t+h} = \frac{1}{M}, \forall i \in [1, M], \forall h \in [1, H]$, $e_{\bar{f}}$ can be arranged and decomposed into bias, variance and covariance [148]. Since we train the single models $f_i, \forall i \in [1, M]$ on a training time series data, i.e., a sequence of the time series X of some length T_{train}, which is a historical realization of some random process that generates X, and the pool of models $\mathbb{P}$ may include some models that require some parameters' initialization drawn according to a random variable, e.g., the weights of a neural network, we use the expectation operator $E\{.\}$ is with respect to these random variables:

$$e_{\bar{f}}(x_{t+h}) = \overline{bias}^2 + \frac{1}{M}\overline{var} + (1 - \frac{1}{M})\overline{covar} \tag{2.61}$$

with

$$\overline{bias} = \frac{1}{M}\sum_{i=1}^{M}\left(E\{f_i(x_{t+h})\} - < x_{t+h} > \right) \tag{2.62}$$

where $< x_{t+h} >$ denotes the expected value of X at time $t+h$ given the noise.

$$\overline{var} = \frac{1}{M}\sum_{i=1}^{M} E\{\left(f_i(x_{t+h}) - E\{f_i(x_{t+h})\}\right)^2\} \tag{2.63}$$

$$\overline{covar} = \frac{1}{M(M-1)}\sum_{i}\sum_{j,j\neq i} E\{\left(f_i(x_{t+h}) - E\{f_i(x_{t+h})\}\right)\left(f_j(x_{t+h}) - E\{f_j(x_{t+h})\}\right)\} \tag{2.64}$$

The decomposition presented in Equation 2.61 shows that in addition to the bias and variance of the individual ensemble members, the ensemble error also depends on the covariance between the members. While the bias and variance terms are restrained to be positive, the covariance between the single models can take some negative value. The covariance term reflects some extent the diversity between the ensemble members and appears as an extra degree of freedom in the bias-variance dilemma. This extra degree of freedom is beneficial in the sense that it allows the ensemble model to approximate functions that are difficult to determine using a single model [151].

It is also important to note the differences between the two decompositions. Opposingly to the error-ambiguity decomposition (Equation 2.59), the bias-variance-covariance (Equation 2.61) does take into account the distribution over possible training sets or possible single models' parameter initialization [152]. Therefore, it is deemed to be more useful since we are generally interested in, of course, the expected error on future time series data points given these distributions.

2.6.2 Ensemble Pruning

Ensemble pruning has been widely studied in the literature for classification problems [29], [30], [153]–[156]. Few works tackled this issue for time series forecasting, especially in an online dynamic fashion [157]. Therefore, this section discusses the works on ensemble pruning in general and shows which methods have been transferred from classification or devised for time series forecasting. Ensemble pruning is a desirable and widely popular method to overcome the deficiency of high computational costs of traditional ensemble learning techniques and improve their 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, e.g., decision trees [28]. The optimization of run-time overhead is imperative for certain applications where real-time requirements have to be met, such as in online forecasting. In addition, when models are distributed over a network, the reduction of models leads to a reduction in the resulting communication costs [30]. Furthermore, both theoretical and empirical studies have shown that ensemble performance depends on the performance of its members and how diverse they are [29], [148], [154], [155]. All the previous works agree that encouraging diversity is beneficial for enhancing the ensemble's performance, but it is hard to tell the theoretical properties of diversity in an ensemble [29]. Therefore, understanding and promoting ensemble diversity remains an important research question for ensemble learning.

The generalization performance of an ensemble depends on its empirical error and diversity [29]. It is thus reasonable to design the model selection criterion accordingly. However, it is very challenging to decide which models exactly need to be selected. Hence, given a set of base learners, it is not easy to estimate the generalization performance of a sub-ensemble. In addition, finding the optimal subset is a combinatorial search problem with exponential computational complexity. Thus, it is infeasible to compute the exact solution by exhaustive search and an approximate search is needed. Several methods have been proposed in the ML literature to solve this issue by searching near-optimal solution either directly using global search [158]–[160] or iteratively using greedy search [154], [161]. Greedy search methods can be further divided into greedy *forward* pruning which starts with an empty set and iteratively adds the learners optimizing a certain criterion, and greedy *backward* pruning that starts with the complete ensemble and iteratively eliminates learners. It has been shown that greedy pruning methods are able to achieve comparable performance and robustness with global search methods but at much smaller computational costs [29]. Both global and greedy search methods can be further divided into three main families based on the paradigm used for the search [30].

The first family encloses ranking-based methods where the ensemble members are sorted according to a selection criterion. Kappa pruning [28] is amongst the most popular methods in this family for classification and uses a diversity measure for the selection. It ranks all pairs of base classifiers based on the κ statistic of agreement calculated on the training set. Kappa pruning could be applied to regression or forecasting by formulating a suitable pairwise diversity measure. In this context, Ma et al. [162] transferred several selection criteria such as Complementarity [163], Concurrency [164], and Reduce Error [28], for rank-based ensemble pruning using a *forward* greedy search procedure, to time series forecasting. It was shown that Complementarity and Reduce Error have the same flaw since they can not guarantee that the base model supplementing the ensemble the most at a given iteration is the one that will be selected. This is mainly explained by the fact that the forecasting error is directional. Therefore, it is not very reasonable to only focus on decreasing the value of the forecasting error while ignoring its direction. Reduce Error-trend that takes into account the trend of the time series and the direction of the forecasting error, is then suggested by the authors to mitigate the above issue.

The second family relies on clustering [165], [166]. Methods of this family include two stages. First, a clustering algorithm is employed in order to discover groups of models that are similar, i.e., make similar predictions. In the second stage, a selection of each cluster's representative is performed to increase the overall diversity of the ensemble. The main issue in this method is the choice of the similarity measure according to which the models will be evaluated and clustered, as well as the optimal number of clusters, i.e. the resulting ensemble size after pruning [30].

The third family of methods includes optimization-based methods. Different optimization techniques can be employed, including genetic algorithms [167], semi-definite programming [153], hill climbing [155] and meta-learning [168]. In the context of forecasting, a meta-learning approach for the estimation of ensemble forecast errors using regression is used in [157] to implement the selection procedure for each forecast. A more sophisticated approach inspired by classification is proposed for time series forecasting in [169], where Extreme Learning Machines (ELMs) and Hierarchical Extreme Learning Machines (H-ELMs) are integrated as the base models, i.e. ensemble members, and four distinct meta-attributes collections, i.e., hard prediction, local accuracy, global accuracy, and prediction confidence, are presented. Each set of meta-attributes is joined to a specific model performance assessment criterion, constructing thus a meta-data set. A meta-learner is trained on this data and used subsequently for deciding whether a base learner should be included in the ensemble or not.

This list of pruning method families is not exhaustive and several paradigms are used to serve this task. A comprehensive review can be found in [30]. However, most importantly for the time series domain, pruning methods need to be dynamic in order to cope with the time-evolving nature of time series that can be subject to significant changes, more precisely to the so-called concept drift phenomenon [11], [89], [170].

2.6.3 Ensemble Aggregation

This section is dedicated to briefly describing the State-of-the-Art approaches for ensemble aggregation, called also ensemble members combination. We enumerate their characteristics and limitations.

2.6.3.1 Averaging Approaches

One of the most common aggregation strategies in ensemble models is to simply compute the average of the outputs from its individual composing models. The averaging method can be improved further by performing ensemble pruning before aggregation [16]. An alternative method is to compute a weighted average. One suitable weighting schema for time series forecasting is to use a time-sliding window approach, in which, the weights are set to be inversely proportional to some ensemble error measure [12], [16]. In addition, a forgetting mechanism can be employed to the time window in order to increase the impact of the recent observations (i.e., most recent performance) [99], [171]. Essentially, these approaches are based on the assumption that the immediate future is more likely to resemble the most recent past. A method for dynamically adjusting the weights of an ensemble of LSTM networks is presented [171]. The weights at each time step are set in an adaptive and recursive way by using both past prediction errors and a forgetting weight factor.

2.6.3.2 Regret Minimization

The notion of regret was first introduced in Freund et al. (1997) [172] and compares the error suffered by a rule $\mathcal{R}$ to the one of a given single model f_i only on time instants when f_i was active, i.e. selected or gets non-null weight. We denote the regret of $\mathcal{R}$ with respect to f_i up to time T by $R_T(\mathcal{R}, f_i)$. First, a loss function ℓ_t evaluating the accuracy of the forecasts output by the rule $\mathcal{R}$ at each time instant t, has to be defined: $\ell_t : \mathcal{W} \to \mathbb{R}$:

$$\ell_t(\mathbf{w}_t) = \ell\Big(\sum_{j \in \mathbb{S}_t} w_{i,t}\hat{x}_t^{f_i}, x_t^{f_i}\Big) \tag{2.65}$$

for all $\mathbf{w}_t \in \mathcal{W}$, and $\mathbb{S}_t$ the set of single active models at a given time instance t, $\mathbb{S}_t \subset \mathbb{P}$. The goal of regret minimization-based approaches is to design sequential convex aggregation rules $\mathcal{R}$ with a small cumulative error $\sum_{t=1}^{T} \ell_t(\boldsymbol{p}_t)$. To reach this goal, the regrets $R_t(\mathcal{R}, f_i), \forall f_i \in \mathbb{P}$ (with respect to fixed experts, to fixed convex combinations of experts, or to sequences of experts with few shifts) should be small. The regrets are given by:

$$R_T(\mathcal{R}, f_i) = \sum_{t=1}^{T} \big(\ell_t(w_{i,t}) - \ell_t(\delta_i)\big)\mathbb{I}_{\{i \in \mathbb{S}_t\}}, \forall f_i \in \mathbb{P} \tag{2.66}$$

where $\delta_i \in \mathcal{W}$ is the Dirac mass on f_i (the convex weight vector with weight 1 on f_i).

These approaches assume that both time series true and predicted values are bounded so does their associated loss functions (i.e convex and bounded). In the following, we present the widely used methods in the Literature that use the principle of regret minimization [173]–[177], the Exponentially Weighted Average forecaster (EWA), the Fixed Share forecaster (FS), and the POLynomially weighted average forecaster with MuLtiple learning rates (MLPOL).

The Exponentially Weighted Average Aggregation Rule: referred to as EWA is an online convex aggregation rule introduced in learning theory by Vovk (1990) [178]. It relies on a parameter $\eta > 0$ and will thus be denoted by $\mathcal{E}_\eta$. At the time t, it assigns to the single model f_i the weight $\hat{w}_{i,t}$. It chooses $\hat{w}_{i,1}$ to be the uniform distribution over $\mathbb{S}_1$ and uses at time instance $t \geq 1$ the convex weight vector $\hat{\mathbf{w}}_t$ such that:

$$\hat{w}_{i,t} = \frac{e^{\eta R_{t-1}(\mathcal{E}_\eta, f_i)}\mathbb{I}_{\{f_i \in \mathbb{S}_t\}}}{\sum_{f_k \in \mathbb{S}_t} e^{\eta R_{t-1}(\mathcal{E}_\eta, f_k)}}, \forall f_i \in \mathbb{P} \tag{2.67}$$

which is exponentially small in the cumulative loss suffered so far by the single model. When the learning parameter η is properly tuned, it has a small average regret $R_T = O\left(1/\sqrt{T}\right)$ with respect to the best-fixed set of single models.

The Fixed Share Forecaster is first presented in [179]. It has the advantage of competing with both the best fixed single model and the best set of single models that may change only a small number of times. This is especially interesting in non-stationary environments, in which the best single model is changing most of the time and should thus be regularly updated. FS uses a learning parameter η, as well as a mixing parameter, called $\alpha \in [0, 1]$, that evaluates the number of changes in the sequence of single active models it is competing with. The initial weight distribution is uniform $\hat{\mathbf{w}}_0 = (1/M, \ldots, 1/M)$. Then, at each time instant t, the weights are updated twice. First, a loss update takes into account the new loss induced by every single model f_i:

$$\hat{\mu}_{f_i,t} = \frac{\hat{w}_{i,t-1} e^{\eta R_{t-1}(\mathcal{E}_\eta, f_i)} \mathbb{I}_{\{f_i \in \mathbb{S}_t\}}}{\sum_{f_k \in \mathbb{S}_t} \hat{w}_{k,t-1} e^{\eta R_{t-1}(\mathcal{E}_\eta, f_k)}}, \forall f_i \in \mathbb{P} \tag{2.68}$$

Second, another update is introduced to ensure that every single model receives a minimal weight $\frac{\alpha}{M}$ by assigning:

$$\hat{w}_{i,t} = (1-\alpha)\hat{\mu}_{f_i,t} + \frac{\alpha}{M} \tag{2.69}$$

This update captures the possibility that the best single model may have exchanged at time t. FS had some nice theoretical properties and vanishing average regret R_T with respect to sets of single models with few shifts.

The Polynomially Weighted Average Forecaster with Multiple Learning Rates MLPOL is inspired by the polynomially weighted average forecaster presented in [180]. The multiple learning rate addition is proposed by Gaillard et al. (2014) [173]. The weights setting rule by MLPOL is referred to as $\mathcal{M}$. First, the initial weights are set to be equal $\hat{\mathbf{w}}_0 = (1/M, \ldots, 1/M)$ and the regrets suffered by each model $\boldsymbol{R}_0(\mathcal{M}) = (R_0(\mathcal{M}, f_1), \ldots, R_0(\mathcal{M}, f_M)) = (0, \ldots, 0)$. Then, for each time instant t. The learning rates for each model $f_i \in \mathbb{P}$ are set such that:

$$\eta_{i,t-1} = 1 / \left(1 + \sum_{t'=0}^{t-1} \left(\ell_{t'}(\hat{\mathbf{w}}_{t'}) - \ell_{t'}(\delta_i)\right)\right)^2 \tag{2.70}$$

where $\delta_i \in \mathcal{W}$ is the Dirac mass on f_i (the convex weight vector with weight 1 on f_i). The next step consists of forming the weights mixture $\hat{\mathbf{w}}_t$ defined component-wise for each model f_i by:

$$\hat{w}_{i,t} = \eta_{i,t-1} \left(R_{t-1}(\mathcal{M}, f_i)\right)_+ \ / \ \boldsymbol{\eta}_{t-1} \cdot \left(\boldsymbol{R}_{t-1}(\mathcal{M})\right)_+ \tag{2.71}$$

where the symbol $_+$ denotes the vector of non-negative parts of the components of $R_{t-1}(\mathcal{M}, f_i)$. Then, the prediction by the weighted ensemble of single models is computed using Equation 2.58. Finally, the regret for every single model f_i is updated:

$$R_t(\mathcal{M}, f_i) = R_{t-1}(\mathcal{M}, f_i) + \ell_t(\hat{\mathbf{w}}_t) - \ell_t(\delta_i) \tag{2.72}$$

Gaillard et al.(2014) [173] proved the regret bound $\boldsymbol{R}_T(\mathcal{M}) = O(1/\sqrt{T})$ with respect to the best fixed single model. MLPOL is particularly interesting since it offers theoretical tuning of its learning parameters, opposingly to EWA and FS, which require some tuning of their learning parameters. The authors of [176] proposed an empirical tuning of the learning parameters of EWA and FS, which comes with no theoretical guarantees but works empirically well.

It is important to note that the above-described versions of EWA, FS, and MLPOL compete only with the best-fixed single model. However, they cannot per se ensure vanishing average regret $\boldsymbol{R}_T$ with respect to the best-fixed convex combination of single models. A reduction of the problem from competing with the best convex combination of a set of single models to the goal of competing with the best fixed single model can be helpful in this case. This reduction is a well-known trick in the literature on individual sequences and is usually referred to as the gradient trick [181].

2.6.3.3 Meta-Learning Strategies

Ensemble aggregation can also be solved using meta-learning. Meta-learning is defined as a way of modeling the learning process of a given learning algorithm, and it can be employed for ensemble combination to learn combination rules for the ensemble members using an ML algorithm [33], [182] and can be used for dynamic aggregation as well [9], [10]. In other words, another ML model called the meta-model is trained to learn the ensemble weights. To do so, a meta dataset is constructed such that each observation is composed of the single models' outputs at a given time instant t, treated as input features, joined to one target that consists of the time series true observed value at that time instant. Naturally, this method allows different types of models combination, i.e. linear and non-linear, depending on the used ML meta-model learning paradigm.

2.6.4 Bagging and Boosting for Time Series Forecasting

Bagging and boosting are considered the most two popular techniques for ensemble learning in a wide variety of learning tasks [183]–[185]. They have also been successfully applied to time series forecasting [186], [187]. Ensemble models based on the bagging and boosting paradigms are considered the State-of-the-Art ensemble methods for time series forecasting in many recent works [9], [10].

2.6.4.1 Bagging

Bootstrap aggregation (bagging) was first proposed by Leo Breiman [31]. Bagging is motivated by reducing the variance without increasing the bias of the predictions and thus helps to achieve better prediction accuracy [188]. In bagging, ensemble members are built using bootstrapping on the original data sample. Outputting a prediction for a new data point is achieved by applying all the base models to this new data point and then combining their predictions using an aggregation operation,e.g., averaging. In this way, bagging includes different forms of uncertainty about data modeling derived from the different hypotheses drawn from the different bootstrapped samples. This uncertainty covers data uncertainty, model uncertainty, and parameter uncertainty.

Even though it seems to be well-established and theoretically motivated in several machine learning contexts, it is just very recently that bagging has been applied successfully for the task of time series forecasting [128], [187], yielding for example very competitive results on the M3 competition dataset [142]. For time series data, the main challenge is the possible presence of non-stationarity sources in the data and the fact that auto-correlation must be taken into account when applying bootstrapping to the data. Hence, non-stationarity makes it very difficult to produce bootstrapped samples that share the same main characteristics of the original time series data. It should also be noted that in the context of time series forecasting, sometimes simple, low-variance models work quite well for many applications, rendering thus the expected benefit of bagging as a variance reduction technique less promising.

2.6.4.2 Boosting

Boosting-based methods adaptively perturb, re-weight, and re-sample training data to create the ensemble members. These methods work in an iterative fashion, focusing at each iteration on harder-to-learn data points, thereby outputting diverse predictions aggregated in a dynamic and adaptive fashion [189]. Therefore, the newest member is created to compensate for the instances incorrectly predicted by the previous members.

Reasoning on time series forecasting task, after applying time delay embedding of dimension p to the time series as explained in Section 2.4.4 and assuming we have T pairs of data point (z_t, x_t) resulting from the embedding procedure, the boosting algorithm executes J boosting iterations to approximate the true function f that generates the time series X by a model $\hat{f}$ such that $\hat{x}_t = \hat{f}(z_t)$ and some loss function $l(x_t, \hat{x}_t)$ is minimized.

In this section, we detail the AdaBoost algorithm, which is considered one of the most widely used boosting algorithms in research and practice [190]. More specifically, we present the gradient boosting procedure for mean regression, also called L_2 Boost [191]. Therefore, the l_2 loss is used $l_2 = (x_t - \hat{x}_t)^2$. The different steps of the gradient boosting algorithm can be written as follows:

1. Initialize the function estimate $\{hatx_t^[0]\}_{t\in[1,T]}$ with starting values. The unconditional mean and is a natural choice for mean regression.

2. Specify the pool of M ensemble members, and set $j = 0$. The members are just some simple regression models on a subset of the initial set of p-dimensional input variables z_t and a single time series value x_t. One way to build these models is to consider an additive model. In this manner, each base model is dependent on exactly one input variable out of the p variables, i.e., the p-lagged values for each value x_t of the time series. Note also that only one ensemble member gets selected for each boosting iteration.

3. Increment the number of iterations j by 1.

4. (a) Compute the negative gradient of the loss function evaluated at the function estimate of the previous iteration $\{\hat{x}_t^{[j-1]}\}_{t\in[1,T]}$:

$$v[j] = \big(- \frac{\partial}{\partial \hat{x}_t^{[j-1]}} L(x_t, \hat{x}_t^{[j-1]}\big)_{t\in[1,T]} \tag{2.73}$$

   For mean regression, that is, with the l_2 loss function, the negative gradients are given by:

$$v[j] = \big(- 2(x_t - \hat{x}_t^{[j-1]})\big)_{t\in[1,T]} \tag{2.74}$$

   (b) Fit each of the M members specified in step 2 using the negative gradient vector $v[j]$ as the response with the corresponding input variable.

   (c) Select the best-performing member, i.e., the model that minimizes the residual sum of squares, and let $\hat{b}_t^{[j]}, \forall t \in [1, T]$ be the fitted values of the best-performing member.

   (d) Update the current function estimate by adding the fitted values of the best-performing base model to the function estimate of the previous iteration $j - 1$:

$$\hat{x}_t^{[j]} = \hat{x}_t^{[j-1]} + \nu \hat{b}_t^{[j]} \tag{2.75}$$

where $0 < \nu \leq 1$ is a shrinkage factor.

(e) Stop if j has reached the maximum number of iterations J, or go to step 3.

Following the above-detailed steps, the final function estimate $\hat{f}$ can be presented as follows:

$$\hat{f}(z_t) = \hat{x}_t^{[0]} + \sum_{j=1}^{J} \nu \hat{b}_t^{[j]}, \forall t \in [1, T] \tag{2.76}$$

and since each component $\hat{u}^{[j]}$ depends only on one variable $k, k \in [1, p]$, the final estimate can be written as an additive model $\bar{\hat{f}}$:

$$\bar{\hat{f}}(z_t) = \bar{\hat{f}}^{[0]}(z_t) + \sum_{k=1}^{p} \underbrace{\sum_{j:kselected} \nu \hat{b}_t^{[j]}}_{\hat{a}_t^k} \tag{2.77}$$

$$= \bar{\hat{f}}^{[0]}(z_t) + \sum_{k=1}^{p} \hat{a}_t^k \tag{2.78}$$

where $\hat{a}_t^k$ is the relative contribution of the variable k to the final estimate, and p is the number of initial input variables (i.e. the dimensionality of $z_t = X_{t-p:t-1}$).

2.7 Explainable Machine Learning for Time Series

With the increasing use of complex forecasting models such as DNNs and their ensembles, a need for comprehending their behavior has increased over the recent years, particularly in high-risk and safety-critical forecasting application domains, e.g., intensive medical care applications, autonomous driving, extreme weather conditions forecasting, to name but a few [192]–[194]. These complex models are also known in the ML literature as black-box models. A model is said to be a black box if it is difficult for humans to understand why certain decisions have been made by this model [40]. Some other definitions go beyond and suggest that a model is a black box if it is not easy to comprehend how the input data is processed and transformed in the modeling process to reach the delivered output [195].

Since most of the time series forecasting models are involved in a decision-making process or a recommendation system and sometimes in safety-critical applications, building a certain level of trust in their output should be established. One of the crucial requirements to build this trust is to be able to understand these models. In this context, several Explainable Machine Learning (XML) techniques have been developed to facilitate human understanding of black-box ML models and frameworks. Explainability has been defined across different studies [40], [196], [197]. For instance, one definition by Miller et al. (2019) [196] states that explainability "is the degree to which a human can understand the cause of a decision." Another definition given by Keem et al.(2016) [197] is "Explainability is the degree to which a human can consistently predict the model's result." However, there is an agreement between all these works that the higher the explainability of an ML model is, the easier it is for a human to understand why certain decisions or predictions have been made. In this sense, a model is considered to be better explainable than another if its predictions or decisions are easier for a human to comprehend than the predictions from the other model.

Several XML techniques, as well as their taxonomies, have been presented in the ML literature [40], [198]. These techniques can be used to explain a single ML model, ensembles of ML models, as well as an ML-based framework, e.g., model selection framework. XML can be grouped into different families according to various criteria.

The first criterion is defined by when the explainability results should be delivered. Pre-model methods are usually independent of a specific model or architecture and delivered before model training. Common examples of these methods are the Principal Component Analysis (PCA) and t-distributed stochastic neighbor embedding [199]. It is also possible to integrate the explainability tool into the ML model. This is known as the in-model approach, such as association rule mining [200]. Different approaches produce explanations after learning the model; therefore, these approaches are called post-model. These approaches can produce important penetrations concerning what a model has learned after training, for example, SHapley Additive exPlanations (SHAP) values [201].

The second criterion denotes whether the explainability technique is local or global. For instance, local explainable methods are dedicated to explaining a specific prediction or result of the model. They can be acquired by representing mechanisms that determine the reason for individual predictions. Opposingly, global approaches focus on the whole ML model or framework and use the model's complete knowledge, learning, and input data. These approaches seek to define the nature, behavior, and performance of the model in common. Feature importance analysis is a very common example of these approaches that studies the features or characteristics that are generally responsible or liable for the model's better and more reliable performance amongst all present features in the data [202].

The third criterion concerns the format of the generated explanations. Some explainability methods rely on visual explanations in an image or figure format, e.g., heat-maps or box-plots [202], [203] while some others use another model to explain the back-box model. This operation is referred to as surrogate modeling. The surrogate model is an ML model that is considered to be explainable per construction and trained to approximate the predictions of the black-box model [40], [204].

Another criterion involves the ML model type. In this context, we distinguish between model-agnostic and model-specific approaches. Model-agnostic approaches are appropriate for every family of ML models and are not restricted to a fixed model type or architecture. These approaches do not generally deliver direct access to the main model properties or parameters, e.g., Partial Dependency plots (PDP) [40], Local Interpretable Model-agnostic Explanations (LIME) [195]. Model-specific approaches depend upon specific families of ML models, e.g., tree-based models such as decision trees or random forests, and neural networks family of models like DNNs. Some models, like DNNs, denote some complex structure that requires precise knowledge of the models. Therefore, several methods have been developed to explain them [203], [205], [206]. These methods are used to make DNNs' decision process more transparent by providing some understandable representations of their latent space [207] or by extracting salient parts in their input data like saliency maps for highlighting important regions in image data [203], [208]. It should be noted that it is possible to use model-agnostic methods for explainability, such as LIME, to explain individual predictions of DNNs [195]. However, two main reasons why it is necessary to consider explainability methods specifically developed for DNNs. First, neural networks learn features and concepts in their hidden layers. Therefore, specific tools are required to reveal them. Second, the gradient can be exploited to implement explainability methods that are much more efficient than model-agnostic methods since they look into the inside dynamics of the DNN. A wide variety of this type of explainability methods has been presented in the literature [40], [203], [208]–[210]. These methods can be grouped into three main families.

The first family includes the so-called "conceptual" explanations. A concept can be defined as any abstraction, e.g., a color in an image, an object, a data property, or even an idea. Given any user-defined concept, although a neural network might not be explicitly trained with the given concept, the concept-based approaches are devised to determine whether this concept is embedded within the latent space learned by the DNN or not [211], [212].

The second family of methods is model-based approaches that use model distillation to explain a neural network with a simpler model [213], [214]. For example, in [213], the authors express the knowledge acquired by the neural net in a decision tree that generalizes better than the one learned directly from the raw training data. Since it relies on hierarchical decisions, explaining a particular decision is made much easier.

The third family of methods consists of visualization-based approaches where either new learned features by the DNN [215] or the most important input data regions for the DNN's output (i.e., decision) [203], [216] are visualized. In the former, feature visualization is used to represent the learned features in an explicit way. Feature visualization for a "unit" of a neural network (i.e., a "unit" refers either to individual neurons, feature maps (channels), entire layers, or the corresponding pre-softmax neuron in the case of classification) is accomplished by finding the input of the unit that maximizes the corresponding activation. In the latter, the so-called heat or saliency maps are used to establish a relationship between the output and the input of a DNN given fixed weights. These maps are widely used in the context of computer vision with CNNs to create class-specific heat maps based on a particular input image, and a chosen class of interest [203], [208]. These maps are used for visualizing regions in the input image that are the most important for a particular prediction/decision of the model. They are computed using the gradient of the network's prediction with respect to the input, holding the weights fixed. This determines which input regions (e.g., which pixels in the case of an image input) need to be changed the least to affect the prediction the most.

Heat maps can be successfully transferred to the context of time series [217]. These maps are then used to understand which time intervals are mostly responsible for a given prediction in the case of univariate time series. For MTS, the maps are exploited for explaining during which time intervals the joint contribution of all the single time series is most important for that prediction. However, only a few works tackled DNNs explainability using heating maps for time series [217], [218]. In [217], an explainable CNN architecture is used to classify MTS data and explain the predictions. The CNN architecture consists of two stages with particular kernel sizes, which allows applying gradient-based techniques for generating heat maps for both temporal and spatial dimensions. Kusters et al. [212] have introduced "conceptual" explanations for DNNs by describing the relation between both global and local input properties of time series, e.g., time series components, stationarity, presence of outliers, etc., and the network's accuracy. Opposingly to heat-mapping methods that focus only on locally relevant input parts triggering the network prediction, the authors suggest also focusing on a global view of the time series to cover global causes behind a particular behavior of the DNN by evaluating the effect of abstract (local or global) input properties. The proposed method is model-agnostic and enables the utilization of domain knowledge.

XML has mainly focused on static learning tasks so far and on explaining learning models themselves. More recently, the focus has been shifted toward explaining online learning processes and other steps within these processes, such as model and parameter selection or model change and adaptation, instead of exclusively concentrating on the model's outcome. For instance, the explainability of online adaptive model change has recently been tackled in [219], [220]. First, model change is quantified using an approximation of the expected discrepancy. Then, significant changes are detected using ADWIN (ADaptive WINdowing) approach [89]. An explanation of the change is created and presented to the user using thresholding on the expected discrepancy and Permutation Feature Importance (PFI) which evaluates how the prediction error increases when a feature is not available. A method for automatically determining regions in the input data space that are affected by a given model adaptation and thus should be explained is presented in [221], [222]. Explaining model adaptation is performed using contrastive explanations (i.e. explaining why an event occurred in contrast to another). The above-mentioned works tackle the classification task. In this thesis, we also consider XML in the context of online learning in dynamic environments, particularly for online model selection but for the task of time series forecasting, where models are continuously adapted and changed over time.

2.8 Quality Predictive Analytics

Industry 4.0 is the new direction of automation and digital data transfer in manufacturing, including Internet of Things (IoT) settings, cyber-physical systems, cloud computing, systems integration, and big-data analytics that help in establishing smart industries and factories. Industry 4.0 encloses large-scale machine-to-machine communication, the integration of AI-based technologies, and the use of ML models in industries [223].

2.8.1 Model-based Quality Prediction

In manufacturing systems, quality deviations only detected at the end of the production chain may result in high amounts of rejected products that require laborious and costly rework or need to be scrapped [224]. To prevent such events, an early quality prediction has to be achieved. Hence, corrective actions are expected to have the largest impact if they are executed as early as possible in the process, avoiding thus costly rework and waste of resources through further processing of defective components [45], [46]. A key requirement for early quality prediction is the full coverage of the product quality in the early stages of the manufacturing process or within the execution of some stages, especially in those where quality testing of the product itself is not possible.

Nowadays, in the context of industry 4.0, the linkage of the production environment through Information and Communication Technologies (ICT) to cyber-physical systems with the goal of monitoring, controlling, and optimizing complex manufacturing systems, enables real-time capable approaches for process data acquisition, analysis, and knowledge discovery [8], [18], [224]. This is achieved in practice by collecting and analyzing sensor data. As a result, data-driven approaches for predicting process quality in real-time and deriving adequate process control interventions in a timely manner can be developed [8], [45]. Sensors are able to generate mass quantities of sensor data as responses to some types of inputs from the physical production environment. Most often, each of these data points is captured at specific time stamps, effectively transforming sensor data into time series data that can be analyzed across this additional dimension [46].

ML algorithms trained on sensor data are used to gain knowledge that can be generalized and used to predict unknown future events, i.e. new data points. A quality prediction that is performed using ML is referred to in Industry 4.0 as model-based quality prediction [224]. To do so, the description of the product or process quality and all the related information should be done at the first step, especially in highly complex dynamic production systems with non-linear interactions between their steps. The second step consists of building and training a predictive model that maps the available quality-related information, e.g., operating states sensor data, process input parameters, to the resulting process/product quality label [45], [46], [225]. This model can be used afterward for predicting the expected quality given a set of input values of non-tested new products. Several industrial applications utilize already existing ML methods and algorithms to solve actual problems in manufacturing from an engineering point of view [224]. These applications cover a wide range of industrial fields, including electronics [224], metallurgical [45], [46], [225], and machining industries [47]. Likewise, the adopted ML solutions are not limited to a specific family of models but include, amongst others, Artificial Neural Networks (ANNs) [47], Support Vector Machines (SVMs) [46], and Decision Trees (DTs) [45]. Most of the aforementioned works focused on quality prediction at the end of the production chain with the goal of reducing the costs of quality inspection assigned by humans or special machines [45], [46], [224]. However, only a few studies have achieved and investigated the impact of early quality prediction in the very early stages of multi-staged processes or within their execution [45].

Quality can be described in the form of continuous real-valued measurements of some process-related physical quantities over time. In this context, time series forecasting methods can be used to predict the correct quality of a process or a product [145]. Industrial processes are subject to changes that are inherently due to the dynamic nature of the process itself [47] or to some external factors in the industry, e.g., change in the environment temperature or some incidents [46]. Therefore, online model adaptation is most often required. This covers the online adaptive model selection and the online management of many models.

2.8.2 Learning from Process Simulation

The performance of machine learning depends to a great extent on the quality and quantity of data available for training [226], [227]. Large data sets are most often required for training [228]. However, data collection from sensors in some industrial settings can turn out to be very challenging due to the high costs related to data collection or storage or to some difficulties in accessing the data due to some privacy issues [47], [229]. Therefore, the fusion of data sets from multiple sources can be helpful but also very challenging [228]. More recently, process simulation has appeared to be a very attractive solution for generating synthetic data for learning purposes. Simulations are most often considered the ground truth because they are based on theoretical knowledge of the specific application domain. Hence, they are used for testing learning models or annotating the data. Simulations are used as experimental test beds that offer scientists the chance to study a range of phenomena in a structured way. This is a standard procedure in computer science (e.g., [230]) as well as in engineering [47] and physics [231], to name but a few. Many investigations of multi-sensor fusion have used simulation environments only for testing their frameworks [232]. Process simulations are also established instruments to investigate processes in a virtual environment prior to deployment [47]. More recently, simulations have been conceived as powerful data generators, providing thus promising opportunities for simulation data mining [233]. However, simulations have many limitations in practice. They cannot provide a completely accurate representation of reality in real-time [234] and usually have only a limited prediction accuracy when modeling complex phenomena [235]. In contrast, ML models can be applied in real-time and offer the opportunity to predict events based on the analysis of a set of explanatory variables. Therefore, new trends in applied ML aim at replacing simulation models with surrogate ML models that have been trained on simulation data [236]. Some recent works focused on learning from simulation data to monitor the real-world process and predict upcoming unknown events with a reasonable accuracy [237].

2.9 Final Remarks

The time-evolving nature of time series and its complex structure that may involve non-stationary processes make time series forecasting one of the most challenging research topics in the ML literature of knowledge discovery from databases. Several ML-based frameworks have been developed to learn from this data by proposing some modeling of how past historical data is linked to future observations. These models are used for predicting the future behavior of the time series. However, forecasting is still considered to be challenging, and further ML-based solutions development is required.

In this chapter, we presented an overview of the ML literature related to the topics evoked in this thesis. More precisely, we presented the main time series characteristics. Then, we revised previous work, with a particular emphasis on the topics of concept drift detection and adaptation, model selection, ensemble learning, and explainability. From the applied ML perspective, we covered the topic of ML model-based quality prediction of industrial processes in which online adaptive model learning is most often required in addition to data enrichment from heterogeneous sources, more precisely, simulations. In the following chapters, we explore these topics in more detail. Indeed, we suggest facing the main limitations of the current State-of-the-Art by developing novel methods for online time series forecasting that are adaptive to changes in this data in an explainable manner. The ultimate goal behind our work is to support organizations and industries exploiting historical data to make correct explainable data-driven decisions about the future.

Part II

Forecasting

3 Online Adaptive Time Series Variables Selection

The main goal of this part of the dissertation is to develop novel ML methods for tackling the time series forecasting task. To accomplish this, we previously highlighted the need for building online adaptive forecasting models. However, before the model-building stage, we need to select time series input variables to be fed to the model in the case of the availability of MTS data. This selection has to be made online and adaptively to the changes in the MTS data and in the time-dependent dependencies among its composing time series variables. The performance of the forecasting model resulting from a given selection of time series input variables can also be tracked to inform about the adequacy of the selection and whether some model updates have to be performed. In this chapter, we then address the task of time series variables selection and online model adaptation for MTS forecasting.

3.1 Introduction

On the one hand, MTS data represents an enriched form of information about the application. On the other hand, the number of its composing variables can increase drastically and might include irrelevant and redundant ones. This may heighten the curse of dimensionality. Therefore, it is necessary to select the most important time series variables carefully. The evolution of MTS is spatio-temporal, along with the different variables and over time, respectively. However, this spatio-temporal data may involve multiple non-stationary processes and the dependencies among its composing variables may also follow a non-stationary process. As a result, the relationship between some time series variables and a target one might change significantly over time and be subject to concept drifts [89]. Hence, previously learned concepts about data become no longer valid, making the offline input variable selection procedures inappropriate for making future predictions. Therefore, the selection of time series variables should cope with the evolving nature of the spatio-temporal dependencies in the MTS data. In addition to adaptive time-dependent spatial variables selection, adequate model selection and adaptation are required to cope with the time-evolving characteristics of MTS data [170], [238]. Most of the existing MTS forecasting models operate in a static manner, i.e., the model is trained offline using some collection of historical data and a fixed selection of input variables. Its parameters are optimized once at training time. At test time, the model is deployed with fixed learned parameters and fixed information about temporal and spatial data [17]. Even though optimizing the input of learning models through feature extraction and selection has proven to be very useful in improving the accuracy of a wide variety of time series learning tasks, including classification [49], [239], [240], clustering [241], [242] and forecasting [243], [244], few works tackled input variable selection for online MTS forecasting [16], [245], [246].

Methods for online MTS forecasting focus either on a very specific application/setting [247] or use a very specific family of ML models, such as Neural Networks [248], [249]. More recently, a drift-aware Vector Autoregressive (VAR) model has been proposed in [16]. In contrast to the classic VAR model, which takes as input, all the variables of the MTS [17], an adaptive selection procedure of a subset of input variables is done in the drift-aware VAR. The update of this subset depends on a change in the Pearson-Correlation (PC) [73] measured between two variables over a time-sliding window, which is assumed to occur due to the presence of concept drift. Even though the proposed method is online and adaptive, it focuses only on a particular model, namely the VAR. In addition, the time series variables selection is made by ranking them according to their relevance to the target time series using PC. No further analysis is carried out to investigate the redundancy in the selected subset. The relevance/similarity to the target is measured using the PC coefficient. However, it has been proven that there is no single universal measure for the similarity between two time series either for relevance or redundancy analysis [73]. The quality of the achieved results in this context depends to a large extent on the used time series measure [73].

In this context, we propose an online adaptive framework for MTS forecasting which performs both input time series variables and adequate forecasting model selection. Model selection is performed since the success of the MTS forecasting task is model-dependent. In addition, changes in the input variables selection trigger an update of the ML model. Input variables selection is made following a two-staged procedure based on relevance and redundancy analysis. The selection is made dynamically and adaptively in an informed manner following concept drift detection. The concept drift detection covers the two MTS dimensions, namely spatial and temporal. Spatial dependencies indicate the similarity between the input variables at one time instant. We monitor the change in the similarity values over time. Temporal dependencies indicate the patterns discovered within the same spatial dimension over time. The drift detection within the temporal dimension is ensured by tracking the change in the estimated model's performance on the target time series variable over time. In addition, the choice of adequate relevance and redundancy measures, as well as the forecasting model, is done in an automated fashion using meta-learning on well-devised MTS meta-features. Our framework is denoted in the rest of the thesis, OAMTS: Online Adaptive Multivariate Time series forecasting. We further conduct a comprehensive empirical evaluation to validate our method using 66 real-world MTS datasets from different domains. We have created separate meta-data which cover a collection of real-world and synthetic MTS with various characteristics for the meta-learning task. The obtained results show that our method achieves excellent results in comparison to the SoA approaches for MTS forecasting. We note that all the experiments are fully reproducible and that both the code and datasets are publicly available [1].

The main contributions of this part of the thesis can be summarized as follows:

- We present a novel method for online drift-aware input time series variables selection using relevance and redundancy analysis.
- The drift detection mechanism is devised to operate on both spatial and temporal dimensions.
- We fully automate the choice of relevance and redundancy measures for MTS, as well as the forecasting model selection using meta-learning.
- We provide a comparative empirical study with SoA methods and discuss its implications in terms of predictive performance and scalability.

3.2 Related Works

In contrast to univariate time series forecasting, i.e., the forecast of a single time series, where several methods for online adaptive single model selection [23] or ensemble learning [9], [10] have been proposed, most of existing methods for MTS forecasting are devised to operate in a static manner [17], [247], [248]. In other words, the models in these methods are learned offline using a collection of historical MTS data, their parameters are optimized using these datasets and stored to be used at test time to make the predictions. In addition, most of these methods are either application-specific [247] or model specific, i.e., use an arbitrarily selected machine learning model family [17]. The most widely used models are VAR [16], [17] or DNNs [248], [249]. In [247], the forecasting of an MTS of energy consumption in smart buildings is transformed into a standard regression task using time series embedding, and then, different types of feature selection methods for regression tasks are applied. The features are extracted offline once and kept static at test time. More recently, some works have exploited the success of some DNNs architectures in computer vision-related applications and successfully transferred and adapted them to MTS forecasting by treating temporal and spatial dimensions in MTS as the 2d dimensions in images. Some of the other works focused on introducing some improvements or adaptations over existing DNNs to cope with the characteristics of MTS [248], [249]. In [249], the authors argued that random weights initialization in Recurrent Neural Networks (RNNs) disallows the neurons from learning the latent features of the correlated variables of the MTS. Therefore, they suggest using a pre-trained LSTM combined with a stacked auto-encoder to replace the random weight initialization strategy adopted in deep RNNs. In [248], Graph Neural Networks (GNNs) are adapted to MTS forecasting by adding a mix-hop propagation layer and a dilated inception layer to capture the spatial and temporal dependencies within the MTS. This is done to make GNNs capable of handling relational dependencies that are not known in advance like in the case of MTS. Even though dependencies between the variables of the MTS may change significantly over time, most of the aforementioned works do not consider a time-dependent selection of the input time series variables for the MTS forecasting

[1] https://www.dropbox.com/sh/z2g0us0nti3nqzg/AAAJ6_6JcGZHN_y10q8XDYa_a?dl=0

model. The choice of the model is most often arbitrary or transferred from another domain like computer vision or regression. In addition, once the model is chosen, its corresponding parameters are kept fixed. It is important to note that it exists some methods for model adaptation to data changes and model performance, more particularly to concept drift in the context of streaming data classification [98], [99] and univariate time series forecasting [16]. These methods can be grouped into two main families, namely blind adaptation and informed adaptation. As we mentioned in Section 2.2.2, in blind adaptation, the model is retrained either at each time instant with each upcoming observation or over a fixed period in time without any consideration of possible data or model performance changes. However, this family of methods is known to be time-intensive, resource-consuming, and unpractical for online forecasting [16]. Informed adaptation methods use some statistical information about the data or model performance to inform the model about the occurrence of concept drift and, if necessary, trigger input data update using adaptive time-windowing approaches and input re-selection [98], [99] and subsequently, model retraining [16] or a new model selection [16].

Our method is based on an informed adaptation for MTS forecasting. This is done by monitoring the changes in spatio-temporal dependencies in the MTS and the model performance over time. Since the results achieved in various time series tasks such as clustering and classification depend to a large extent on the used measures for evaluating time series dependencies/similarities [73], we suggest also automating the choice of the adequate dependencies measures as well as the selection of the adequate model for a particular application by means of meta-learning.

It is worth mentioning that many SoA approaches for feature selection by combining relevance and redundancy analysis of the features using a forward sequential search that repeatedly adds the feature which has the best ratio between relevance and redundancy to the already selected features [250]–[252]. These methods have quadratic computational complexity. Adding constraints on the number of features to be considered or selected using thresholding techniques helps in reducing the overall run-time [252], [253], but the thresholding parameter needs to be tuned, which increases the overall run-time again. Opposingly, in our framework, we use backward search where we remove in one run $N - top - N$ features using similarity to the target variable ranking. The remaining $top - N$ are reduced further by means of clustering, which has been widely proven to be effective in grouping similar high dimensional continuous features [253], [254]. Many of the mentioned works investigated the aspect of feature selection stability, which refers to the robustness of the selected features, with respect to data sampling and to its stochastic nature [251], [255], [256]. For instance, in [256], a fast and efficient method based on an ensemble technique such that the stability of feature selection comes with little or even no extra run-time is presented. However, the method is based on a cross-validation technique which should be revisited for time series data [257]. Since we base our analysis on the assumption that the relevance and/or redundancies of the variables would change in the course of time, we did not focus on the stability aspect of MTS variable selection.

3.3 Drift-aware Input Time Series Variables Selection

In this section, we present our input variables selection procedure and its main stages. Figure 3.1 shows a visualization of an MTS $\mathbf{X}$ collected till time t. The choice of the target time series variable is application-dependent (i.e., a variable of interest for the user). It is worth mentioning that traditional MTS methods like VAR take as input all the variables and output simultaneously N-dimensional vector of forecast values for the N time series variables of the MTS. Our goal is to improve the accuracy of the MTS forecasting task through adaptive timely time series variables selection. Therefore, if all the variables are evaluated of the same importance and need to be predicted. Our selection procedure can then be computed for all the MTS variables. For simplicity, we assume in the following that we have only one time series variable of interest, i.e., the target variable. However, the reasoning applied to one target variable can be generalized to all the remaining variables.

$$\mathbf{X}_{1:t} = \begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_{t-1}^1 & x_t^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^r & \cdots & \cdots & x_{t-1}^r & x_t^r \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^N & \cdots & \cdots & x_{t-1}^N & x_t^N \end{pmatrix} = \begin{pmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_{t_i} \\ \vdots \\ \mathbf{X}_t \end{pmatrix}^{\top}$$

Input time series variable, Target variable

Figure 3.1: An MTS $\mathbf{X}$ collected till time t

For a given MTS data, the input time series variables to be used for forecasting are determined in a timely manner by computing their *relevance* to the target time series variable. Once the most relevant variables are identified, *redundancy* analysis through time series clustering is carried out to remove redundant variables. The choice of adequate time series measures for *relevance* and *redundancy* is determined beforehand by the meta-learning component that decides as well which model to be used for a particular MTS data. Both input time series variables and model updates are triggered once a concept drift in the spatio-temporal dependencies among these variables or/and model performance is detected. Basically, new variables are selected, and time windows are adjusted to update the time series variables with recent observations.

3.3.1 Preliminaries

3.3.1.1 Notations

Given the MTS $\mathbf{X} = \{X^1, X^2, \cdots, X^N\}$, and the target time series variable X^r with r some arbitrary index in $[1, N]$, the goal of online input variables selection is to determine which time series variables $X^i, i \in [1, N]\backslash\{r\}$ should be fed into the forecasting model at time t to forecast the next value at time $t+1$. We denote by $X^i_{t_s:t_e}$ the subsequence of X^i starting at time instant t_s and ending at time instant t_e. We divide the MTS $\mathbf{X}$ into $\mathbf{X}^{train}_{\omega} = \{X^1_{1:t-\omega}, X^2_{1:t-\omega}, \cdots, X^N_{1:t-\omega}\}$ and $\mathbf{X}^{val}_{\omega} = \{X^1_{t-\omega+1:t}, X^2_{t-\omega+1:t}, \cdots, X^N_{t-\omega+1:t}\}$, with ω a provided window size. $\mathbf{X}^{train}_{\omega}$ is used for training the forecasting model, and $\mathbf{X}^{val}_{\omega}$ is used to compute the *relevance* and *redundancy* measures, since both input and target time series variables are required to be known.

3.3.1.2 Forecasting Model Learning

Standard approaches for addressing MTS forecasting tasks include traditional techniques for MTS analysis, such as the popular Vector Autoregressive VAR family of methods [17], or ARIMAX [54] which is the extension of Autoregressive Integrated Moving Average model (ARIMA) (See Section 2.4.4.1) to MTS where some input time series variables are provided as exogenous variables to forecast the dependent variable, i.e., the target variable. These models take as an input multiple time series sequences $\mathbf{X}_{1:t}$. In addition, regression models can be employed in the context of MTS forecasting by using a time-delayed embedding (See Section 2.4.4). To optimize further the MTS forecasting task, we aim to select the forecasting model given the characteristics of the MTS data in question. This is done by the meta-learning components in Section 3.4. Therefore, we consider a pool of candidate forecasting models $\mathbb{P}$ which is designed to contain a set of various and *heterogeneous* models, such as VAR [17], Gaussian processes [258], support vector regression [259], and DNNs [15]. The candidate models are trained on $\mathbf{X}_{\omega}^{train}$ using the same number p of lagged values for each variable in the MTS as input to model the following value in the time series.

3.3.2 Adaptive Input Time Series Variables Selection

Given a target time series $X^r \in \mathbf{X}$, in order to forecast its value at a future time instant $t+h, h \geq 1$ (for simplicity of notation, we assume $h=1$), the selection of the time series variables $X^i, i \in [1,N]\backslash\{r\}$ whose p-lagged values will be used as input for the forecasting model in addition to the p-lagged values of the target time series X^r, has to be determined in a timely-manner at t. The selection is decided by measuring how much each of $X^i, \forall i \in [1,N]\backslash\{r\}$ is relevant to X^r and whether X^i is redundant in the presence of the other variables.

3.3.2.1 Relevance

The relevance of each $X^i, \forall i \in [1,N]\backslash\{r\}$ to X^r is measured by computing the similarity between them on the time window $T_{\omega}^{val} = [t-\omega+1, t]$, denoted by:

$$s_t^{i,r} = sim(X_{t-\omega+1:t}^i, X_{t-\omega+1:t}^r). \tag{3.1}$$

The time series variables $X^i, \forall i \in [1,N]\backslash\{r\}$ are sorted according to their $s_t^{i,r}$ and the top-n most similar variables to X^r are selected. There is no single universal similarity measure between time series that is valid for every application. The choice of an adequate similarity measure is made by considering the characteristics of the MTS in question.

3.3.2.2 Redundancy

The top-n selected input time series variables may include some redundant variables that would lead to increasing the dimensionality of the MTS forecasting task without contributing to the model's accuracy. Relying on the computed similarity measures is not sufficient since they are measured over a time window of observations T_{ω}^{val}. For instance, two candidate variables can have the same level of similarity to the target variables while being effectively similar to it on two distinct time intervals included within T_{ω}^{val}. Therefore, we suggest removing redundancies by clustering the top-n variables and selecting only one-time series representative per cluster. To compute clusters for time series, several techniques are proposed in the literature, which can be classified based on the way they treat the data and how the underlying grouping is performed [260]. One classification depends on whether the whole series, a subsequence of it, or individual time points are to be clustered. In our case, we cluster the subsequences $\{X_{t-\omega+1:t}^{1}, X_{t-\omega+1:t}^{2}, \cdots, X_{t-\omega+1:t}^{N}\}$. On the other hand, the clustering itself may be shape-based, feature-based, or model-based. The choice of time-series representation and the clustering algorithm has a big impact on performance with respect to the clustering quality and execution time [84], [93], [261]. Again, no single clustering method is universally valid, and the success of the method depends on the characteristics of the time series data [260]. Denote with the $c_t^{i,j}$ the clustering measure used for computing the distance between the two sequences $X_{t-\omega+1:t}^{i}$ and $X_{t-\omega+1:t}^{j}$, with $i, j \in TOP_n$ and TOP_n denotes the subset of selected input time series variables,i.e., $|TOP_n| = top_n$. The choice of the clustering algorithm, together with the corresponding distance measure, is decided by the meta-learning component. Further details are provided in Section 3.4.

3.3.2.3 Drift-aware Variables Selection Adaptation

Both relevance and redundancies are monitored continuously over time. For relevance, with each upcoming data observation at $t+h, h \geq 1$, we slide T_{ω}^{val} by one step, i.e. to include the observation at $t+h$, and we measure $s_{t+h}^{i,r}, \forall i \in [1, N] \backslash \{r\}$. Then, we compute:

$$s_{t+h}^{min} = \min_{i \in [1,N] \backslash \{r\}} s_{t+h}^{i,r} \tag{3.2}$$

in order to determine the distance between the target sequence and the most dissimilar sequence within the $N-1$ input variables. Then, we compare it to the initial calculated distance $s_{t_i}^{min}$. In our case, $t_i = t$ indicates the start of the online forecasting stage. The distance is treated as time series where s_{t+h}^{min} is its value at time $t+h$.

Definition 11 (Weak Stationary Similarity) *The similarity structure between a set of input time series variables and a target time series is said to be weakly stationary if the true mean of* Δ^s *is 0, with:*

$$\Delta_{t+h}^{s} = |s_{t+h}^{min} - s_{t_i}^{min}| \tag{3.3}$$

Following this definition, we can assume that the distance between the target time series sequence and the most dissimilar input sequence sets its boundary under a form of a logical *diameter*. If this boundary diverges in a significant way over time, a drift is assumed to take place. We propose to detect the validity of such an assumption using the well-known Hoeffding Bound [262], which states that after ω independent observations of a real-valued random variable with range R, its true mean has not diverged if the sample mean is contained within $\pm\zeta$ and ζ is given by:

$$\zeta = \sqrt{\frac{R^2 \ln(1/\mu)}{2\omega}} \tag{3.4}$$

with a probability of $1 - \mu$ (μ is a user-defined hyperparameter).

Once the condition of the *weak stationary similarity* presented in Definition 11 is violated at t_{d_s}, a drift is assumed to take place at t_{d_s}. A relevance re-computation is then triggered. A re-clustering is also performed, the selection of the variables is updated and the reference diameter $s_{t_i}^{min}$ is reset by setting $t_i = t_{d_s}$. This drift type is denoted `Drift Type I`.

Similarly, for the redundancy, we continuously monitor the distance measure used for clustering $c_{t+h}^{i,j}, \forall X^i, X^j \in TOP_n$, which results in the similarity matrix given by:

$$\mathcal{C}_{t+h} = (c_{t+h}^{i,j})_{1 \leq i,j \geq top_n} \in \mathbb{R}^{top_n \times top_n} \tag{3.5}$$

Then, we place all the elements of $\mathcal{C}_{t+h}$ in a vector ς_{t+h}, where $\varsigma_{j,t+h} \geq \varsigma_{j-1,t+h}, \forall j \in \{1, \cdots, top_n^2\}$. Let ς_{t_i} denote the value of ς at the initial instant $t_i = t$ of the generation of $\mathcal{C}$. We monitor the deviation Δ_{t+h}^{ς} similarly to Δ_{t+h}^{s}:

$$\Delta_{t+h}^{\varsigma} = |\varsigma_{1,t+h} - \varsigma_{1,t_i}| \tag{3.6}$$

We test the occurrence of concept drift within the clusters following the same condition defined in Definition 11. If a concept drift is detected at t_{d_c}, both relevance and redundancies re-computation are triggered, and a re-selection of input variables is performed. We reset then $\varsigma_{t_i} = \varsigma_{t_{d_c}}$. This drift type is denoted `Drift Type II`.

3.3.3 Forecasting Model Adaptation

Drifts in the relationships among the time series variables, i.e., `Drift Type I` and/or `Drift Type II` trigger the update of their selection. This update has an impact on the model naturally since, with each new selection, a new model has to be trained. However, the change in the dependencies structure of time series variables with the target variable or among each other is not the only indicator of outdated or irrelevant input. The increase in the forecasting error may indicate a possible change in the relationship between the input variables and the target time series and/or outdated model parameters due to outdated time series observations that were used for training. Therefore, necessary measures such as input variables re-selection and model re-training with recently acquired data have to be taken. To do so, the forecasting error ϵ is estimated using the Root Mean Square Error (RMSE) and is monitored over the sliding window of the recent observations T_{ω}^{val}. The error can be viewed as a time series, and at $t + h$:

$$\epsilon_{t+h}^{\omega} = \frac{1}{\omega} \sum_{j=t+h-\omega}^{t+h-1} (x_j^r - \hat{x}_j^r)^2 \tag{3.7}$$

with $\hat{x}_j^r$ the predicted value of X^r at time j.

Naturally, with time-evolving data, the model's error changes over time and may follow non-stationary concepts. Let ϵ_{t_i} denote ϵ value at the initial instant of its generation $t_i = t$. Since the forecasting error is directional, drift detection using the absolute value of the error deviation with the Hoeffding-bound can be misleading. Therefore, we suggest using the Page-Hinkley Test [89] to detect significant increases in the forecasting error. The pseudo-code of the Page-Hinkley Test is described in Algorithm 1.

Algorithm 1: Page-Hinkley Test for Error Drift Detection

```
Data: Error time series: $\epsilon_t$; Admissible Change $\nu$; Threshold $\varrho$
Result: Drift at time $t_{d_\epsilon}$: $alert_{t_{d_\epsilon}}$
/* Initialize CUSUM and the error estimator: $E(0) = 0$, $cusum(0) = 0$ ,$M_T = 1$;
$alert_0 = 0$ ;
for $i \in \{1...\infty\}$ do
    $E(i) \leftarrow E(i-1) + \epsilon_{t+i-1}$ ;
    $cusum(i) \leftarrow cusum(i-1) + \epsilon_{t+i-1} - \frac{E(i)}{i} - \nu$ ;
    $M_T \leftarrow min(M_T, cusum(i))$ ;
    if $(cusum(i) - M_t) \geqslant \varrho$ then
        $alert_i = 1$;
        $t_{d_\epsilon} = t + i$;
    else
end
```

where ν and ϱ are user-defined hyper-parameters, where ν is the tolerable change in the estimated error and ϱ is a threshold. A larger ϱ avoids detecting false drift alarms but can also lead to missing true drifts [89]. The error drift detection is denoted `Drift Type III`. An alert at time t_{d_ϵ} declares the occurrence of `Drift Type III` and triggers the update of the input variables through new selection, i.e., new relevance and redundancy re-computation, and updates the current model with new inputs and recent observations. It also restarts the Page-Hinkley Test from the beginning. The model also gets updated with the update of the input triggered by `Drift Type I` and/or `Drift Type II`.

3.4 Online Automated MTS Forecasting

As discussed above, there are no single universal similarity measures for relevance and redundancies. Similarly, for the forecasting model, adequate model selection has to be performed to cope with the characteristics of the MTS data in question. Once the model is selected, the online adaptation scheme in our framework (See Section 3.3.3) takes care of the adaptation of the model in an informed manner to the real-time changes in the data and the model's performance. To automate the choice of the measures and the model, we use meta-learning. Let $\mathbb{S}$ and $\mathbb{C}$ be the spaces of the relevance and redundancy measures, respectively. Denote with $\mathbb{M}$ the space of candidate models to solve the MTS forecasting task. Using a set of m MTS characteristics represented here by the so-called meta-features, the goal of the meta-task is to fit model $f_{meta} : \mathbb{R}^m \rightarrow \mathbb{S} \times \mathbb{C} \times \mathbb{M}$ to predict the best combination of relevance and redundancies measures and forecasting model choice given a vector of m MTS meta-features as input.

3.4.1 MTS Meta-Features

Several works have been proposed for extracting Univariate Time Series (UTS) meta-features [97]. Therefore, most of the existing works that tackled the same task for MTS use the same features developed for UTS by extracting meta-features from each time series variable in the MTS and concatenating them in one feature vector [247]. In this work, in addition to the transfer of the most often used meta-features in the context of univariate time series to the MTS domain, we propose to add MTS-specific meta-features. We additionally adapted the concept of *land-marking* developed for meta-tasks in classification, and regression [34] to MTS data. The extracted meta-features can be grouped into three main families.

3.4.1.1 UTS-specific Features

For each time series variable in the MTS, we extract different time series-specific features that can be grouped into three families, including descriptive statistics, frequency domain, and auto-correlation features [135]. The list of these features includes the trend, skewness of series, turning points, kurtosis of series, step changes, length of series, non-linearity measure, the standard deviation of de-trended series, maximal value, no. of peaks not lower than 60% of the max, Auto- and Partial Correlations at lags one and two, and seasonality. Since the number of variables in the MTS can be very big, we compute the mean and the standard deviation of each extracted feature over the different variables.

3.4.1.2 MTS-specific Features

We suggest investigating the relationships/dependencies among the MTS variables. To do so, we compute several similarity measures [73], including Pearson Correlation, Euclidean distance, Dynamic Time Warping distance, Mahalanobis distance, Amplitude and Phase differences of the Fourier Transform (FT), and Shape similarity based on derived FT amplitude and phase differences, between each pair of variables. These similarity computations result in similarity matrices for each measure. Instead of concatenating all the coefficients of all the matrices in one feature vector and increasing the meta-task input dimensionality, we suggest computing the diversity in similarity/dependence along with all the variables pairs for each similarity matrix. Denote with $\mathcal{S} \in \mathbb{R}^{N \times N}$ the resulting similarity matrix of a given similarity/distance s between all the N MTS variables. We define diversity as (note the similarity values are normalized between -1 and 1) :

$$div(\mathcal{S}) = 1 - \frac{1}{\sum_{1 \leq i \neq j, \leq N}} \sum_{1 \leq i \neq j, \leq N} s(X^i, X^j) \tag{3.8}$$

3.4.1.3 Landmarking-based Features

This type of meta-features is designed to describe the performance of some learning algorithms, called *landmarkers*, in various learning contexts on the same data. *Landmarkers* are ML models that are relatively computationally cheap either in training or testing compared to other models. So far, all the proposed *landmarkers* and corresponding meta-features have been proposed for classical meta-learning applications to classification problems, and one work has added an extension of this concept to regression [34], whereas we focus on *landmarkers* integration for MTS forecasting. In regression, the process starts by creating one landmarking model over the entire training set. A small artificial neighborhood for each training example is created using Gaussian noise. Then descriptive statistics of the models' output, mean, stdev., 1st/3rd quantile, are extracted. In our case, we use, LASSO, 1NN , MARS and CART , as *landmarkers* [34] and train them on $\mathbf{X}_{\omega}^{train}$. We can distinguish three types of Landmarking features:

- *Global landmarking*: We evaluate each model on $\mathbf{X}_{\omega}^{val}$, and we extract the descriptive statistics of the models' output.
- *Performance-based local landmarking*: we split $\mathbf{X}_{\omega}^{val}$ into equally-sized non-overlapping time windows of size n_{ω} . We evaluate each model on each time window, and we extract for each window the descriptive statistics of the models' output.
- *Model-based local landmarking*: This type of local landmarking is designed to characterize the *landmarkers* within a particular time series region, in our case, each time window of size n_{ω}. To do so, we extract the knowledge that the *landmarkers* have learned about each window. In addition to the prediction of each *landmarker* on each window, we compute the depth of the leaf which makes the prediction and the number of examples in that leaf, and the variance for each window for CART, the average over each window of the width and mass of the interval in which each time value falls for MARS, and the average over each window of the absolute distances to the nearest neighbor for 1NN.

The different stages of OAMTS are illustrated in Figure 4.1.

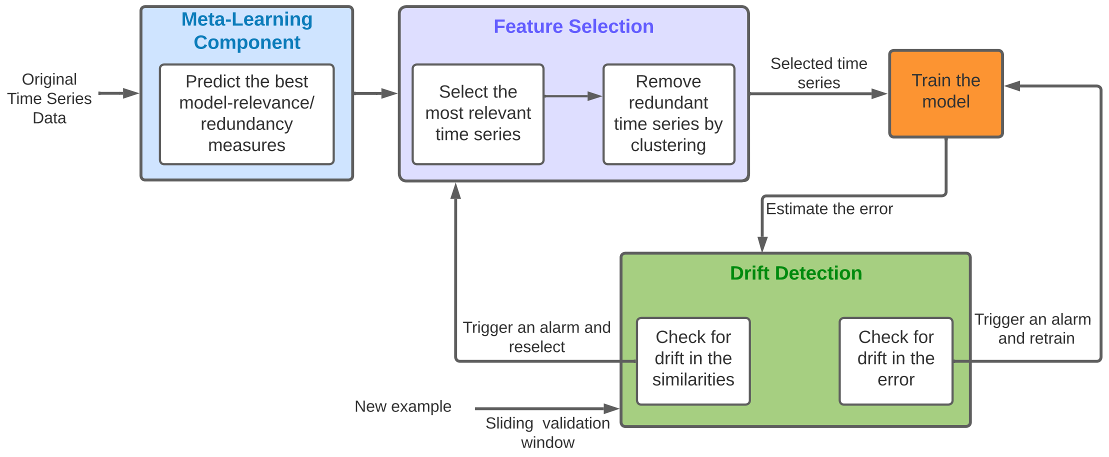

Original Time Series Data
Meta-Learning Component
Predict the best model-relevance/ redundancy measures
Feature Selection
Select the most relevant time series
Remove redundant time series by clustering
Selected time series
Train the model
Estimate the error
Drift Detection
Trigger an alarm and reselect
Check for drift in the similarities
Check for drift in the error
Trigger an alarm and retrain
New example
Sliding validation window

Figure 3.2: Schematic visualization of OAMTS.

3.5 Empirical Experiments

We present the experiments carried out to validate OAMTS and to answer these research questions:

- **Q1:** How does OAMTS perform compared to the SoA and existing online methods for MTS forecasting?
- **Q2:** What is the importance of each component, namely relevance, and redundancy, in the input time series variables selection on the performance?
- **Q3:** What is the benefit of each drift type detection for the performance of OAMTS?

- **Q4:** To which extent is it necessary to automate the choice of adequate relevance and redundancies measures, as well as the forecasting model choice?
- **Q5:** How scalable is OAMTS in terms of computational resources compared to the most competitive online model selection methods? and what is the computational advantage of the drift-aware adaptation of the framework?

3.5.1 Experimental Setup

The methods used in the experiments were evaluated using the Root Mean Squared Error (RMSE) (see Equation 2.57). We collected a total of 166 MTS from various real-world applications. 100 MTS are exclusively used for the meta-learning task, while the remaining 66 MTS are used for testing the meta-model, which recommends which relevance and redundancy measures and forecasting model from the pool of candidate models that we have devised to use. Following the recommendation of the meta-model, these 66 MTS are used to validate the online forecasting performance of OAMTS. Each of the 66 MTS was split using 50% for training (X_{ω}^{train}), and 25% for validation (X_{ω}^{val}) and 25% for testing. Note that in each MTS, we have chosen one variable as the target one depending on the application and the remaining variables as different input variables. However, for some applications like taxi demand forecasting, all the variables can play the role of the target one and change the role between variables. A full list of the used datasets, together with a description, is given in the Appendix Tables A.1 and A.2, and the code repository is given under this link [2].

3.5.1.1 Candidate Models Setup

We construct the pool $\mathbb{P}$ of candidate models. We mentioned earlier that there is no single method for MTS forecasting that outperforms all the other methods on every time series. Hence, we incorporate and test different families of models.

- Traditional MTS forecasting model: Vector Auto-Regressive (VAR) [17].
- Regression models are also included in $\mathbb{P}$ and are applied after using MTS embedding of dimension $N \times p$. These models include:
  - Gradient Boosting Machines (GBM) [263].
  - Support Vector Regression (SVR)[264].
  - Random Forest (RF)[31].
  - Projection Pursuit Regression (PPR) [265].
  - MARS (MARS) [266].
  - Partial Least Squares Regression (PLS) [267].
- Neural networks: that are designed for the time series forecasting task are introduced to $\mathbb{P}$ such as:
  - Multi-Layer Perceptron (MLP) [268].

[2] https://www.dropbox.com/sh/z2g0us0nti3nqzg/AAAJ6_6JcGZHN_y10q8XDYa_a?dl=0

  - Bidirectional-LSTM (bi-LSTM) [110].
  - Convolutional Neural Network LSTM (CNN-LSTM) [269].
  - Convolutional-LSTM (Conv-LSTM) [269].

Using different parameter settings for each family of models, we generated a pool of 20 candidate models.

3.5.1.2 Meta-learning Task Setup

The list of similarity measures considered to measure the relevance of the variables includes:

- Pearson Correlation.
- Spearman Correlation.
- Euclidean Distance.
- Dynamic Time Warping Distance.
- Manhattan Distance.
- Fourier-based Distance.

For a detailed description of each measure, see Section 2.1.5. For redundancies, we have chosen K-means [22] as the clustering algorithm with distance measures either Euclidean distance or Dynamic Time Warping distance. For the models, we consider the selection from the pool $\mathbb{P}$.

Note that for the meta-data labeling, we consider all the possible combinations of relevance, redundancy measures, and model type, and we evaluate our framework performance on each MTS dataset in the meta set by splitting it into 80% for training the framework and 20% for validation. Even though the meta-task is performed fully offline (only meta-model predictions are output online), this annotation is very resource-consuming because of the big number of combinations. That is why we restrained the size of the metadata to 100 MTS. However, we aim to enlarge this data in the future.

There are different options on how to tackle the meta-learning task. One possible option would be to encode all the combinations of relevance, redundancy measures, and model type, which would lead to a high number of classes compared to the size of the meta-data. Another option is to consider it as a multi-label classification task. However, a classifier's performance on different labels can vary significantly. Therefore, we have chosen to split the task into three learning tasks. The first one is for relevance measure prediction and is a multi-class classification task solved with SVM [264]. The second task is for redundancy measure prediction and is a binary classification task solved with SVM [264]. The third task is for model selection and is a multi-class classification task solved with RF [31]. The choice of the learning algorithm is decided using a cross-validation evaluation of the accuracy of the meta-data.

3.5.1.3 OAMTS Setup:

OAMTS also has a number of hyper-parameters that are summarized in Table 3.1.

We compare OAMTS against the following approaches, which include SoA methods for MTS forecasting. Some of them operate in an online fashion.

Table 3.1: Hyper-parameters of OAMTS and their values for the experiments.

Parameter	Description	Value
p	number of lagged values from each variable	5
ω	size of validation set	25% of the dataset length
μ	Hoeffding-Bound parameter	0.05
ν	Admissible change in the Page-Hinkley Test	0.005
ϱ	Page-Hinkley Test threshold	0.025

SoA Forecasting Models:

- ARIMAX [54]: Auto-Regressive Moving Average model with exogenous variables.
- LSTM [110]: Long Short Term Memory Network which has shown better performance than the remaining neural networks such as MLP and CNN-LSTM and comparable performance with bi-LSTM.
- VAR [17]: Traditional Vector Autoregressive model. Its order is tuned using Akaike Information Criterion (AIC) using the `R-package` 'vars'.
- Drift-aware VAR [16] A recent framework that selects the relevant variables using Pearson-Correlation for the VAR model and updates them following concept-drift detection. It also uses L1-regularization to prevent over-fitting. However, redundancies are not removed.

OAMTS Variants:

- OAMTS-Ran: The variant of OAMTS that is computed using a random selection of relevance and redundancies measures and model.
- OAMTS-VAR: The variant of OAMTS that uses VAR as the forecasting model instead of the automatic model selection. Relevance and Redundancies are selected by the meta-model.
- OAMTS-Rel: The variant of OAMTS that performs adaptive input selection by considering only the relevance.
- OAMTS-Red: The variant of OAMTS that performs adaptive input selection by considering only the redundancy.
- OAMTS-DI-II: The variant of OAMTS that performs model adaptation following concept drift in the input structure (`Drift type I` and `Drift type II`.
- OAMTS-DIII: The variant of OAMTS that performs model adaptation following the changes in the error (`Drift type III`).
- OAMTS-Per: The variant of OAMTS that performs model adaptation periodically without any consideration of concept drift occurrence, with each upcoming 10% MTS observations.
- OAMTS-BG: The variant of OAMTS where we assume we know the background truth of which relevance and redundancies measures to use and which model to select. This is

done by evaluating all the possible combinations on the test set. This variant is used as a reference model to know how well the meta-learning component performs.

3.5.2 Comparing OAMTS to the State-of-the-Art

Table 3.2 presents the average ranks and their deviation for all the methods. A rank of 1 means the method is the best on all the data sets. Therefore, the lower the rank is, the better the method is. For the paired comparison, we compare our method OAMTS against each of the other methods. We counted the wins and losses on each dataset using the RMSE scores. We use the non-parametric Wilcoxon Signed Rank test [270] to compute significant wins and losses (significance level 0.05). In the results in Table 3.2, OAMTS outperforms the baseline methods

Table 3.2: Comparison of OAMTS to different SoA for 66 time series. The rank column presents the average rank and its standard deviation across different time series. A rank of 1 means the model was the best performing on all time series. We report only significant wins and losses of OAMTS against the remaining methods.

Method	Our Method		
	Wins	Losses	Avg.rank
VAR	40	0	7.7±0.9
ARIMAX	20	20	3.0±2.2
LSTM	40	0	7.8±1.4
Drift-aware VAR	40	0	6.7±0.9
OAMTS-VAR	40	0	7.6±0.7
OAMTS-Ran	40	0	5.0±0.9
OAMTS-Rel	40	0	5.9±1.3
OAMTS-Red	40	0	6.3±0.6
OAMTS-Per	39	1	4.3±0.8
OAMTS-DI-II	30	10	3.2±1.2
OAMTS-DIII	7	33	2.9±0.7
OAMTS	-	-	**2.2±0.5**
OAMTS-BG	-	-	**1.9±0.6**

in terms of wins/losses in the pairwise comparison. The online MTS forecasting methods, e.g., the Drift-aware VAR [16] and OAMTS-VAR show inferior performance compared to OAMTS. VAR and LSTM, SoA methods for MTS forecasting, are considerably worse in average rank compared to OAMTS. The most competitive SoA approach to OAMTS is ARIMAX. Nevertheless, it has a higher average rank and a lower performance than our method. VAR is considered to be the most widely used method for MTS forecasting, but it can be seen from OAMTS-VAR that it is not always the best model choice. This is also confirmed by the Drift-aware VAR performance. These results address the research question **Q1**.

3.5.3 Comparing OAMTS to its Variants

It can also be seen that none of OAMTS-Rel and OAMTS-Red is able on its own to reach the performance of OAMTS which shows the importance of both relevance and redundancies consideration in the input selection.

From Table 3.2, we can also see that none of the drift adaptation methods is able on its own to perform as well as OAMTS, which deploys the three drift types to monitor changes in the input dependence structure as well as the model performance. In addition, OAMTS which relies on the informed adaption of the framework using concept drift detection, is better than OAMTS-Per. This can be explained by the fact that unnecessary updates are not always beneficial. This answers the research questions **Q2-Q3**.

3.5.4 Importance of the Meta-learning Component

Table 3.3 presents some examples where we show the ground truth of which are the best relevance and redundancies measures, as well as the model choice for some data sets. It is

Table 3.3: Ground truth of the best model and relevance/redundancies measures for some datasets.

Dataset	Model	Similarity measure	Clustering method
Taxi-1	PLS	Pearson correlation	DTW
Taxi-2	MARS	Euclidean distance	DTW
Taxi-3	PLS	Spearman correlation	DTW
Taxi-4	PLS	Pearson correlation	k-Means
Taxi-5	PLS	Spearman correlation	k-Means
Chengdu-city-1	MARS	Euclidean distance	k-Means
Chengdu-city-2	PLS	Spearman correlation	DTW
Chengdu-city-3	MARS	Spearman correlation	Euclidean

clear from Table 3.3 that there is no one single best relevance and redundancies measures, as well as one optimal model choice, even for MTS data sets extracted from the same data source like Taxi1, 2, 3, 4, and 5, that are extracted from NYC Trip Record Data (Yellow taxi 2021). This justifies the necessity of automating these choices. Random choices would lead to considerably worse performance which is reflected in the performance of OAMTS-Ran in Table 3.2. In addition, comparing OAMTS to OAMTS-BG, we can see a slight difference in the ranks in favor of the course of OAMTS-BG, but it highlights the usefulness of the meta-learning component in our framework for automating all the choices. These results address the research question **Q4**.

3.5.5 Scalability Analysis

In the next experiment, we compare the run-time of OAMTS and its variants against some SoA methods in Table 4.3.

All the reported run-times concern only the online predictions and any operation computed offline is not taken into account. The results demonstrate that OAMTS has a lower run-time than OAMTS-Per. This is due to using drift detection to update the model and the selection of input data only when necessary. This results in faster predictions and less computational requirements. The high deviation of the run-time of OAMTS is due to the different numbers of drifts per time series. This answers the question **Q5**.

3.5.6 Discussion

The empirical results indicate that OAMTS has performance advantages compared to popular MTS forecasting methods. We show that our method, for adaptively selecting input MTS variables and performing the model update, is able to gain excellent and reliable empirical performance in our setting. The informed adaptation following concept drift detection makes our method in addition to better predictive performance, computationally cheaper than blind

Table 3.4: Empirical run-time comparison between different methods in Seconds.

Method	OAMTS	OAMTS -Per	LSTM
Avg. Runtime	34.26	72.12	150.09
± Std.	94.51	35.29	29.26

adaptation methods like periodic ones. In future work, we plan to enhance further the meta-learning component by adding more datasets and annotating them, and establishing a direct mapping to the best combination of measures and model choice as target labels as we assume that there is a link in addition to the MTS characteristics that we tried to cover from different perspectives, between relevance and redundancies measures and the chosen forecasting model. This investigation will make the scope of our future work. In addition, we count on adding more time series clustering algorithms so that we change the mapping to the clustering algorithm directly instead of the relevance measure. We may also think about enlarging the pool $\mathbb{P}$.

3.6 Concluding Remarks

This chapter introduces OAMTS: a novel, practically useful online adaptive framework for multivariate time series forecasting. OAMTS uses adaptive input time series variables selection by investigating relevance and redundancies. Both input variables and learning models are updated in an informed manner following the detection of different types of concept drifts. More specifically, we distinguish between drifts in the dependencies structure between the time series variables and in the forecasting model's performance. The choice of the relevance and redundancies measure, as well as the model, is automated using meta-learning. An exhaustive empirical evaluation, including several real-world datasets and multiple algorithms comparisons, showed the advantages of OAMTS in terms of performance and scalability.

The choice of the forecasting model is automated using meta-learning that exploits the characteristics of the MTS data in question. The model is updated by updating the input with recent time series observations from the same time series variables or newly selected variables. However, once selected, the model type, e.g., VAR or LSTM, is kept fixed. However, we have discussed in the previous chapters that there is no single universal model that is valid for every time series and over time. This is mainly due to the time-varying forecasting models' performance that can be subject to concept drift, as we have shown in this chapter. Therefore, we will investigate next the online single model selection and ensemble members selection, i.e., ensemble pruning over time in an adaptive informed manner.

4

Online Adaptive Single Model Selection and Ensemble Pruning

In this chapter, we propose a set of methods that are devised to select either a single forecasting model or many models to construct an ensemble model in an online adaptive manner.

4.1 Introduction

As we mentioned before, none of the proposed forecasting models in the Literature is universally valid for every application [9]. Even if we consider a particular application, forecasting models' performance is time-dependent [9], [10], [22]. This can mainly be explained by the fact that different forecasting models have different areas of expertise placed over different parts, i.e., time series subsequences, in the input time series. Each of these parts is usually referred to as the so-called region of expertise or a *Region-of-Competence*(RoC) of the model that achieves the best predictive performance for that part, called the expert model [10]. Therefore, appropriate and adaptive model selection in real-time is often required to cope with the time-evolving nature of time series and the fact that forecasting models have a certain expected level of competence in predicting a particular region in the time series. Various tactics have been proposed for selecting a single model, ranging from statistical estimations to the use of meta-learning to learn an appropriate selection strategy.

While some works have focused on the online selection of a single model [271], others have extended on the assumption that no single model is an expert all the time and have proposed to adaptively combine multiple forecasting models into an ensemble model [9], [10], [22]. Ensemble members must also be carefully selected in real-time to account for changes in both the time series and the time-changing forecasting performance of the members. Similar techniques for single model selection can be used for ensemble member selection, i.e., ensemble pruning. As discussed in Section 2.6.2, ensemble pruning is very challenging since finding the optimal subset of models is a combinatorial search problem with exponential computational complexity. Therefore, it is not possible to compute the exact solution by exhaustive search, and an approximate search is required [29], [30]. A taxonomy of ensemble pruning methods using approximate search is presented with details in Section 2.6.2. In this context, Section 4.3 is dedicated to an online adaptive ensemble pruning method that can be applied for single model selection, using a combination of a ranked-based approach together with a clustering-based approach. The ensemble members' ranking stage is made to be drift-aware of the members' performance over time to improve the ensemble's accuracy. Then, a model clustering procedure is applied to enhance the ensemble diversity. This method is denoted in the following by DEMSRC: Drift-aware Ensemble Members Selection using a mixture of Ranking and Clustering-based approaches.

More recently, the concept of *Regions of Competence* (RoCs) has been used for both the selection of a single model [22] and the selection of ensemble members [10], [22] in the task of time series forecasting. Different ways to determine these RoCs have been proposed [9], [22]. Model-type-independent approaches use meta-learning by either training meta-models to predict the performance of candidate models on the most recent time series pattern of p-lagged values [22] or at a particular test time [9], [10]. Model selection approaches in these works are performed online in a blind manner at each time step, i.e., without taking into account the occurrence of significant changes in either the time series data or the performance of the candidate models.

In Section 4.4, we present an online adaptive model selection method that is based on an online clustering of a set of time series sequences contained in an adaptive sliding-window validation set and an evaluation of the candidate models. The RoCs of these models are created by assigning each cluster to the model with the best performance. The proposed method is suitable for both single model selection and ensemble member selection. This method is referred to as OMS-ROC: Online Model Selection using Regions of Competence in the rest of the paper.

In section 4.5, we show how the computed RoCs provide an explanatory tool for the reason for outputting a particular forecast value at a particular time instant and for the reason for selecting a particular individual model or an ensemble member at a particular time instant or interval. Since the RoCs are calculated independently from the model family, the provided explanations are model-agnostic.

We further conduct a comprehensive empirical analysis to validate our methods using 100 real-world time series datasets from different domains. The obtained results show that our methods achieve excellent results in comparison to SoA approaches for the selection of a single model and/or ensemble members and to different baselines for time series forecasting. Note that the time series used for the validation of our methods are univariate. However, the methods can be generalized to MTS data. Further details are given in Chapter 9- Section 9.2.3.

4.2 Related Works

4.2.1 On Online Model Selection and Ensemble Pruning

4.2.1.1 Online Single Model Selection

Online single model selection has been revealed to be challenging in the context of time series forecasting due to the dynamic nature of time series data [9], [10] and the difficulty in estimating the resulting time-evolving models' performance/competence [122]. Given a set of candidate models for performing a well-defined forecasting task, different tactics ranging from statistical estimations to applying meta-learning to learning the adequate selection strategy have been suggested. The approaches for online single model selection can be divided into three main families. The first family of methods is based on approximating a posterior over the expected error of the different candidates using parametric [272] or non-parametric estimation methods [26]. These methods are not practical in the context of online forecasting since continuous composite densities for the error-derived as a function of target and estimated time series values have to be approximated. The results depend largely on the quality of the approximation. The second family consists of using empirical estimation of the unseen error of a given model using an independent validation/calibration dataset. Models with the lowest estimated error are selected subsequently [27]. These methods are quite ineffective in practice since the estimated empirical error is usually lower than the true error. The third family is based on the meta-learning paradigm, where the selection of the adequate method is decided by another machine learning model that learns from the evaluation of previous selection realizations characterized by a set of devised meta features [9]. The design of the meta-learning task, more specifically the meta-data, is generally very challenging due to the difficulty of determining the most adequate meta-feature to be designed to characterize the task and the data [34]. Therefore, model selection is performed in a static manner, i.e., the decision is made once at a time in favor of one model, and this model is used subsequently to forecast all the required values online at test time. The selection can also be updated continuously (i.e., blindly at each time instant or periodically) [9], [10]. However, this is usually expensive in terms of time and resources.

4.2.1.2 Online Ensemble pruning

For online ensemble members selection (i.e., pruning), both theoretical and empirical studies have shown that ensemble performance depends on the performance of its individual models and how diverse they are [29]. However, it is very challenging to decide which models exactly need to be selected in real-time, and even though all the previous works agree that encouraging diversity is beneficial to the performance of an ensemble, but it is hard to enforce it in practice [148], [152]. Therefore, understanding and promoting ensemble diversity remains an important research question in online ensemble learning.

Similarly to single model selection, meta-learning-based methods can be used for ensemble pruning [157], [168]. A meta-learning approach for the estimation of ensemble forecast errors using regression is used in [157] to implement the selection procedure online for each forecast. Another approach that uses lasso regression as a meta-learner is proposed in [273] to combine the forecasts of four base models: a global Recurrent Neural Network (RNN), Theta [274], Trigonometric Box-Cox ARMA Trend Seasonal (TBATS) [275], and Dynamic Harmonic Regression ARIMA (DHR-ARIMA) [97]. Both meta-learner and forecasting models are kept static over time.

In addition to meta-learning, as we mentioned in general for ensemble pruning in Section 2.6.2, ranking and clustering-based approaches are used. Several studies used ranking of the ensemble members following a selection criterion that takes into account the ensemble members' errors and/or ambiguity [30], [162]. However, most of the devised criteria ignore the fact that the error in time series forecasting is directional, and this may lead to inadequate ensemble members selection [162]. In clustering-based approaches, ensemble members are clustered, and a selection of clusters' representatives is performed to increase the overall diversity of the ensemble. The main issue in this method is the choice of the clustering similarity measure, and the optimal number of clusters, i.e., the resulting ensemble size after pruning [30]. The last thing to note is that the ensemble combination/aggregation stage can also be used to serve the pruning stage implicitly by learning sparse ensemble weighting schemes. This is achieved by assigning different weights to the ensemble members and setting some to zero in order to exclude some of the members. Several methods for automatically learning dynamic ensemble weighting schema for time series forecasting are suggested in the literature [176], [177], [276]. However, not all of them guarantee that some of the weights would be set to zero [176], [276].

Again, Dynamic methods can adapt the pruning strategy either in a blind manner over time, i.e., at each time instant at test time, or periodically [9], [10], [157]. However, to the best of our knowledge, none of the existing works in the literature performs online ensemble pruning adaptation in an informed fashion following concept-drift detection in the time series or the ensemble members' performance over time.

4.2.2 On the Use of Regions of Competence for Model Selection

More recently, the concept of the *Regions of Competence* (RoCs) has been used both for the selection of a single model as well as for ensemble members [9], [10], [22] in the time series forecasting task. In [22], at test time, the most similar pattern to the current input (i.e. in this case, a time series input sequence) is determined, and the model with the smallest error on that pattern is selected for prediction. In [9], [10], meta-learning is used to build ML models capable of modeling the competence of each of the ensemble members across the input space. The authors frame their ensemble learning as a ranking task, in which ensemble members are ranked sequentially by their decreasing weight (i.e., the one predicted to perform better is ranked first). Correlation among the output of the members 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. Opposingly, if it is completely uncorrelated with its ranked peers, it gets ranked with its original weight. In the above methods, the meta-models responsible for computing the RoCs are kept static over time. Only distances or error comparisons are performed online in a blind manner at each time step, i.e., without taking into account the occurrence of significant changes in either the time series data or the performance of the candidate models.

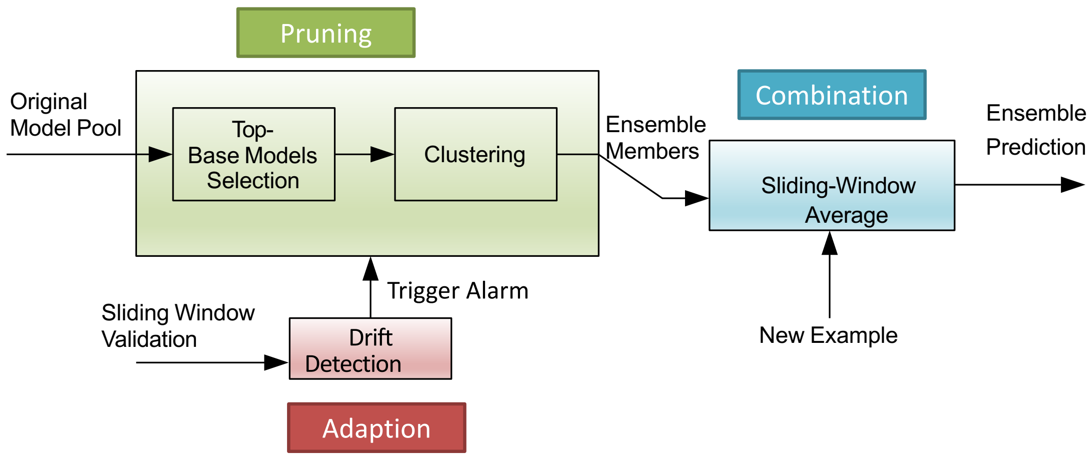


Figure 4.1: DEMSRC Framework.

4.3 Drift-aware Ensemble Members Selection using a Ranking-Clustering-based Approach

This Section introduces **DEMSRC** and its three basic components:

- First, we describe the drift-aware pre-selection step to get the top best-performing single models.
- The second stage consists of first clustering these top models and selecting one representative model for each cluster.
- Finally, each selected model's output is integrated into a weighted average where the weights are set to be inversely proportional to the model's recent loss over a time sliding window.

Figure 4.1 summarizes the main components of DEMSRC.

4.3.1 Preliminaries

We consider the pool $\mathbb{P}$ of M forecasting models as defined previously, trained to forecast a time series X. Let $\hat{x}_{t+h} = (\hat{x}^{f_1}_{t+h}, \hat{x}^{f_2}_{t+h}, \cdots, \hat{x}^{f_M}_{t+h})$ be the vector of forecast values of X at a future time instant $t+h, h \geq 1$ (i.e. x_{t+h}) by each of the models $f_i, i \in [1, M]$ in the pool $\mathbb{P}$ and $\bar{f}_{\mathbb{P}}$ an ensemble model of $\mathbb{P}$ at time instant $t+h$.

Definition 12 ***Online Ensemble Pruning Problem Definition*** *The goal of dynamic online ensemble pruning is to identify the subset of models $\mathbb{S} \subset \mathbb{P}$ that should compose the ensemble at each time step $t+h$ such that the expected prediction error of the pruned ensemble is reduced compared to the full ensemble $\bar{f}_{\mathbb{P}}$ for each forecast.*

$$\underset{\mathbb{S} \subset \mathbb{P}}{\text{argmax}} \quad \mathbb{E}\big[\big(x_{t+h} - \bar{f}_{\mathbb{P}}(\hat{x}_{t+h})\big)^2 | X_{1:t+h-1}\big] - \mathbb{E}\big[\big(x_{t+h} - \bar{f}_{\mathbb{S}}(\hat{x}_{t+h})\big)^2 | X_{1:t+h-1}\big] \tag{4.1}$$

To reach this goal, a two-staged selection procedure is devised. The first stage is a pre-selection stage which aims to keep only accurate single models' forecasts using drift detection in models' performance over time. This stage discards models with poor performance, i.e., whose forecasts inclusion in the ensemble would deteriorate the forecasting accuracy. This deterioration is more perceptible using a simple average for integration and can be covered to some extent using a weighting strategy. The second stage aims to enhance the diversity aspect with the use of clustering.

4.3.2 A Drift-Aware Ensemble Members Pre-Selection

In the previous chapter, the drift-based time series selection is applied in the context of spatio-temporal features selection to be the input of an MTS forecasting model (See Section 3.3.2). Similarly, we can treat the set of ensemble members' forecasts as a set of explanatory variables or causes to our target time series X. To do so, dependencies between the set of ensemble members' forecasts and the time series X can be continuously computed and monitored over a sliding-window validation set. Suppose we want to compute the prediction for time instant $t+1$, the validation sliding-window of size ω over X is defined by the sequence $X_{t-\omega+1:t} = \{x_{t-\omega+1}, x_{t-\omega+2}, \cdots, x_t\}$. Let $\hat{X}^{f_i}_{t-\omega+1:t} = \{\hat{x}^{f_i}_{t-\omega+1}, \hat{x}^{f_i}_{t-\omega+2}, \cdots, \hat{x}^{f_i}_{t}\}$ be the predicted sequence of values by the model f_i on $X_{t-\omega+1:t}, \forall f_i \in \mathbb{P}$.

A subset K of highly correlated models with the target time series X denoted "top-K" based models are selected using a sliding-window similarity measure computed on the time interval $[t-\omega+1:t]$. K is a user-defined hyperparameter. Hereby, we propose to use a custom measure based on Pearson's correlation distance between $X_{t-\omega+1:t}$ and $\hat{X}^{f_i}_{t-\omega+1:t}$ for all the models $f_i \in \mathbb{P}$ and (See Section 2.1.5) denoted as *SRC* - Scaled Root Correlation and defined as:

$$corr(\hat{X}^{f_i}_{t-\omega+1:t}, X_{t-\omega+1:t}) = \frac{\tau - \frac{\sum_{j=1}^{\omega} \hat{x}^{f_i}_{t-\omega+j} \sum_{j=1}^{\omega} x_{t-\omega+j}}{\omega}}{\sqrt{\sum_{j=1}^{\omega} (\hat{x}^{f_i}_{t-\omega+j})^2 - \frac{(\sum_{j=1}^{\omega} \hat{x}^{f_i}_{t-\omega+j})^2}{\omega}} \sqrt{\sum_{j=1}^{\omega} (x_{t-\omega+j})^2 - \frac{(\sum_{j=1}^{\omega} x_{t-\omega+j})^2}{\omega}}} \tag{4.2}$$

$$SRC(\hat{X}^{f_i}_{t-\omega+1:t}, X_{t-\omega+1:t}) = \sqrt{\frac{1 - corr(\hat{X}^{f_i}_{t-\omega+1:t}, X_{t-\omega+1:t})}{2}} \in [0,1] \tag{4.3}$$

where $\tau = \sum_{j=1}^{\omega} \hat{x}^{f_i}_{t-\omega+j} x_{t-\omega+j}$. Naturally, with time-evolving data, dependencies change over time and may follow a non-stationary process. Stationarity in this context can be formulated as follow:

Definition 13 ***Weak stationary Models Dependencies*** *Let $C_t \in \mathbb{R}^{M \times M}$ be a resulting symmetric similarity matrix between the candidate ensemble members and the target time series over the sliding window of size ω (i.e., derived from the above similarity metric Equation 4.3), where $M = |\mathbb{P}|$ and c_t be a vector containing all the elements in C_t where $c_{j,t} \geq c_{j-1,t}, \forall j \in \{1 \dots M^2\}$. Let ν denote the minimum SRC coefficient of $\mathbb{P}$ at the initial instant of its generation t_i. The dependence structure is said to be weakly stationary if the true mean of Δc_t is 0:*

$$\Delta c_t = |c_{1,t} - \nu| \tag{4.4}$$

Following this definition, we can assume that the distance between the most dissimilar model within the same pool of models to the target time series sets its boundary under a form of a logical *diameter*. If this boundary diverges in a significant way over time, a drift is assumed to take place. We propose to detect the validity of such an assumption using the Hoeffding Bound [262], which states that after ω independent observations of a real-valued random variable with range R, its true mean has not diverged if the sample mean is contained within $\pm\epsilon_F$:

$$\zeta = \sqrt{\frac{R^2 \ln(1/\mu)}{2\omega}} \tag{4.5}$$

with a probability of $1 - \mu$ (a user-defined hyperparameter). Once the condition of the *weak stationary dependencies* presented in Definition 13 is violated, an alarm is triggered, and the top single models using C_t are updated. Afterward, the dependency monitoring process is continued by sliding the time window for the next prediction, and the reference *diameter* ν is reset by setting $t_i = t_d$, where t_d is the time instant at which the drift is detected.

4.3.3 Model Clustering

One of the most important aspects of successful ensembles is diversity [10], [148], [151], [152]. Typically, this diversity is initially reflected in the distinctive patterns of the ensemble members' inductive bias derived from the different hypotheses on which each base learner is built to model the input data and its dependence structure. As we mentioned, the enforcement and evaluation of diversity on ensembles for time series data are still quite an unexplored topic-especially for forecasting problems [11]. However, the expected error decomposition for ensemble schema, in general, helps to get an intuition about the importance of diversity [148], [150], [277]. More details are provided in Section 2.6.1. More precisely, the expected error of the ensemble can be decomposed into *bias*, *variance*, and *covariance* terms (See Equation 2.61).

In DEMSRC, we propose a second-stage selection that tries to ensure such diversity through *clustering*. Predictions of top-K selected models of the time sequence $X_{t-\omega+1:t}$ are considered as ω-dimensional feature vectors to cluster these models. To compute clusters for time series, several techniques are proposed in ML literature such as K-means and hierarchical clustering, Dynamic Time Warping clustering [82], [261], [278]. However, one of the main issues presented by time series clustering is the choice of similarity/distance measure as we mentioned and investigated in the previous chapter. Most of the typical distance measures, such as Euclidean distance, do not take dependence structures of time series data into account [83]. To overcome this issue, we used an Improper Maximum Likelihood Estimator based Clustering method (IMLEC) [279], which is based on a multivariate Gaussian model where parameters are estimated using Expectation Maximization algorithm [280]. This method has the advantage over Euclidean Distance-based clustering methods by contributing to achieving near-zero covariance between the time series sequences belonging to different clusters and thus to the reduction of the overall ensemble error. For instance, traditional clustering methods like K-Means with Euclidean distance do not take into account the covariance of the time series. If we consider two candidate time series that have dependence over a high number of components of their ω-dimensional feature space (i.e., high covariance is assumed to take place), the probability of attributing them to the same cluster by fitting the adequate parameters of the Gaussian mixture to the data is higher than simply using a Euclidean Distance-based method, which would probably assign them to different clusters based on their closeness to the current cluster centers. As a result, models belonging to different clusters have a more likely near zero covariance. Therefore, the final step in the selection consists of selecting one representative model for each cluster. We simply select the closest model to each cluster center.

4.3.4 Model Combination

The final selected models are integrated using a sliding-window weighted average [12]. Let $\mathbb{S}$ be the pool of final selected models, i.e., ensemble members selected to take part in the ensemble for the prediction of time instant $t+1$. The final prediction is obtained by:

$$\hat{x}_{t+1} = \sum_{f_i \in |\mathbb{S}|} \frac{\left[(1-\chi_{i,t})\hat{x}^{f_i}_{t+1}\right]}{\sum_{f_i \in |\mathbb{S}|}\left(1-\chi_{i,t}\right)} : \chi_{i,t} \in [0,1], \forall i,t \tag{4.6}$$

where $\chi_{i,t}$ is a normalized version of the recent loss of the model f_i on $[t-\omega+1, t]$ which computation is given by an evaluation metric of interest (i.e., normalized version of the Root Mean Square Error (RMSE) (See Equation 2.57) in our case).

This combination stage using adaptive weighting schema following the recent performance of the final selected ensemble members further enhances the ensemble accuracy and allows for a blind adaptation to drift by shifting the importance of the models in the final decision.

DEMSRC is exhaustively tested over data collected from 100 real-world time series datasets. Details about these experiments are provided in the following Section.

4.3.5 Empirical Experiments

In this section, we present the experiments carried out to validate DEMSRC and to answer the following research questions:

Q1: How is the performance of single candidate models over time and across different time series?

Q2: How is the performance of DEMSRC compared to the State-of-the-Art methods for time series forecasting and to existing dynamic ensemble selection approaches?

Q3: What is the advantage of the performance drift detection mechanism, which triggers the ensemble members' pre-selection, in terms of accuracy?

Q4: What is the impact of clustering, and how does the IMLEC-clustering perform compared to commonly used clustering methods for time series data?

Q5: What is the impact of different combination strategies on performance?

Q6: Is there an advantage in terms of computational resources if the ensemble members selection is made in an *informed* fashion (i.e., only triggered by drift detection alarms)?

4.3.5.1 Experimental Setup

The methods used in the experiments are evaluated using the root mean squared error (RMSE) (See Equation 2.57). In each experiment, the time series data was split into 75% for training and 25% for testing. The results are compared using the non-parametric Wilcoxon Signed Rank test [270]. We use 100 real-world time series from a diverse set of application scenarios, such as transport data, sensor readings, and financial indices. A full list of the used datasets, together with a description, is given in the Appendix in Table A.3 and the code repository is available under this URL [1].

Base Models Setup: We construct the pool $\mathbb{P}$ of candidate single models. We mentioned earlier that there is no single ML model for forecasting that outperforms all the other models on every time series. Hence, we incorporate and test different families of models.

- Traditional time series forecasting:
  - Auto-Regressive Integrated Moving Average ARIMA [67].
  - Exponential Smoothing ETS [54].
- Regression models: are also included in $\mathbb{P}$ and are applied after using time series embedding of dimension $p = 5$ (For more details, see Section 2.4.4). These models include:
  - Gradient Boosting Machines: GBM [263].
  - Gaussian Processes: GP [281].
  - Support Vector Regression: SVR [264].
  - Random Forest: RF [31].
  - Projection Pursuit Regression: PPR [265].
  - Multivariate Adaptive Regression Splines: MARS [266].
  - Principal Component Regression: PCMR [282].
  - Decision Tree Regression: DT [167].
  - Partial Least Squares Regression: PLS [267].
- **Neural networks:**
  - Multi-Layer Perceptron: MLP[268].
  - Long Short-Term Memory network: LSTM [118].
  - Bidirectional LSTM: bi-LSTM [110].
  - Convolutional Neural Networks:
    * CNN-LSTM [283]: convolutional layers of different filter sizes, followed by a max pooling layer, an LSTM layer, a dense layer, and an output layer.
    * Conv-LSTM [269]: the convolutional reading of input is built directly into each LSTM unit. The Conv-LSTM was developed for reading two-dimensional spatial-temporal data but can be adapted for use with univariate time series data.

[1] `https://github.com/AmalSd/DEMSC`

Using different parameter settings for each family, we generate a pool of 43 candidate models.

DEMSRC Setup DEMSRC has a number of hyperparameters that are summarized in Table 4.1. For IMLEC-clustering, we used the R-package of the authors of [279]. The maximum number of clusters is a user-defined parameter. However, it can be automatically reduced by removing outliers and noisy data that cannot be fitted to any cluster. We compare DEMSRC

Table 4.1: Hyperparameters of DEMSRC and their chosen values for the experiments.

Parameter	Value
Number of Top-K Base models	Half of candidate pool $\mathbb{P}$
Maximum number of clusters	Half of Top-K Base models
Hoeffding-Bound δ	0.95
Size of Sliding Window ω	20

against the following approaches, which include State-of-the-Art (SoA) methods for single model selection and ensemble pruning methods for forecasting. Some of them operate in an online fashion. Since DEMSRC contains a model combination/aggregation stage, we compare DEMSRC against SoA methods for online ensemble aggregation. We also compare DEMSRC with some variants of itself. All of these variants, except one, use sliding window weighting for combining the ensemble members' predictions.

SoA Forecasting Models These models include: ARIMA [61], ETS [97], LSTM [118], and CNN-LSTM [112].

SoA for Ensemble Learning These models include: GBM [263], RF [31].

SoA for Online Ensemble Pruning These methods use the same pool of candidate models $\mathbb{P}$ as DEMSRC.

- Ran: consists of a random selection of single models to construct the ensemble.
- Ens: is an ensemble of all the models in $\mathbb{P}$ and uses a simple average to aggregate them. Note that even selecting all the models in the pool $\mathbb{P}$ to take part in the ensemble can be considered a pruning decision.
- SW-Ens [12]: is an ensemble of all the models in $\mathbb{P}$ and uses a sliding-window weighted average to aggregate all the models (See Equation 4.6).
- ADE [9], [10]: was recently developed for an online dynamic ensemble of forecasters construction. A meta-learning strategy is used that specializes candidate single models across the time series to determine their Regions-of-Competence (RoCs). More specifically, it uses a meta-learner to predict the candidate models' errors and selects a set of models based on the estimated errors. Weighting is also based on the estimated errors combined with a softmax. Weights are additionally adapted to take diversity between ensemble members into account such that additional members are discarded.
- DETS [9]: is considered as an advanced version of SW-Ens, selects a subset of members based on recent errors, and uses a smoothing function on the average of recent errors for weighting.

- KNN-RoC(5) [22]: computes static RoCs using a fixed splitting of a validation set into small intervals as input and the rank of the best performing individual candidate model on each interval as a label for a *KNN* classifier, using DTW distance and for many models selection, $K = 5$ is used for comparison. At test time, the *KNN* predicts which candidate models should be selected.

SoA for Online Ensemble Aggregation

- OGD: Online Gradient Descent [284] aggregation over all the models in $\mathbb{P}$. An approach based on online gradient descent that provides theoretical loss-bound guarantees.
- FS: Fixed Share method [276] for aggregation over all the models in $\mathbb{P}$. More details about FS are provided in Section 2.6.3.
- EWA: Exponential Weighting [276] Aggregation over all the models in $\mathbb{P}$. More details about EWA are also provided in Section 2.6.3.
- MLPOL: Polynomial Potential aggregation rule [175] for aggregation over all the models in $\mathbb{P}$. More details about MLPOL are also provided in Section 2.6.3
- Stacking [33]: An approach for ensemble aggregation using Random Forest as a meta-learner to learn the ensemble weighting schema.

DEMSRC Variants

- OCL: Same as our method but without the Top-K single models selection stage. Clusters are updated in a drift-aware manner as described for DEMSRC.
- OTOP: Only the drift-aware pre-selection of the Top-K models based on correlation (no clustering is applied afterward).
- DEMSRC-kMeans: The clustering method is replaced with K-Means with Euclidean distance (the number of clusters is tuned using the average silhouette method).
- DEMSRC-DTW: The clustering method is replaced with Dynamic Time Warping clustering.
- DEMSRC-stacking: The stacking variant differs from our method only in the combination step. Instead of using a sliding window weighted average, a stacking approach is used in this variant for aggregation, and PLS is used as the meta-learner.

4.3.5.2 On the Performance of Single Candidate Models

As a preliminary analysis, we evaluate the performance of these models on a subset of the 100 datasets and compute the rank of each model based on its RMSE. A rank of one on one dataset means that the corresponding model is the best-performing one on that dataset.

Figure 4.2 demonstrates that all the forecasting models have a high variance as their performance changes across different time series, showing that there is no obvious best-performing model. This prompts the fact that not only an ensemble of diverse forecasting

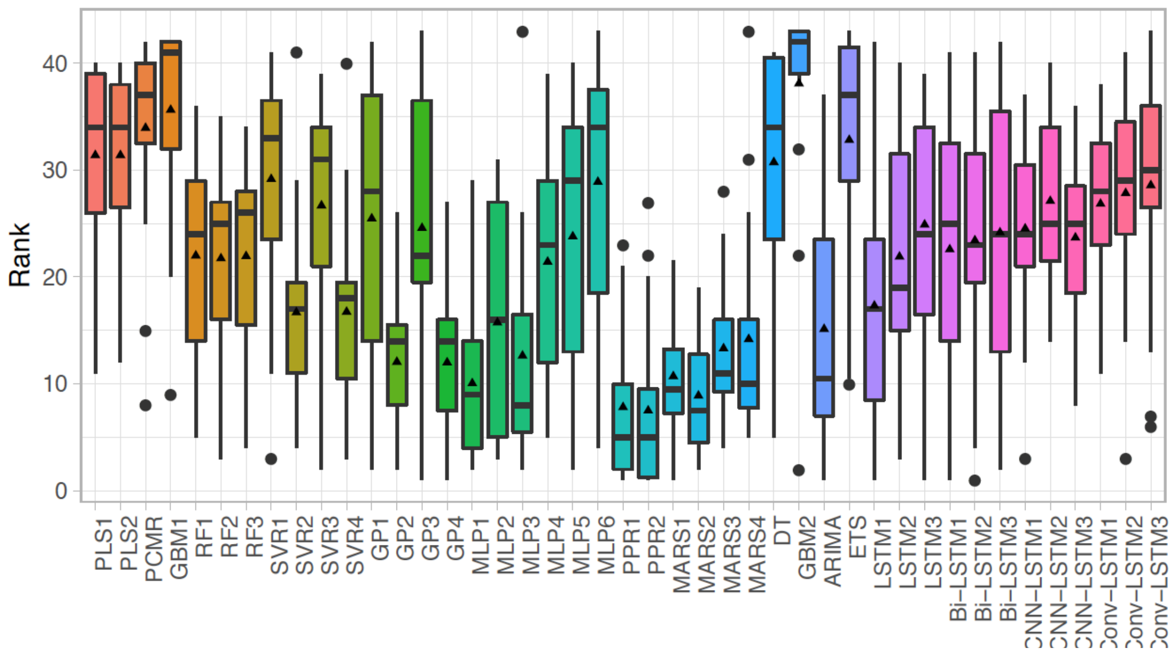


Figure 4.2: Distribution of the ranks of the single models across 20 different time series datasets, models within the same family in the x-axis have similar names attached to different numbers for stating different models' parameters.

models is a more desirable option, but also efficiently selecting and combining them would make forecasting more accurate.

4.3.5.3 Comparing DEMSRC to the State-of-the-Art

Table 4.2 presents the average ranks and their deviation for all methods. For the paired comparison, we compare our DEMSRC against each of the other methods. We counted the wins and losses for each dataset using the RMSE scores (See Equation 2.57). We use the non-parametric Wilcoxon Signed Rank test [270] to compute significant wins and losses, which are presented in parenthesis (i.e., the significance level =0.05).

DEMSRC has advantages over the compared methods except for ADE. The approaches for pruning and combining individual forecasters, which are Ens, SW-Ens, DETS, OGD, FS, EWA, MLPOL, and KNN-RoC(5), show a big difference in the average rank compared to DEMSRC. ARIMA, ETS, and LSTM, the SoA method for forecasting, have a big difference in the average rank compared to DEMSRCas well. Common ensemble methods like RF, GBM, SW-Ens, and Stacking compare poorly to all methods specialized for combining forecasters. The two competitive approaches to our method are ADE and DETS, with DETS having a higher average rank but performing well in the pairwise comparison. ADE is competitive to DEMSRC. It is comparable to DEMSRC in terms of wins and losses but has also a higher average rank. Regarding research question **Q2**, our results show that DEMSRC is competitive with ADE and outperforms other SoA approaches for time series forecasting, including the online methods. The average rank of DEMSRC is better than ADE, but we do not see the main advantage of our method in the performance but more in the complexity and computational requirements, which we will discuss later.

Table 4.2: Paired comparison between DEMSRC and different baseline methods for 100 time series datasets. The rank column presents the average rank achieved by each model and the standard deviation of the rank across the different datasets. A rank of 1 means the model was the best performing in all these datasets.

Method	Our Method		
	Losses	Wins	Avg. Rank
RF	10(5)	89(73)	16.25 ± 2.7
GBM	9(3)	91(16)	18.07 ± 2.0
ARIMA	10(1)	87(71)	13.50 ± 4.1
ETS	11(1)	85(71)	14.37 ± 3.2
LSTM	11(3)	86(71)	13.85 ± 5.3
CNN-LSTM	12(4)	85(71)	12.76 ± 6.5
Ens	13(2)	85(73)	11.42 ± 3.8
SW-Ens	15(4)	82(74)	11.85 ± 3.5
ADE	48(20)	50(40)	4.24 ± 3.0
DETS	30(15)	67(50)	7.15 ± 4.8
KNN-RoC(5)	32(15)	68(54)	6.48 ± 3.5
EWA	29(11)	71(62)	7.3 ± 2.6
FS	18(10)	80(63)	9.5 ± 3.6
OGD	25(14)	70(62)	6.8 ± 2.5
MLPOL	22(10)	74(60)	7.2 ± 3.5
Stacking	15(8)	85(74)	14.9 ± 3.8
OCL	30(25)	70(64)	7.9 ± 3.7
OTOP	17(12)	83(74)	10.2 ± 4.9
DEMSRC-kMeans	14(9)	86(76)	10.7 ± 5.1
DEMSRC-DTW	22(18)	78(70)	11.6 ± 3.5
DEMSRC-stacking	31(9)	69(56)	5.7 ± 5.3
DEMSRC	-	-	$\mathbf{3.87 \pm 2.7}$

4.3.5.4 Comparing DEMSRC to its Variants

Comparing the three different clustering methods (DEMSRC-kMeans, DEMSRC-DTW, DEMSRC(IMLEC)), we can see the clear advantage of the IMLEC-clustering. Using Dynamic Time Warping clustering gives a slight improvement over K-Means, but both of them do not improve the pre-selection. Using IMLEC-clustering improves the performance drastically. This can be explained partially by enhancing the ensemble diversity aspect discussed in Section 2.6.1.

We see that the aggregation part of our method has a small impact by comparing DEMSRC to DEMSRC-stacking. Both variants using IMLEC-clustering, with either a sliding window weighted average or a stacking approach for the aggregation stage, have clear advantages over the other clustering variants (see question **Q4**). To further investigate the differences in the average ranks, we use the post-hoc Bonferroni-Dunn test [285] to compute critical differences. We present in figure 4.3 the critical differences of the methods by considering the average rank. The only variant where the difference in average rank to DEMSRC is not critically different is the stacking variant (DEMSRC-stacking). However, the stacking variant's average rank is higher. This answers the research question **Q5** regarding the impact of each component of the method.

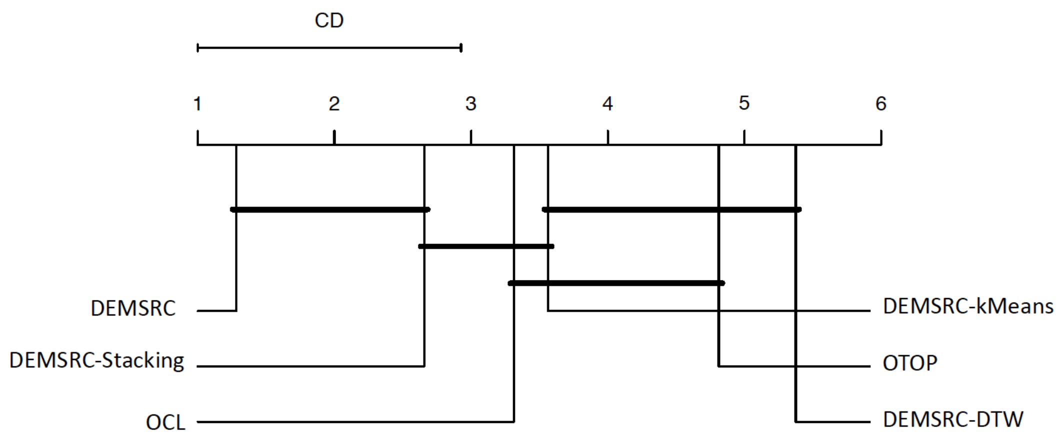


Figure 4.3: Critical difference diagram for the post-hoc Bonferroni-Dunn test, comparing OAMTS against variants of our method.

4.3.5.5 Importance of the Drift-aware Models Selection

We can also answer the research question **Q3**, asking about the impact of using drifts detection in the models' dependence on the target time series to trigger the new ensemble members' re-selection. Performance-wise, we can see that our method performs on the same level or even slightly better than the best online SoA approach, namely ADE. The motivation for using drift detection is to update the ensemble only when necessary. This should result in faster predictions and lower computational requirements (see question **Q6**). We see in Table 4.3 that the average run-time of ADE is more than twice as long as the run-time of our method. The high deviation in the run-time of our method is due to different datasets having more or fewer drifts detected.

Table 4.3: Empirical run-time comparison between DEMSRC and the most competitive State-of-the-Art method (ADE).

Method	Avg. Run-time in sec.
DEMSRC	97.69 ± 54.4
ADE	166.95 ± 15.3

4.3.5.6 Final Remarks

We presented results that empirically showed that DEMSRC has performance advantages compared to other ensemble methods for forecasting and is competitive with the most recent SoA approaches for dynamically pruning and combining forecasting methods.

We show that our method, using a combination of a performance-based pre-selection and clustering, is able to perform at a high level. The pre-selection ensures that only accurate models are used in the ensemble. The clustering groups similar models based on their predictions together. We then select only clusters' representatives. This leads to an ensemble with accurate and diverse members, which has been theoretically shown to be required for an ensemble to outperform its members. None of these two stages is able on its own to reach the full performance of DEMSRC, but the combination of both is very powerful.

The usage of a drift detection mechanism for model selection enables our method to construct a new ensemble, given changes in the nature of the dependencies between the single candidate models and the target time series. If there is no change, then there is also no need to construct a new ensemble. Therefore, drift detection reduces computational efforts.

DEMSRC method and the complex meta-learning approach of ADE perform on the same level. To reach the same performance, we only need pre-selection and clustering triggered by drift detection. Therefore, compared to ADE, which needs to train a meta-model for each candidate, our method is computationally cheaper. For the experiments, a prediction with ADE needed, on average, twice as long as our method.

DEMSRC is also suited to online single model selection by selecting the Top-1 best-performing model in the pre-selection stage.

4.4 Online Model Selection using Regions of Competence

In this section, we present OMS-ROC and its main stages.

- In the first stage, we train several candidate forecasting models from different families of models offline.
- The second stage consists of determining the RoCs of these models using sliding windows over a validation set. The RoCs are computed by splitting the validation set into equally-sized sequences and clustering them. An evaluation of the candidate models on the sequences in each cluster is performed. The model with the best average performance for a given cluster is considered as the expert for that cluster. In other words, this cluster is considered an RoC for this model.
- In the third stage, to generate a forecast at a given time t_f, the distance from the current input time sequence of p observations to the calculated cluster centers (i.e., RoCs centers) is measured. When a single model is selected, the model corresponding to the RoC center with the smallest distance is selected to predict the time series value at t_f. In the case of ensemble pruning, a distance ranking is computed, and the top models with the top closets RoCs are selected to form the ensemble. The final prediction is computed using averaging.

The RoCs (i.e., clusters) are continuously updated by adding the most recent recorded pattern at the time of testing to the RoC of the model with the best performance on that pattern, regardless of its proximity to the other RoCs. A concept-drift detection mechanism is used to check for the occurrence of significant changes in each cluster over time. If at least one change is detected, a re-clustering is triggered, and the assignment of RoCs to the different candidate models is updated. In addition, the RoCs can be used to provide suitable explanations for the reason for selecting a particular model(s) at a particular time interval or point in time. Practical examples of these explanations are presented in detail in Section 4.5.

4.4.1 Preliminaries

We use the same notations defined in Section 4.3.1. We divide the time series X_t into $X_\omega^{train} = \{x_1, x_2, \cdots, x_{t-\omega}\}$ and $X_\omega^{val} = \{x_{t-\omega+1}, x_{t-\omega+2}, \cdots, x_t\}$, with ω a provided window size. X_ω^{train}

is used for training the models in $\mathbb{P}$ and X_{ω}^{val} is used to compute the RoCs since to measure models performance, both true and predicted values of the time series are required. For OMS-ROC, the candidate models in $\mathbb{P}$ are trained on X_{ω}^{train} using the same number p of lagged values of the time series as input to model the following value in the time series.

4.4.2 Online Model Selection

4.4.2.1 RoCs Computation

First, we divide X_{ω}^{val} into equally sized subsequences of length p. In order to increase the number of resulting subsequences, the division is performed in a step-wise manner, i.e., at each time step $t_j, \forall t_j \in [t-\omega+1, t-p+1]$, we extract the following $p-1$ values. The extracted subsequence at time t_j is denoted by $s_{t_j}^p = \{x_{t_j}, x_{t_j+1}, \cdots, x_{t_j+p-1}\}$. Since all the models are built to predict the future value of a time series using the past lagged p values, for each $s_{t_j}^p$, we also store the subsequent value x_{t_j+p} (i.e., the value coming after the p observations), denoted here as $x^{s_{t_j}^p}$. The subsequences $S^p = s_{t_j}^p, \forall t_j \in [t-\omega+1, (t-p+1)]$ are clustered into K-clusters using K-Means clustering with Euclidean distance D_E. Each cluster is expected to enclose subsequences that reveal similar patterns. The next step consists of determining which candidate model is an expert than the others in predicting the upcoming values once observing a pattern similar to the patterns enclosed in each cluster. To do so, for each cluster $C_k, \forall k \in [1, K]$, an evaluation of each of the models in $\mathbb{P}$ on predicting the value coming after each subsequence in C_k is performed.

$$\epsilon_{C_k}^{f_i} = \frac{1}{c_k} \sum_{s_l^p \in S_k^p} \left(x^{s_l^p} - (\hat{x}^{s_l^p})^{f_i}\right)^2 \tag{4.7}$$

where c_k is the size of the cluster C_k, S_k^p is subset of S^p that are enclosed in the cluster C_k, i.e., S_k^p contain s_l^p with $l \in [1, c_k]$, $x^{s_l^p}$ is the subsequent value of each s_l^p and $(\hat{x}^{s_l^p})^{f_i}$ its corresponding forecast by $f_i \in \mathbb{P}$. C_k gets assigned as the RoC of the model f_z satisfying:

$$f_z = \underset{f_i \in \mathbb{P}}{\operatorname{argmin}}\, \epsilon_{C_k}^{f_i}, \forall k \in [1, K] \tag{4.8}$$

In this way, each cluster $C_k, \forall k \in [1, K]$ is assigned to an expert model $f_z, z \in [1, M]$. One model can acquire more than one RoC (i.e., cluster) as it can be an expert in more than one particular pattern. It is also possible that one model never gets selected if it performs worse than the remaining candidates on all the clusters. Its RoCs are then empty. The set of RoCs of a model f_z are denoted by $RoC_z = \{R_z^1, \cdots, R_z^{n_z}\}$. Since $R_z^n, n \in [1, n_z]$ is originally a cluster, its center is denoted by $\bar{r}_z^n$.

4.4.2.2 Online Forecasting

At test time, after computing the candidate models' RoCs, an online decision on model selection for forecasting the value of X at $t+h$ (assume $h = 1$ for simplicity) has to be made.

Single Model Selection The models in $\mathbb{P}$ are devised such that they use the same p-lagged values of the time series as input, $input_t^p = \{x_{t-p+1}, \cdots, x_t\}$, $(t \geq p)$. To perform the selection, the distance of the input pattern $input_t^p$ to the RoCs centers of each candidate model is measured. Euclidean distance D_E is used to measure the similarity between $input_t^p$ and each $\bar{r}_z^n, \forall n \in [1, n_z], \forall z \in [1, M]$, The model f_s satisfying :

$$f_s = \underset{\substack{n \in [1,n_z]; \\ z \in [1,M]}}{\operatorname{argmin}} D_E(\bar{r}_z^n, input_t^p) \tag{4.9}$$

is selected to forecast $t+1$.

Ensemble Pruning As we mentioned before, the expected error of the ensemble of models in $\mathbb{P}$ at $t+1$ can be decomposed into a weighted average error term of the single models in $\mathbb{P}$ which reflects the ensemble *accuracy* and an ambiguity term which is simply the variance of the ensemble around the weighted mean of its members and it reflects the ensemble *diversity* [148], [150] (More details in Section 2.6.1). Since different RoCs (i.e. clusters of patterns) centers may belong to the RoCs set of the same model, we start first by computing the minimum distance d_z^t of $input_t^p$ to the set of RoCs per model $f_z, \forall z \in [1, M]$.

$$d_z^t = \min_{n \in [1,n_z]} D_E(\bar{r}_z^n, input_t^p) \tag{4.10}$$

We sort these computed minimums in an ascending order $D^t = \{d_{z_1}^t, d_{z_2}^t, \cdots, d_{z_{Top_K}}^t, \cdots, d_{z_{z_N}}^t\}$ with $z_j \in [1, M], \forall j \in [1, M]$. We select the set of Top-K models $\mathbb{S}_{Top_K} \subset \mathbb{P}$ corresponding to the Top_K smallest d_z^t. Considering only the minimum distances to the RoCs for the selection of many models (Equation 4.9) may lead to the selection of the same models more than one time since one model can be an expert on more than one RoC. However, computing the minimum per model (Equation 4.10) enforces the selection of distinct models that are experts on different clusters of patterns. This encourages ensemble *diversity* which is a desirable property for ensemble learning. In addition, the selection following the minimum per model guarantees also the selection of models that have expertise in similar patterns to the current pattern, which enhances the ensemble *accuracy*.

4.4.2.3 RoCs Adaptation

At test time, the RoCs of the models are enriched continuously with new patterns that reveal new potential areas of the models' competence. That is why in the first stage, we select the model whose RoCs will be enriched based on the performance, and in the second stage, we consider the closeness. After forecasting the value of X at $t+1$, the most recent pattern $input_t^p$ is added to the RoC of the model with the best performance at $t+1$ using $input_t^p$ as input, regardless of its proximity to the other RoCs:

$$f_b = \underset{f_j \in \mathbb{P}}{\operatorname{argmin}} \left(x_{t+1} - \hat{x}_{t+1}^{f_j}\right)^2 \tag{4.11}$$

Since f_b may have many RoCs (i.e., many clusters of patterns) $RoC_b = \{R_b^1, \cdots, R_b^{n_b}\}$, $input_t^p$ is added to $R_b^{n_c}$ whose center is the closest:

$$R_b^{n_c} = \underset{n \in [1,n_b]}{\operatorname{argmin}} D_E(\bar{r}_b^n, input_t^p) \tag{4.12}$$

where $\bar{r}_b^n$ is the center of R_b^n.

Changes in the clusters are continuously monitored over time. Once an update of a cluster $R_b^{n_c}$ that belongs to the RoCs of model f_b takes place, we test whether a significant change has been applied to this cluster or not. To do so, we compute the distance between the two most dissimilar sequences in this cluster and compare it to the initial calculated distance δ_i. The distance is treated as time series where δ_{t_d} is its value at time t_d. If at this time instant, $R_b^{n_c}$ does not undergo any change, the difference automatically gets the value of the previous time instant, i.e.,$\delta_{t_d} = \delta_{t_d - 1}$. If it gets updated, we compute the new distance by taking into account the addition of the new pattern.

Definition 14 (Weak stationary RoC structure) *The RoC structure is said to be weakly stationary if the true mean of Δ_t is 0 with:*

$$\Delta_t = |\delta_t - \delta_i| \tag{4.13}$$

As we explained previously for different drift types, e.g., drift in the models' errors, and drift in the dependencies between the models, following this definition, we can assume that the distance between the two most dissimilar sequences within the same RoC sets its boundary under a form of a logical *diameter*. If this boundary diverges in a significant way over time, a drift is assumed to take place. We propose to detect the validity of such an assumption using the Hoeffding Bound [262], which states that after N independent observations of a real-value random variable with range R, its true mean has not diverged if the sample mean is contained within $\pm\zeta$:

$$\zeta = \sqrt{\frac{R^2 \ln(1/\mu)}{2N}} \tag{4.14}$$

with a probability of $1 - \mu$ (a user-defined hyperparameter). Once the condition of the *weak stationary RoC structure* presented in Definition 14 is violated, a re-clustering is triggered, and the RoCs assignment is updated. This adaptation ensures the enrichment of RoCs without altering the homogeneity of the clusters over time. This makes our selection procedure accurate over time since we only consider the distances to the centers of the clusters and also promotes the proposed explainability aspects that rely on clusters' visualization. Further details are provided in Section 4.5.

4.4.3 Empirical Experiments

We present the experiments carried out to validate OMS-ROC and to answer these research questions:

- **Q1:** How does OMS-ROC perform compared to the State-of-the-Art (SoA) and existing online single model selection and online ensemble pruning methods for time series forecasting?
- **Q2:** What is the advantage of updating the RoCs by taking into account at the first stage the models' performance and then the topological closeness?
- **Q3:** What is the advantage of updating the RoCs in an informed fashion (i.e., after drift detection)?

- **Q4:** How scalable is OMS-ROC in terms of computational resources compared to the most competitive online model selection methods? and what is the computational advantage of the *drift-aware* adaptation of the models' RoCs?

4.4.3.1 Experimental Setup

The methods used in the experiments are evaluated using the root mean squared error (RMSE) (See Equation 2.57). The used time series was split using 50% for training (X_ω^{train}), and 25% for validation (X_ω^{val}) and 25% for testing. We use the same 100 real-world time series datasets described in the previous experiments and presented with the details in the appendix Table A.3. We use the same pool $\mathbb{P}$ of candidate models as in the previous experiments composed of the 43 forecasting model. For further details, see Section 4.3.5.1. The code of the method can be found under this repository [2].

OMS-ROC Setup: OMS-ROC has also a number hyperparameters that are summarized in Table 4.4.

Table 4.4: Hyperparameters of OMS-ROC and their values for the experiments.

Parameter	Description	Value
p	number of lagged values (input pattern and the RoCs subsequences sizes)	5
ω	size of validation set	25% of the dataset length
K	number of clusters in K-means	automatic in R-package NbClust
top_K	selected top of models in the ensemble	5
μ	Hoeffding-Bound parameter	0.05

SoA Methods Setup We denote by OMS-ROC-single, the variant of our method that selects the best single model, and by OMS-ROC-ens the variant of our method that performs ensemble pruning (See Section 4.4.2). We compare **OMS-ROC** against the following approaches, which include SoA methods for single model selection and ensemble pruning for time series forecasting. Some of them operate in an online manner.

SoA Forecasting Models:
These models include:ARIMA [61], ETS [97], LSTM [118], and CNN-LSTM [112].
Online SoA for single model selection:
OMS-ROC-single is compared against these methods.

- KNN-RoC(1) [22] computes static RoCs using a fixed splitting of a validation set into small intervals as input and the rank of the best performing individual candidate model on each interval as a label for a *KNN* classifier, using DTW distance and for single model selection $K = 1$, is used for comparison. At test time, the *KNN* predicts which candidate model should be selected.

- ADE-Single [9], [10] ADE was recently developed for an online dynamic ensemble of forecasters construction. A meta-learning strategy is used that specializes candidate single models across the time series to determine their RoC (See Section 4.2.2 and Section

[2] `https://www.dropbox.com/sh/4wy44r4cfnsq6yr/AACYU5QKs9AVwMEL1MzVLddGa?dl=0`

4.3.5.1). However, instead of selecting many models, we select the best-performing model using the same principle.

- DETS-single [9], at each test time instant, it picks a single base model to make a prediction. The picked model is the one that has the lowest loss prediction by the meta models (See Section 4.3.5.1).

Online SoA for ensemble pruning: OMS-ROC-ens is compared against these methods that use the same pool of candidate models $\mathbb{P}$ as OMS-ROC-ens: Ran, Ens, KNN-RoC(5), ADE, and DETS are the same SoA ensemble methods described in Section 4.3.5.1. We also compare OMS-ROC-ens to our previously developed method DEMSRC and its variants OCL and OTOP.

OMS-ROC-single/ens Variants: we compare OMS-ROC, i.e., in both cases OMS-ROC-single or OMS-ROC-ens denoted here by OMS-ROC-single/ens to various variants of themselves:

- OMS-ROC-single/ens-st: is the same as OMS-ROC-single/ens, but the RoCs are not updated using the drift detection mechanism. The RoCs are computed and stored offline, and only model selection takes place online.
- OMS-ROC-single/ens-Per: is the same as OMS-ROC-single/ens, but the RoCs are updated periodically in a blind manner (i.e., without taking into account the occurrence of concept drifts) with a periodicity of each upcoming 10% time series data observations.
- OMS-ROC-single/ens-dis: is the same as OMS-ROC-single/ens, but the update of the RoCs is done by considering only the closeness to the pre-computed RoCs independently of the models' performance.

4.4.3.2 Comparing OMS-ROC to the State-of-the-Art

For the paired comparison, we compare our method OMS-ROC against each of the other methods. We counted the wins and losses of OMS-ROC on each dataset using the RMSE scores. We use the non-parametric Wilcoxon Signed Rank test [270] to compute significant wins and losses, which are presented in parenthesis (significance level 0.05). Tables 4.5 and 4.6 present the average ranks and their deviation for all methods for OMS-ROC-single and OMS-ROC-ens, respectively.

From the results in these two Tables, we can see that OMS-ROC-single outperforms the baseline methods in terms of wins/losses in the pairwise comparison. The online single model selection approaches, e.g., KNN-RoC(1), ADE-Single, and DETS-single show inferior performance compared to OMS-ROC-single. ARIMA, ETS, LSTM, and CNN, SoA methods for forecasting have considerably worse average ranks compared to OMS-ROC-single. Similarly, OMS-ROC-ens outperforms all the online SoA methods for ensemble pruning, namely, KNN-RoC(5), ADE, DETS, OCL, OTOP, and DEMSRC, as well as the standard ensembling techniques such as averaging all the methods in Ens or opting for random pruning in Ran. The ensemble version of OMS-ROC also has performance advantages over the single model selection version.

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

Method	Losses	Wins	Avg. Rank
ARIMA	9(8)	91(89)	9.90 ± 4.39
ETS	7(6)	92(84)	10.04 ± 4.87
lSTM	19(10)	80(70)	8.97 ± 3.61
CNN-LSTM	20(8)	75(73)	8.00 ± 3.70
KNN-RoC(1)	25(7)	72(70)	7.39 ± 3.59
ADE-Single	22(14)	75(71)	7.8 ± 4.18
DETS-Single	22(16)	78(73)	8.41 ± 2.95
OMS-ROC-single-Per	45(19)	55(30)	3.11 ± 2.7
OMS-ROC-single-st	45(19)	57(30)	4.28 ± 3.90
OMS-ROC-single-dis	40(28)	60(40)	5.01 ± 3.1
OMS-ROC-single	-	-	$\mathbf{2.3 \pm 2.0}$

Table 4.6: Comparison of OMS-ROC-ens to different SoA methods on 100 time series datasets. The rank column presents the average rank and its standard deviation across different time series. A rank of 1 means the model was the best performing in all the time series.

Method	Losses	Wins	Avg. Rank
ARIMA	5(3)	95(93)	15.90 ± 4.35
ETS	4(3)	96(96)	16.04 ± 4.26
lSTM	15(8)	85(83)	12.97 ± 3.60
CNN-LSTM	20(6)	80(78)	10.00 ± 3.60
Ran	20(6)	80(71)	10.30 ± 3.59
Ens	17(12)	83(72)	11.84 ± 4.18
KNN-RoC(5)	22(16)	78(73)	8.41 ± 3.59
ADE	31(22)	69(61)	6.28 ± 4.18
DETS	21(18)	79(67)	9.37 ± 3.95
OTOP	30(18)	70(65)	6.7 ± 2.90
OCL	32(21)	68(64)	6.2 ± 2.60
DEMSRC	32(21)	68(64)	6.15 ± 4.35
OMS-ROC-ens-Per	38(21)	62(61)	5.45 ± 4.26
OMS-ROC-ens-st	34(26)	66(61)	6.08 ± 3.60
OMS-ROC-ens-dis	36(25)	64(59)	5.67 ± 3.59
OMS-ROC-single	45(40)	55(50)	3.3 ± 2.7
OMS-ROC-ens	-	-	$\mathbf{2.9 \pm 2.18}$

More particularly, we can see in Table 4.6 that OMS-ROC is considerably better than DEMSRC. This can be explained by the fact that while OMS-ROC starts in the first place by clustering the models to promote ensemble *diversity* and then introducing ranking to eliminate the worst performing models, DEMSRC starts by performing ranking at the first place which may kill to some extent *diversity*. Hence, even though clustering is performed afterward to select diverse models, the ranking can already lead to clustered models around the most recent pattern (i.e., high agreement between the models). As a result, the second stage of clustering becomes profitless. These results address the research question **Q1**.

4.4.3.3 Comparing OMS-ROC to its Variants

Comparing OMS-ROC-single/ens to their different variants, we see a clear advantage in using all the choices in our method. First, the update of the RoCs by considering in the first place the candidate models' performance, and then in the second place, the topological closeness to the pre-computed RoCs is shown to be better than updating the RoCs whose patterns are the closest to the current one. This is proven by the clear better performance of OMS-ROC-single/ens compared to OMS-ROC-(single/ens)-dis. This answers the research question **Q2**.

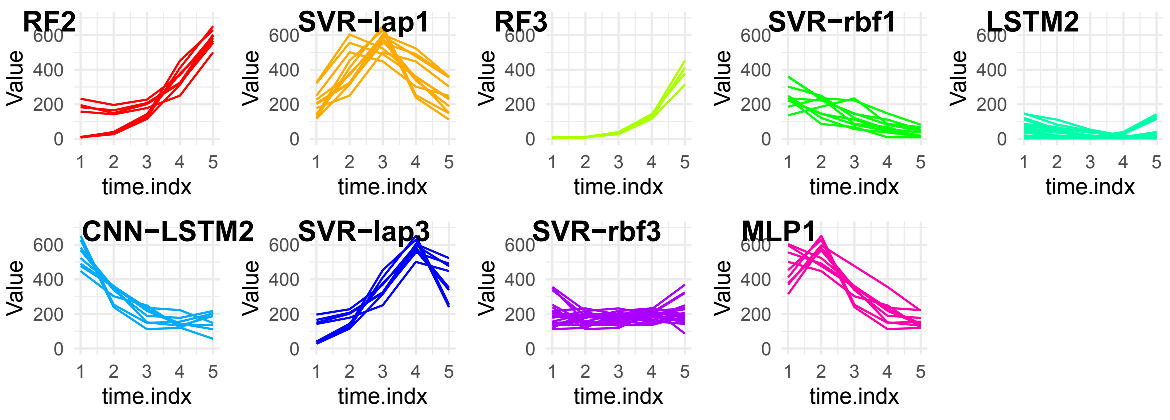
4.4.3.4 Importance of the Drift-aware Adaptation

Our RoCs adaptation manner contributes to creating richer information about possible new appearing RoCs of the different candidates that are included within the recently acquired time series data points. OMS-ROC-(single/ens)-st is even better than OMS-ROC(single/ens)-Per, which shows that unnecessary updates are not always beneficial. Opposingly, OMS-ROC-(single/ens), which relies on an informed adaptation of the RoCs using concept drift detection, is better than OMS-ROC(single/ens)-Per and OMS-ROC(single/ens)-st. This can be explained by the fact that the update of the RoCs is only beneficial when concept drifts in the RoCs structure are detected. These drifts indicate either a change in the competence of the models over time or the appearance of new regions with the newly acquired patterns which have not been seen before and may be relevant for model selection for future forecasts. Taking into account the probable appearance of new patterns in the time series data is helpful for the selection of models since a knowledge of which models are more adequate to handle these patterns if they ever reoccur again is gained, and the old sets of RoCs are enriched.

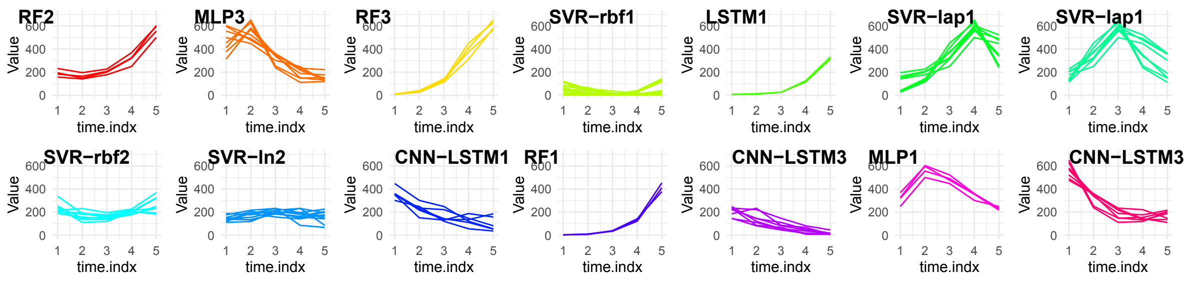
Figure 4.4 shows an illustrative example of the RoCs computation before and after drift detection. New RoCs are discovered, and new models appear to be competent, such as CNN-LSTM1 and CNN-LSTM3. The RoCs (clusters) get more refined, and their homogeneity is restored after the adaptation. For example, RF2 and SVR-rbf3 disappeared to give their place to SVR-rbf1/2 and SVR-ln2. This answers the research question **Q3**.

4.4.3.5 Scalability Analysis

In the next experiment, we compare the run-time of OMS-ROC-(single/ens) and its variants to some SoA methods, namely ADE-Single [10] (forOMS-ROC-single) and DEMSRC(for OMS-ROC-ens), in Table 4.7. Note that the computation of the RoCs is done offline, and the reported



(a)



(b)

Figure 4.4: RoCs for the candidate models before (top) and after (bottom) drift detection. Note same model name with different numbers means the same family of models but with different parameters, like MLP1 and MLP3 have different numbers of layers.

run-time is calculated only for computing the time series predictions in real-time by searching the most similar RoC to the current input pattern $input_t^p$. OMS-ROC relies on informed updates (i.e., using a drift detection mechanism). OMS-ROC-Per relies on a periodic update of the RoCs. The reported run-time for OMS-ROC-(single/ens) and OMS-ROC-(single/ens)-Per takes into account the computation of new RoCs once a drift is detected. ADE-Single [9] relies on a blind update of the meta-learning strategy behind the selection. DEMSRC relies on a drift-aware update of the ensemble pruning as explained in Section 4.3.2. All the reported run times concern only the online predictions, and any operation computed offline is not taken into account.

The results demonstrate that OMS-ROC-(single/ens) and its variants have a lower average run-time than ADE-Single and are comparable to DEMSRC. OMS-ROC-(single/ens) has lower run-time compared to OMS-ROC-(single/ens)-Per. This is achieved thanks to the informed RoCs adaptation using drift detection. This results in faster predictions and lower computational requirements. The high deviation of the run-time of OMS-ROC-(single/ens) is due to the different numbers of drifts per time series. This answers the question **Q4**.

4.4.3.6 Final Remarks

The empirical results indicate that OMS-ROC has performance advantages compared to popular forecasting methods and the most recent SoA approaches for online model selection and ensemble pruning for time series forecasting. We show that OMS-ROC designed for

Table 4.7: Empirical run-time comparison of different methods in Seconds.

Method	Avg. Run-time	± Std.
ADE-Single	157.87	56.40
DEMSRC	8.42	18.30
OMS-ROC-single	**5.33**	**4.80**
OMS-ROC-single-st	0.90	1.65
OMS-ROC-single-Per	154.11	204.22
OMS-ROC-ens	**6.03**	**4.6**
OMS-ROC-ens-st	1.08	1.8
OMS-ROC-ens-Per	168.71	208.9

adaptively computing the RoCs of different forecasters, is able to gain excellent and reliable empirical performance in our setting. The informed update of the RoCs following concept drift detection makes our method, in addition to better predictive performance, computationally cheaper than the most competitive SoA methods, namely ADE-single and DEMSRC.

4.5 Explainable Online Model Performance and Selection

In this section, we show how OMS-ROC can be used to provide appropriate explanations for why a particular model or a set of models are selected at a particular time interval or point in time and why a particular forecast value can be expected to be output by this selection. To do so, we exploit the visualization of the time-adaptive clustering procedure involved in OMS-ROC to provide visual explanations for the model behavior and selection process.

The visualization of the RoCs helps in understanding the expertise of the candidate models. For example, Figure 4.4 shows that while SVR-lap-based models are experts in predicting accurately sharp patterns, SVR-rbf-based models are better in predicting valley-shaped patterns. "rbf" and "lap" denote radial basis function and laplacian kernels, respectively.

In addition, visualizing both the input patterns and their subsequent values help in predicting or anticipating the forecasting model's output which is considered to be one of the most important aspects of Explainability in ML [195]–[197]. Figure 4.5 demonstrates an example of a single model selection process using OMS-ROC-single by showing a comparison between the current input time series pattern $input_t^p$ (left part in black) with the closest cluster of RoCs of the selected model to perform the forecast. A clear similarity between both patterns can be observed. A clear similarity between the output forecast by the model and the values that are subsequent to the RoCs sequences (dotted line) is also observed. This also justifies our choice of the RoCs attribution to expert models based on the performance of these models on the subsequent values to the candidate sequences (i.e., the patterns of p lagged values for each data point) (See Equation 4.8). These similarities also highlight the importance of updating the RoCs by considering, in the first place, models' performance on the newly acquired pattern before the topological information (See Equation 4.11 and Section 4.4.2.3 for more details). In addition, it justifies at test time the choice of the model that has been proven to show some degree of competence in forecasting using similar patterns to the selected RoC cluster of patterns as input (See Equation 4.9).

The selection process of one or many models can also be understood by visualizing a comparison between the current time series pattern $input_t^p$ and the pre-computed RoCs

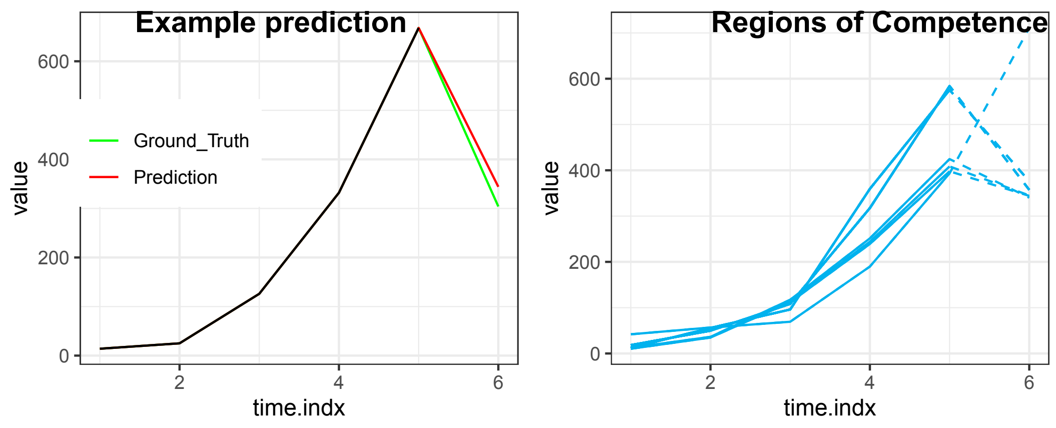


Figure 4.5: Comparison of the current input pattern to the closest RoC.

clusters of the candidate models. A more general overview of the RoCs and the selection process for ensemble pruning using OMS-ROC-ens is provided in Figure 4.6. We plot only the RoCs centers (i.e., cluster centers). The selected models are marked in red rectangles. The selection is made in favor of models whose patterns are similar to the current pattern, which encourages ensemble *accuracy*. In addition, the minimum distance is computed per model (See Equation 4.10) which enforces the selection of distinct models experts with different clusters of patterns which encourages ensemble *diversity*. Even though two Random Forests are selected RF1 and RF3, but they do have different hyper-parameters that are also different from the other RFs in the pool $\mathbb{P}$, which highlights the importance of the timely selection of the models' hyper-parameters when considering the same family of models. Regarding the forecast value, it can be seen that averaging the subsequent values (dotted lines in the three red marked rectangles) in each of the selected patterns results approximately in the true observed value that is subsequent to the current pattern, which explains the output forecast value by the ensemble.

4.6 Concluding Remarks

We have devised two main methods for online single model selection and ensemble pruning for the task of time series forecasting.

First, DEMSRC: a novel, practically useful dynamic ensemble members selection framework is developed. DEMSRC uses a two-stage selection procedure. On the one hand, DEMSRC enhances the ensemble's accuracy by performing an informed ensemble pruning adaptation at test time using candidate models' performance drift detection mechanism. On the other hand, DEMSRC enhances the ensemble's diversity using an online models' clustering procedure.

Second, OMS-ROC: a novel, practically useful online model selection method is presented. OMS-ROC uses the concept of adaptive Region of Competence (RoCs) and is suitable for online single model selection, as well as online ensemble pruning. These RoCs are computed using an adaptive clustering procedure of input time series subsequences. In addition, the RoCs are updated in an informed manner using concept drift detection in the computed RoCs structure. OMS-ROC can also be successfully used for providing useful explanations for

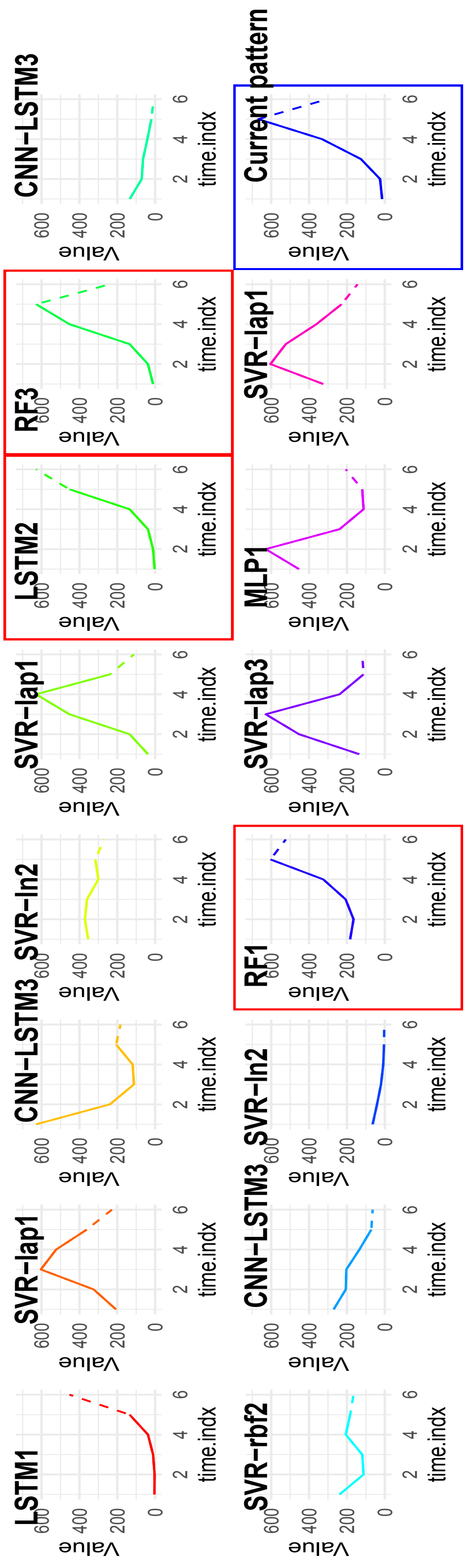


Figure 4.6: Example of ensemble pruning for Ab.Heartbeat data using OAMTS

model(s) selection and observed forecast values which help in simulating and understanding the behavior of forecasting models.

An exhaustive empirical evaluation, including 100 real-world time series datasets from various application domains and multiple algorithms comparisons from SoA under different perspectives, shows the advantages of both methods, DEMSRCand OMS-ROC, in terms of performance and scalability.

In the next chapter, we will introduce two methods specifically devised for online single DNN selection and an ensemble of DNNs pruning. We will explain the reasons behind this special attention to DNNs in this thesis. We will also provide the details of these methods.

5 Explainable Online Adaptive Deep Neural Network Selection

In this chapter, we specifically focus on Deep Neural Networks (DNNs) for three main reasons. First, DNNs are nowadays widely used in the context of time series forecasting. Second, DNNs are usually known to be extensive resource-consuming models making thus their online management, including online single DNN selection or ensembles of DNNs pruning, very challenging, especially in resource-constrained environments. Third, these models are also known to be very complex and usually referred to as black-box models, making thus their explainability very challenging but also highly required. Therefore, we will show in this chapter how the gradient-based training mechanism of DNNs, which offers rich information about the internal structure of the DNN, can be exploited for both DNN-based model selection and explainability.

5.1 Introduction

More recently, Deep Artificial Neural Networks (DNNs) have shown some improvements over previous shallow ANN architectures [112] and have been successfully applied to solve the time series forecasting task [112], [286]. This success is mainly explained by their ability to automatically learn new, complex, and enriched feature representations from input data [109]. Recurrent-based NNs such as Long Short-Term Memory Networks (LSTMs) [118], as well as Convolutional Neural Networks (CNNs) [114], have been widely used as State-of-the-Art NN methods in the context of forecasting [112], [113]. Many improvements over these networks' architectures have been proposed in the literature, ranging from optimizing the architecture structure to combining these networks together in one single forecasting task [113], [114]. However, DNNs typically have high variance, and low bias [113]. In complex and dynamic application environments, it is difficult for an individual deep learning model to maintain a high forecasting accuracy [287]. This is due to the fact that, as we mentioned before, different

DNNs have different areas of expertise or Regions of Competence in the time series and varying forecasting performance over time [9], [10], [22].

Two possible solutions to mitigate this issue are either to perform adequate and adaptive single DNN selection in real-time or to combine forecasts of different DNNs in an ensemble model.

The selection of the adequate DNN is also known as the selection of the adequate DNN architecture [116]. However, the search for an optimal network architecture for a given application is still an open research question [116]. This is even more challenging in the case of online forecasting, where the decision for adequate architecture has to be made in real-time. Due to their high training run-time and general resource consumption, it is usually impractical to search for adequate architecture at test time at each time instant or even in a periodic manner. Therefore, we focus in this chapter on approaching this problem by considering different candidate models of DNNs with different architectures (e.g., based on CNNs combined with other NNs models) and performing the online selection of the adequate architecture in real-time in an adaptive informed manner.

The first method is called in the following OS-PGSM: Online CNN-based models Selection using Performance Gradient-based Saliency Maps. We start by computing the Regions of Competence (RoCs) of candidate CNNs using saliency maps. Saliency maps are usually used to establish a relationship between the output and the input of a neural net given fixed weights. They are widely used in the context of computer vision with CNNs to create a class-specific heat map based on a particular input image and a chosen class of interest [288], also called class-activation maps. These maps are used for visualizing the regions of input that are "important" for predictions by the model and for understanding a model's predictions [203]. We suggest not only transferring class-activation maps from the context of classification to forecasting but also establishing a mapping between the input time series and the performance of candidate CNNs so that dynamic RoCs are computed for each single CNN. Opposingly to the aforementioned approaches, the RoCs are considered as dynamic since their size is automatically decided and changed over time by the saliency map depending on the input time series sequence and the corresponding CNN performance. The RoCs are computed using a time-sliding window over a validation set. At test time, we produce forecasts step by step. At each time step, the distance between the recently observed window of time series observations (i.e., p-lagged values used to compute the forecast) and the pre-computed RoCs is determined. The model corresponding to the RoC with the smallest distance is selected to output the forecast. Additionally, the pre-computed RoCs are adaptively updated in case a concept drift is detected in the time series by sliding the validation set to take into account the probable presence of new concepts in the data when computing RoCs.

The second method is an extension of the first method to an ensemble of DNNs pruning. We introduced some modifications to the way of computing the DNNs' RoCs, and we provided some theoretical insights guiding the construction of the ensemble and helping in setting up its final size, i.e., number of its composing models. More precisely, we propose a two-stage online pruning procedure for an ensemble of DNNs for time series forecasting. Since, as we discussed Section 2.6.2, the performance of the ensemble depends largely on the performance of its members and on the *diversity* amongst them [19], [29], [30], [156], our pruning procedure is devised to take into account these two aspects. This is achieved by comparing the computed RoCs of the DNNs to each other and to the most recent time series sequence pattern of p observations. The RoCs are computed using saliency maps, derived similarly to the way used in OS-PGSM by establishing a mapping between the input time series and the performance of the DNNs. On the one hand, *diversity* is encouraged using clustering of the corresponding DNNs RoCs, thus promoting the selection of diverse patterns by considering only cluster representatives. On the other hand, the second selection stage takes into account the *accuracy* of the cluster representatives by considering their closeness to the most recently acquired time series pattern. The distance between the current pattern and the most distant RoC from the selected models in the second stage sets its boundary under a form of a diameter. By setting up the number of clusters from the first stage, a bound for this diameter is derived, which automatically determines the final number of models that should be selected so that the expected ensemble error over the most recent time series window is reduced. At test time, we produce forecasts step by step. In the first time step, we perform the pruning as explained above. Afterward, at each time step, we compute the minimum distance between the RoCs of the candidate DNNs and the current time series pattern, i.e., the most recently observed window of time series observations. We keep monitoring the deviation of this minimum distance from the initial computed minimum. If this deviation diverges significantly over time, a drift in the dependency structure between the candidate DNNs and the target time series is assumed to take place (i.e., similar to the way the candidate models' dependence structure to the time series is monitored in DEMSRCin Chapter 4) and the ensemble pruning is updated by recomputing the models' selection using the most recently acquired time series pattern following the same methodology. The deviation is tested incrementally using the well-known Hoeffding Bound [262]. Another drift-detection mechanism is employed to test the presence of drifts in the time series values. The occurrence of such drift triggers the update of both RoCs and the pruning strategy. This method is denoted in the following OEP-ROC: Online Ensemble Pruning using performance saliency maps-based Regions-Of-Competence.

Since DNNs selection is based on the pre-computed saliency maps-based RoCs, suitable explanations of both single DNN selection, as well as an ensemble of DNNs pruning are provided. In addition, we explain their resulting performance.

We further conduct a comprehensive empirical analysis to validate both OS-PGSM and OEP-ROC using the same 100 real-world time series datasets used to validate DEMSRC and OMS-ROC. The obtained results demonstrate that both methods achieve excellent or on-par results in comparison to State-of-the-Art (SoA) approaches for DNNs selection and an ensemble of DNNs pruning and learning as well as to several baselines for time series forecasting.

5.2 Related Works

In Section 4.2.1.1, we have presented and discussed the related works on online model selection and ensemble pruning for time series forecasting. In this section, we discuss related works to the recent developments in deep learning for time series forecasting and the use of saliency maps to promote explainability in the deep learning context.

5.2.1 On the Recent Developments in Deep Learning for Time Series Forecasting

Over recent years, deep learning models have been successfully applied to a wide variety of learning tasks, including time series forecasting [114], [115], [289]. Currently, Recurrent Neural Networks (RNNs), and particularly Long-Short Term Memory (LSTM) nets, are considered to be State-of-the-Art in time series forecasting [114], [115], [290]. Thanks to their design based on recurrent connections (See Section 2.4.5.5), these networks have the ability to learn from the entire history of previous time series values. Another alternative for the use of DNNs in the forecasting task is to employ a Convolutional Neural Network (CNN) with multiple layers of dilated convolutions [291]. The dilated convolutions are based on filters allowing to skip certain elements and others representing certain repeating patterns in the input series. The layered structure of CNNs enables them to work well in noisy series, by removing the noise within each subsequent layer and extracting only meaningful patterns, performing thus similarly to neural networks which use the wavelet transform on input time series [290]. This also allows for the receptive field of the network to expand exponentially, thereby making the network, similar to RNNs, access a wide range of historical data. Due to the convolutional structure of the network, CNNs have fewer trainable weights than recurrent networks, allowing them thus to be much more efficient in training and predicting [290].

Some works have focused on improving CNN-based architectures by combining a CNN and an LSTM in one single model to take advantage of the ability of LSTMs to cope with long temporal correlations [114], [115]. In [292], the authors propose an undecimated convolutional network for time series forecasting using the undecimated wavelet transform. An autoregressive weighting schema for forecasting financial time series is presented in [293] where the weights are learned through a CNN. However, convolutional architectures in literature are much more commonly applied to time series classification problems compared to forecasting [120], [290].

The aforementioned works have focused on searching the most suitable network architecture for a well-defined application. At test time, the architecture and the learned weights are kept fixed and used to produce forecasts. However, as we mentioned before, to cope with the time-evolving nature of time series data, the forecasting schema has to be designed in a dynamic adaptive manner. Since the same model can't be guaranteed to hold the same performance over time [9], [10], [22], online adequate model selection is required. This is usually hard to achieve with DNNs in general since the architecture tuning and the re-training of such models are intensively time-consuming operations [114], [290]. We suggest mitigating this problem by training different CNN-based models with various architectures offline and deciding on the online selection of the adequate network at each time instant at test time. The selection is achieved using a computation of the so-called RoCs using saliency maps.

5.2.2 Saliency Maps for DNNs

Saliency maps (i.e., known in the context of classification as attribution heat maps or Class Activation Maps (CAMs)) [288] were originally designed for computer vision classification tasks to visualize which part of an image is of high relevance to the network to make its decision [203], [288]. They are considered as tools for better understanding a model's behavior, e.g., by providing insights into model failure modes [288]. Many saliency map generation methods are post-hoc methods in the sense that they are applied to an already-trained model. This is distinct from trainable attention, which involves learning how to produce attention maps during training by learning particular parameters.

CAMs use feature maps produced by the last convolutional layer of a CNN. This is motivated by the fact that the last convolution layer is expected to contain both high-level semantic and detailed spatial information [203]. More recently, CAMs have been applied in the context of time series classification to explain which features and which joint contribution of all the features during which time interval are responsible for a given time series class [218]. However, to the best of our knowledge, our method OS-PGSM is the first work to apply saliency maps for online CNN-based model selection for time series forecasting. The generation of these maps is performed in an informed, adaptive fashion. In addition, they are exploited to provide explanations for particular timely model selections.

5.3 Online Deep Neural Network Selection using Adaptive Saliency Maps

This section introduces OS-PGSM and its main stages:

- In the first stage, we train different candidate CNN-based models with various architectures offline.
- The second stage consists of determining the RoCs of these models using sliding windows over a validation set. The RoCs are computed using a modified version of CAMs, in the sense that instead of using these maps as a class attribution method, we employ them to establish a relation between a relative "good" performance of a given candidate CNN and a particular pattern within the input time-sliding window sequences. We base our method on a gradient-based technique for generating saliency maps called Grad-CAM [203]. We call our modified version in the following "Performance Gradient-based Saliency Maps (PGSMs)".
- In the third stage, in order to produce a forecast at a given time instant t_f, the distance of the current input time sequence (i.e., time series observations from $t_f - p$ to $t_f - 1$, $t_f > p$) to the computed RoCs of each model is measured. The model corresponding to the RoC with the smallest distance is selected to forecast the time series value at t_f. A concept drift detection mechanism in the time series is employed at test time (i.e., during forecasting). Once a drift is detected, an alarm is triggered to update the validation set by taking into account the new observed time series values and subsequently updating the RoCs.

The PGSMs can be used to provide suitable explanations for the reason behind selecting a specific model at a certain time interval or instant. Practical examples of these explanations are shown with details in Section 5.5.

5.3.1 Preliminaries

We keep the same notations as in the previous chapter. However, the pool of candidate models $\mathbb{P}$ contains a set of trained CNN-based models $f_i, i \in [1, M]$. Similarly, let $\hat{x}_{t+h} = (\hat{x}^{f_1}_{t+h}, \hat{x}^{f_2}_{t+h}, \cdots, \hat{x}^{f_M}_{t+h})$ be the vector of forecast values of the time series X recorded until time t at time instant $t + h, h \geq 1$ (i.e. x_{t+h}) by each of the models in $\mathbb{P}$.

We divide the time series sequence $X_{1:t}$ into $X^{train}_{\omega} = \{x_1, x_2, \cdots, x_{t-\omega}\}$ and $X^{val}_{\omega} = \{x_{t-\omega+1}, x_{t-\omega+2}, \cdots, x_t\}$, with ω a provided window size.

X^{train}_{ω} is used for training the models in $\mathbb{P}$ and X^{val}_{ω} is used to compute the RoCs using the PGSMs since to measure models' performance, both true and predicted values of the time series are required. The RoCs for each model $f_i, i \in [1, M]$ are obtained by performing time-sliding window operations of size $n_{\omega}, n_{\omega} < \omega$ over X^{val}_{ω} either by one step or by z steps.

5.3.2 Candidate CNN Architectures

The candidate models are CNN-based models that share more or less the same basic types of layers. The common basic structure consists of a sequence of 1D-convolutional layers with different filter and kernel sizes, followed by a batch normalization layer, in some cases an LSTM layer, and an output layer of one neuron. The different architectures are obtained by varying the number of the convolutional layers and their corresponding parameters (i.e., the size of filters and kernels) and, in some cases adding or removing another neural network type to the last convolutional layer, like an LSTM layer. To obtain further architectures variations, the number of units in the LSTM layer is also varied.

5.3.3 Online Model Selection

5.3.3.1 Performance Gradient-based Saliency Maps

The PGSMs are inspired by the class activation saliency maps, more specifically, Grad-CAM [203]. We choose Grad-CAM since it is considered one of the most successful methods for generating saliency maps [208], [218]. In fact, this method 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 [208]. However, instead of using these maps to derive the importance of certain features for a given class, we use them to map the performance of a given forecasting model to a specific time interval. The performance of each model $f_i, i \in [1, M]$ is evaluated using an error-related measure, namely the Mean Squared Error, ϵ^i_j on $X^{val,j}_{n_{\omega}}$: the j^{th} time interval window of X^{val}_{ω} of size n_{ω}. Assume that it starts at the instant $t = t_j$, then ϵ^i_j is given by:

$$\epsilon^i_j = \frac{1}{n_{\omega}} \sum_{l=t_j}^{t_j+n_{\omega}-1} (x_l - \hat{x}^{f_i}_l)^2 \tag{5.1}$$

Our goal is to estimate the importance of each value in $X^{val,j}_{n_{\omega}}$ to the measured error ϵ^i_j of f_i. This can be interpreted similarly to Grad-CAM as exploiting the spatial information that is preserved through the convolutional layers in order to understand which parts of an input image are important for a classification decision. However, we are focused here on the temporal information explaining certain behavior/performance of f_i. To do so, the last layer, which has produced the last feature maps fea_{maps} is considered. For each activation unit u at each generic feature map A, an importance weight w^{ϵ} associated with ϵ_j^i, is obtained. This is done by computing the gradient of the ϵ_j^i with respect to A. Subsequently, a global average over all the units in A is computed:

$$\alpha^{\epsilon} = \frac{1}{U} \sum_{u} \frac{\partial \epsilon_j^i}{\partial A_u} \tag{5.2}$$

where U is the total number of units in A. We use α^{ϵ} to compute a weighted combination between all the feature maps for a given measured value of the error ϵ_j^i. Since we are mainly interested in highlighting temporal features contributing most to ϵ_j^i, a ReLU is used to remove all the negative contributions by:

$$L_j^i = ReLU(\sum_{fea_{maps}} \alpha^{\epsilon} A) \tag{5.3}$$

$L_j^i \in \mathbb{R}^U$ is used to find the regions in $X_{n_\omega}^{val,j}$ that have mainly contributed to ϵ_j^i of the network f_i. Note that the candidates are designed such that $U < n_\omega$.

5.3.3.2 RoCs Computation

Our goal is to determine the Region of Competence of each model on X_ω^{val}. However, one single evaluation of the models on X_ω^{val} obviously leads to just one best model. Therefore, we need to split X_ω^{val} into equally sized time intervals of size n_ω, so that different evaluations of the candidate models are performed and different rankings are derived. To increase this number of evaluations, the intervals of size n_ω can be obtained using a time-sliding window approach over X_ω^{val} where the sliding operations are performed each z-steps. The lower z, the higher the number of evaluations is. If z is set to 1, the sliding window approach is performed in a step-wise manner. After evaluating the models on each of the $X_{n_\omega}^{val,j}$, the RoC of the model with the lowest error is computed using L_j^i of the PGSMs, where i the index of the candidate model f_i satisfying:

$$f_i = \underset{z \in \{1, \cdots, M\}}{\text{argmin}} \ \epsilon_j^z \tag{5.4}$$

To obtain one continuous region RoC R^i within the time series sequence $X_{n_\omega}^{val,j}$, a smoothing operation is applied to L_j^i. This is achieved by normalizing L_j^i values between 0 and 1 and applying a threshold $\tau = 0.5$ to filter out smaller values (i.e., these values are set to 0). Further smoothing using a moving-average operation of size 3 is applied where each point is compared to the previous and the subsequent value. Whenever R^i of model f_i is computed, it is added to a corresponding RoC^i buffer which includes all collected RoCs for the model f_i (i.e. since f_i can be the best performing model on different $X_{n_\omega}^{val,j}$).

5.3.3.3 Online Forecasting

For forecasting the value of X at $t+h$ (assume $h=1$ for simplicity), the candidate CNNs are devised such that they use the same p-lagged values of the time series as input, $input_t^p = \{x_{t-p+1}, \cdots, x_t\}$, $(t \geq p)$. To perform the selection, the distance of the input pattern $input_t^p$ to the RoCs for each model in $\mathbb{P}$ is measured. The RoCs of a given model $f_i, i \in [1, M]$ are already collected in $RoC^i = \{R_1^i, R_2^i, \cdots, R_{M^i}^i\}$, where M^i is the total number of regions of competence that have been determined by the PGSMs. Since the length of each RoC can be different from p (i.e. length of $input_t^p$), Dynamic Time Wrapping (DTW) [77] (See Section 2.1.5) is used to measure the similarity between $input_t^p$ and each $R_z^i, z \in [1, M^j]$ within each $RoC^i, i \in [1, M]$. The model f_b satisfying :

$$f_b = \underset{\substack{i \in [1,M];\\ z \in [1,M^i]}}{\operatorname{argmin}} DTW(R_z^i, input_t^p) \tag{5.5}$$

is selected to forecast $t+1$. The following steps are forecasted using the same principle by moving $input_t^p$ by one value.

5.3.3.4 RoCs Update

As explained above, the RoCs are computed offline using the validation set X_ω^{val}. However, due to the dynamic behavior of time series, The RoCs have to be updated to take into account the possible presence of new patterns after the occurrence of significant changes framed under the so-called concept drifts and also to gain knowledge of which models are more adequate to handle these patterns if they ever reoccur again (i.e., note that the already computed RoCs are preserved and enriched with the new ones). Once a drift in the time series is detected, an alarm is triggered to update the ROCs by sliding X_ω^{val} to include the new recent observations. The detection of concept drifts is performed by monitoring the deviation Δm_{t_h} in the mean of the time series, similar to the different drift types we detected in the previous methods presented in this thesis, such as the drifts in the forecasting error in OAMTSor in the dependence structure between the candidate model in DEMSRC:

$$\Delta m_{t_h} = \mathbb{E}(X_{1:t_h}) - \delta \tag{5.6}$$

with $\delta = \mathbb{E}(X_{1:t}), t \leq t_h$, the initial computed mean of X up to time t, a drift is assumed to take place at t_h if the true mean of Δm_{t_h} diverges in a significant way from 0. We propose to detect the validity of this using the Hoeffding-Bound [262], which states that after ω independent observations of a real-value random variable with range R, its true mean has not diverged if the sample mean is contained within $\pm\zeta$:

$$\zeta = \sqrt{\frac{R^2 \ln(1/\mu)}{2\omega}} \tag{5.7}$$

with a probability of $1-\mu$ (a user-defined hyperparameter). Once the bound is violated, an alarm is triggered, and the reference mean δ is reset by setting $t = t_h$. This checking procedure is continuously applied online at forecasting time. All the steps of OS-PGSM are summarized in Algorithm 4.

In addition to the pseudo-code for our OS-PGSM algorithm 4, we also show how it works in Figure 5.1. There, we show an example time series X_ω^{val} with the sliding window approach of step size z. For each window of size n_ω, we compare the prediction of all models $f_j \in \mathbb{P}$ from the model pool and choose the best performing one. Then, for this model, we compute the region of competence (shown in red) and add it to the corresponding RoC^j buffer.

```
Algorithm 2: OS-PGSM
   Data: Parameters: size of the validation set: $\omega$; size of time windows within the
         validation set: $n_\omega$; CNNs Pool: $\mathbb{P}$.
 1 Models Training and RoCs Computation: ;
 2 Train each $f_i \in \mathbb{P}, i \in [1, M]$ on $X_\omega^{train}$. ;
 3 Initialize RoC buffers $RoC^i$ for each $f_i, i \in [1, M]$ ;
 4 for each $X_{n_\omega}^{val,j} \in X_\omega^{val}$ do
 5     Determine the best performing $f_i$.;
 6     Compute the corresponding $R_j^i$ using PGSMs (i.e. $L_j^i$ Equation 5.3 );
 7     Add $R_j^i$ to the corresponding buffer $RoC^i$ ;
 8 end
 9 Online Forecasting: Forecasting next $H$ values:;
10 predict $x_{t+1}$ by the model $f_b$ selected using Equation 5.5;
11 for $j \in [2, H]$ do
12     if an alarm is triggered (concept drift detected) then
13         Update $X_\omega^{val} = \{x_{t-\omega+j}, x_{t-\omega+2}, \cdots, x_{t+j-1}\}$ ;
14         Recompute and add new RoCs (steps:4-7);
15     end
16     Predict $x_{t+j}$ by the model $f_b$ selected using Equation 5.5 ;
17 end
```

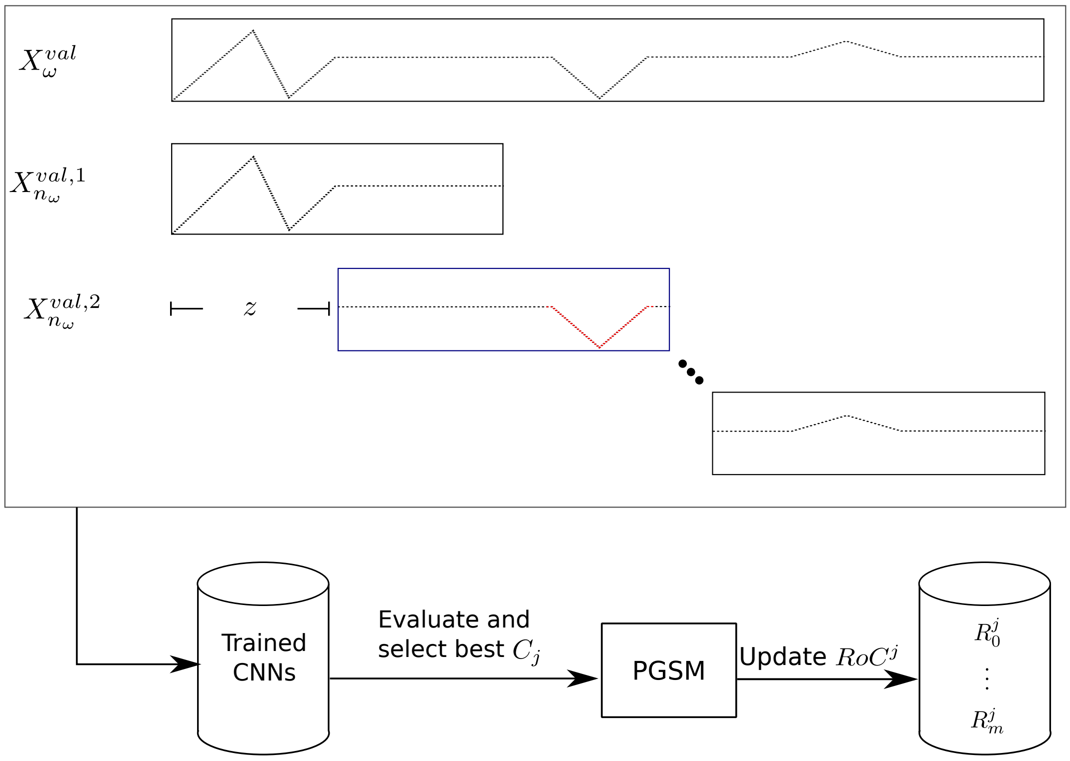


Figure 5.1: Schematic visualization of OS-PGSM extracting RoCs from the validation time series.

5.3.4 Empirical 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 (SoA) 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 X_ω^{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? and what is the computational advantage of the *drift-aware* adaption of the models' RoCs?

5.3.4.1 Experimental Setup

The methods used in the experiments were evaluated using the root mean squared error (RMSE). The used time series was split using 50% for training (X_ω^{train}), and 25% for validation (X_ω^{val}) and 25% for testing. We use the same 100 real-world time series as in the previous chapter and presented in the appendix Table A.3. In addition, a full list of these datasets, together with a description, is given in the code repository[1].

OS-PGSM Setup and Baselines We construct a pool $\mathbb{P}$ of CNN-based candidate models using different parameter settings (e.g., number of filters varies in $\{32, 64, 128\}$, kernel size varies in $\{1, 3\}$) like explained in Section 5.3.2. By combining these different parameters and adding or removing LSTM layers, 12 different CNNs with different architectures are created. OS-PGSM also has a number of hyperparameters that are summarized in Table 5.1. In our

Table 5.1: Hyperparameters of OS-PGSM and their values for the experiments.

Parameter	Description	Value
p	number of lagged values (size of the input to CNNs $input_t^p$)	5
ω	size of validation set	25% of the dataset length
n_ω	size of time windows within the validation set	5
z	number of time steps with which time windows within the validation set are slided	1
μ	Hoeffding-Bound parameter	0.05

experiments, p is set equal to n_ω, and the RoCs (after computation and smoothing) result in an even smaller size than p. However, this is not problematic since we are interested in extracting distinctive patterns that are responsible for the good performance of a given candidate. The difference in lengths of the RoC and the input sequence (p) are handled by the DTW distance.

We compare OS-PGSM against the following approaches, which include SoA methods for forecasting and model selection methods devised for the task of forecasting. Some of them operate in an online fashion. First, we compare against ARIMA [61] and Exponential Smoothing ETS [97]. Next, we add the two best-performing candidate models, denoted as CNN [15] and CNN-LSTM [112]. Additionally, a simple LSTM is added for comparison [118]. 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 [27]. We compare also OS-PGSM against KNN-RoC(1) [22], ADE-single [9]. For further details about these methods, see Section 4.4.3.1. As Stacking, we denote a method where a meta-learner (Random

[1] `https://github.com/MatthiasJakobs/os-pgsm/tree/ecml2021`

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 [33]. Finally, we compare ourselves against Adaptive Mixture [294], which consists of some experts (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. Note we input the same pool $\mathbb{P}$ of candidate models as OS-PGSM to KNN-RoC(1), ADE-single, Stacking, and Adaptive Mixture methods.

We also compare OS-PGSM against the recently presented Deep Learning models for time series forecasting in the literature. Many of these models are transferred from computer vision or speech recognition domains and adapted to time series. We include these methods as they are also presented as bench-marking methods in the Monash repository [1] and almost all of our evaluation datasets in Table A.3 are extracted from the Monash repository. These models include:

- DeepAR [295]: Probabilistic forecasting with Autoregressive Recurrent networks [295], is based on an autoregressive RNN architecture. Technically, any network that used previous data from a sequence to predict a future value in that sequence can be considered autoregressive. The difference is that the network processes all the historical time series data without explicit setting of the number of lags by the user. Hence, the DeepAR algorithm trains an RNN that learns an approximation of the time series generating process and uses it to predict how the time series evolves in the future. The model can also handle time series that are associated with a vector of time-invariant categorical features and a vector of time-variant features, denoted exogenous time series. the DeepAR model is trained by randomly sampling training examples from the time series. Each training example is composed of a window of lagged values joined to a prediction window with a fixed predefined length.

- N-BEATS [296]: consists of a DNN architecture based on backward and forward residual links and a very deep stack of fully-connected layers.

- N-BEATS-Ensemble [296]: The ensemble version of N-BEATS. The ensemble is built using several sources of *diversity*. First, the ensemble members are fit on three different metrics: sMAPE, MASE, and MAPE, a version of sMAPE that has only the ground truth value in the denominator. Second, individual models are trained on input windows of different lengths. Thus, the overall ensemble exhibits a multi-scale aspect. Finally, a bagging procedure [31] is applied, and models that are trained with different random initializations are added.

- WaveNet [290]: An adaptation of the recent Deep Convolutional WaveNet architecture to time series forecasting is proposed. The network contains stacks of dilated convolutions that allow it to access a broad range of history when forecasting, a ReLU activation function, and conditioning that is performed by applying multiple convolutional filters in parallel to separate time series windows which allows for the fast processing of data, and the exploitation of the correlation structure in the case of multivariate time series.

- Transformer [297]: consists of an adaptation of the sequence transduction model to time series forecasting. The model is based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.

We have also tried to adapt the popular ROCKET framework [239], [240] widely and successfully used for time series classification to forecasting, but the first results on some datasets from Table A.3 were not promising (i.e., performs worse than many poor baselines). ROCKET transforms the time series by first convolving each series with a huge number of random convolutional kernels. The random convolutional kernels have random lengths, weights, bias, dilation, and padding. Then, it separately applies global max pooling and PPV "Proportion of Positive Values" pooling to the convolutional output to produce two features per kernel, one feature for each pooling method. The poor performance in the context of forecasting can be explained by the large number of produced features compared to the size of the time series datasets. In addition, capturing the dependence structure between subsequent observations in a time series which is required for forecasting, can be lost along the huge number of transformations. In addition, the most critical element of ROCKET that contributes to its high accuracy is the PPVs that are devised to indicate how to weight the prevalence of a pattern captured by the kernel, which is more suited for classification. Therefore, future investigations and adjustments of the method are required to be applied for forecasting.

We also compare OS-PGSM with some variants of itself. Note that OS-PGSM uses the Hoeffding-based drift detection mechanism to update the RoCs.

- OS-PGSM-Int: Same as our method, but the time windows of size n_ω are slided with step size $z = n_\omega$.
- OS-PGSM-Euc: Instead of using DTW as a distance measure, we use Euclidean distance. However, values in the RoC corresponding to 0 are not taken into consideration in the sequence of the p-lagged values ($input_t^p$).
- OS-PGSM-Int-Euc: It is a combination of OS-PGSM-Int and OS-PGSM-Euc.
- OS-PGSM-St: Same as our method, but the RoCs are not updated using the drift detection mechanism. The RoCs are computed and stored offline. Only the selection takes place online.
- OS-PGSM-Per: Same as our method, but the RoCs are updated periodically in a blind manner (i.e., without taking into account the occurrence of concept drifts) with the periodicity of each upcoming 10% data points.

5.3.4.2 Comparing OS-PGSM to the State-of-the-Art

Table 5.2 presents the average ranks and their deviation for all the methods. For the paired comparison, we compare our method OAMTS against each of the other methods. We counted the wins and losses for each dataset using the RMSE scores. We use the non-parametric Wilcoxon Signed Rank test [270] to compute significant wins and losses, which are presented in parenthesis (significance level 0.05). In the results in Table 5.2, OS-PGSM outperforms the baseline methods in terms of wins/losses in the pairwise comparison. The online model

Table 5.2: Comparison of OS-PGSM to different SoA methods for 100 time series. The rank column presents the average rank and its standard deviation across different time series. A rank of 1 means the model was the best performing on all the time series datasets.

Method	Our Method Losses	Wins	Avg. Rank
ARIMA	5(4)	95(93)	15.90 ± 4.35
ETS	3(3)	97(96)	18.04 ± 4.26
LSTM	15(8)	85(83)	12.97 ± 3.60
CNN	16(6)	84(80)	14.00 ± 3.60
CNN-LSTM	21(6)	79(71)	12.30 ± 3.59
DeepAR	21(6)	79(70)	12.90 ± 4.65
N-BEATS	3(3)	97(96)	14.04 ± 4.86
N-BEATS-Ensemble	15(8)	85(83)	13.97 ± 3.80
WaveNet	16(6)	84(80)	16.00 ± 3.61
Transformer	21(6)	79(71)	15.30 ± 5.59
Validation	18(12)	82(72)	8.84 ± 3.18
KNN-RoC	20(16)	80(73)	7.96 ± 3.45
ADE-Single	30(19)	70(61)	4.78 ± 3.90
Stacking	9(9)	91(80)	13.11 ± 4.12
Adaptive Mixture	15(8)	85(84)	11.95 ± 4.78
OS-PGSM-Int	38(7)	62(54)	3.62 ± 2.80
OS-PGSM-Euc	34(10)	66(56)	3.86 ± 3.10
OS-PGSM-Int-Euc	31(9)	69(66)	3.98 ± 3.15
OS-PGSM-St	47(10)	53(44)	3.01 ± 2.65
OS-PGSM-Per	43(7)	57(46)	3.19 ± 3.44
OS-PGSM	-	-	$\mathbf{2.75 \pm 2.63}$

selection approaches, e.g., KNN-RoC(1), ADE-Single, and Adaptive Mixture show inferior performance compared to OS-PGSM. ARIMA, ETS, LSTM, and CNN, SoA methods for forecasting are considerably worse in the average rank compared to OS-PGSM. CNN-LSTM shows slightly better performance but is still worse than OS-PGSM. Comparing OS-PGSM to State-of-the-Art Deep Neural Networks [1] such as N-BEATS, WaveNet, and Transformer shows that opting for deep architecture is not always beneficial. This can be explained by the fact that these models are prone to overfitting, especially with relatively small to medium-sized time series datasets. These networks are transferred from domains where usually huge amounts of training data are available. In addition, capturing the dependence structure between time series observations should be taken into account in the design of these networks. This explains why DeepAR has a relatively better performance.

The most competitive SoA approach to OS-PGSM is ADE-Single. Nevertheless, it has a higher average rank and a lower performance than all the variants of OS-PGSM.

To further investigate the differences in the average ranks, we use the post-hoc Bonferroni-Dunn test [285] to compute critical differences. We present the critical differences between the methods relative to each other in figure 5.2. We note critical differences between OS-PGSM and most of the other methods, with the exception of ADE-Single. However, unlike OS-PGSM ADE-Single does not share critical differences with the other methods. These results address the research question **Q1**.

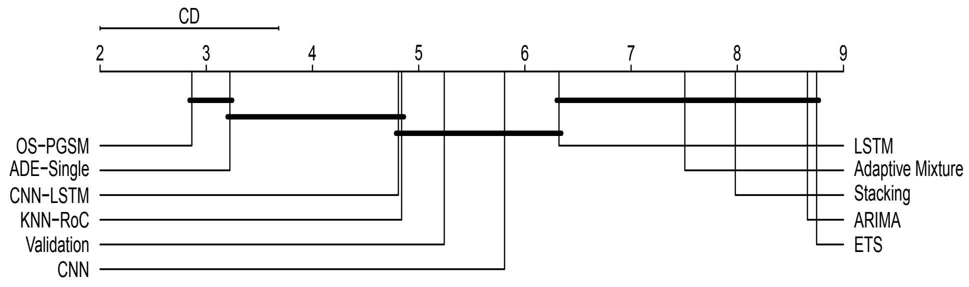


Figure 5.2: Critical difference diagram for the post-hoc Bonferroni-Dunn test, comparing OS-PGSM with the other baseline ensemble methods.

5.3.4.3 Comparing OS-PGSM to its Variants

Comparing OS-PGSM to its different variants, we see a clear advantage in using all the choices in our method. First, the DTW distance is better in sketching 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 $z = 1$, a higher number of windows of size n_ω are created, and as a result, a higher number of RoCs are computed (See Section 5.3.3.2). This contributes to creating richer information about the RoCs of different candidates, compared to setting $z = n_\omega$ in OS-PGSM-Int and OS-PGSM-Int-Euc. This answers the research question **Q2**.

5.3.4.4 Importance of the Drift-aware Adaptation

OS-PGSM-St is even better than OS-PGSM-Per, which shows that unnecessary updates are not always beneficial and additional computational efforts are invested without gaining advantages in terms of performance. Opposingly, OS-PGSM which relies on an informed adaption of the RoCs using concept drift detection, is better than 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 can be detected, and more probably, taking into account these newly appearing concepts is helpful for the selection of models since a knowledge of which models are more adequate to handle these patterns if they ever reoccur again is gained, and the old sets of RoCs are enriched. Figure 5.3 show an illustrative example of the RoCs of the candidate CNN C_7 before and after drift detection. New patterns are added as new RoCs to the old RoCs of C_7. This answers the research question **Q3**.

In the next experiment, we compare the run-time of OS-PGSM and its variants against the most competitive SoA method, ADE-Single, in Table 5.3. Note that the computation of the RoCs is done offline, and the reported run-time is calculated only for computing the time series predictions in real-time by searching the most similar RoC to the current input pattern

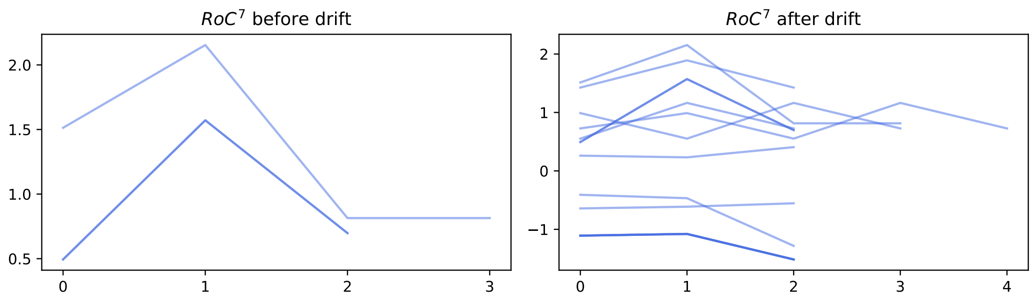


Figure 5.3: RoCs for the candidate model C_7 before and after drift detection.

Table 5.3: Empirical run-times comparison between different methods in Seconds.

Method	ADE-Single	OS-PGSM	OS-PGSM St	OS-PGSM Int	OS-PGSM Per
Avg. Run-time	157.87	9.45	2.15	0.95	160.15
±	58.40	19.35	5.80	3.65	205.25

$input_t^p$. OS-PGSM relies on informed updates (i.e., using drift detection). OS-PGSM-Per relies on a periodic update of the RoCs. The reported run-times for OS-PGSM and OS-PGSM-Per take into account the computation of the new RoCs. ADE-Single [10] relies on a periodic update of the meta-learning strategy behind the selection (same periodicity as OS-PGSM-Per). All the reported run-times 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 run-times than ADE-Single. OS-PGSM-Int is faster than OS-PGSM-St since fewer evaluation windows of size n_ω are created, and as a result, a lower number of RoCs is generated for distance comparisons. OS-PGSM has a lower run-time than OS-PGSM-Per. This is due to using drift detection to update the RoCs only when necessary. This results in faster predictions and less computational effort. The high deviation of the run-time of OS-PGSM is due to the different numbers of drifts per time series. This answers the research question**Q4**.

5.3.4.5 Final Remarks

The empirical results indicate that OS-PGSM has performance advantages compared to popular forecasting methods and the most recent SoA approaches for online forecasting model selection. We show that our method, using PGSMs for adaptively computing RoCs of different CNN-based forecasters, is able to gain excellent and reliable empirical performance in our setting. The informed update of the RoCs following concept drift detection makes our method, in addition to better predictive performance, computationally cheaper than the most competitive SoA, namely ADE-single. OS-PGSM can also successfully be used for providing useful explanations behind model selection which can be used to optimize our framework further.

5.4 Online Ensemble of Deep Neural Networks Pruning

OEP-ROC: Online Ensemble Pruning using Regions-of-Competence, is an online two-staged ensemble pruning framework based on clustering and ranking. It is developed to prune an ensemble of deep learning models for time series forecasting by taking into account both ensemble *accuracy* and *diversity* at each stage. Both stages are dependent on each other, and a bound for the optimal number of final models to be kept is derived. The pruning is not directly performed using the candidate deep models' outputs but is based on their pre-computed Regions-of-Competences (RoCs), which have been proven to be very useful for single online DNNs selection for time series forecasting, as we have shown in the previous Section. The pruning is also made in an adaptive informed manner following concept drift detection in time series and models' RoCs structure. More details about the method are provided in this Section. The RoCs are calculated using the PGSMs, i.e., Performance Gradient-based Saliency Maps, a method similar to the way presented in the previous Section for OS-PGSM.

5.4.1 Preliminaries

We keep the same notations used in the previous Section for the time series, the pool of models, and the forecasts. The ensemble model of the pool $\mathbb{P}$ is denoted by $\bar{f}_{\mathbb{P}}$. For simplicity, we set the weights to be equal, i.e., $w_i = \frac{1}{M} \forall i \in [1, M]$. As we mentioned in Section 2.6.2, the goal of dynamic online ensemble pruning is to identify the subset of models $\mathbb{S} \subset \mathbb{P}$ that should compose the ensemble at each time step $t + h$ such that the expected prediction error of the pruned ensemble is reduced compared to the full ensemble $\bar{f}_{\mathbb{P}}$ for each forecast.

The pruning is performed using the RoCs of the candidate DNNs computed by the PGSMs derived in Section 5.3.3.1. X^{train}_{ω} is used for training the models in $\mathbb{P}$ and X^{val}_{ω} is used to compute the RoCs using the PGSMs.

5.4.2 Base Learners

Similarly to the pool $\mathbb{P}$ of candidate CNNs considered in the previous Section for OS-PGSM, the ensemble base models in OEP-ROC are CNN-based models that share more or less the same basic types of layers. The common basic structure consists of a sequence of 1D-convolutional layers with a different number of filters, followed by a batch normalization layer, in some cases an LSTM layer, and an output layer of one neuron. The different architectures are obtained by varying the number of the convolutional layers and their corresponding parameters (i.e., the number of filters) and, in some cases adding or removing another neural network type to the last convolutional layer, e.g., an LSTM layer. To obtain further architectural variations, the number of units in the LSTM layer is also varied. The candidate CNNs are devised such that they use the same p-lagged values of the time series as input to forecast the subsequent value.

5.4.3 RoCs Computation

As we mentioned, the RoCs of the candidate DNNs are computed using the PGSMs introduced in Section 5.3.3.1. However, for this method, the performance (i.e., the forecasting error) of all the candidate models is measured, and the RoCs for each model are computed. This can be viewed as computing the most important time series interval responsible for a certain observed performance of each base model in the ensemble. In this way, a buffer RoC^i that contain all the pre-computed RoCs R^i_j on the different validation windows $X^{val,j}_{n_\omega}, j \in [1, Z]$ for each model $f_i, i \in [1, M]$ is created, i.e., $RoC^i = \{R^i_1, \cdots, R^i_Z\}$. It is important to note that in order to obtain one continuous region RoC R^i_j within the time series sequence $X^{val,j}_{n_\omega}$, a smoothing operation is applied to L_j^i (Equation 5.3). This is done by normalizing L_j^i values between 0 and 1 and applying a threshold $\tau = 0.1$ to filter out smaller values (i.e., these values are set to 0). In addition, a moving average is applied where each point is compared to the previous and the subsequent value. After the smoothing operation, some regions may become empty. This is mainly due to some low L_j^i values (i.e., low relevance of the time point) or high discontinuity.

The base models use the same p-lagged values of the time series as input, $input_{t+h-1}^p = X_{t+h-p:t+h-1} = \{x_{t+h-p}, \cdots, x_{t+h-1}\}$, $(t + h \geq p)$. In addition to the smoothing operation applied similarly to OS-PGSM, in order to constrain the RoCs lengths to be equal to p, we reject the smoothed RoCs with lengths different from p. At test time, in order to forecast the value of X at $t+h, h \geq 1$, the similarity between the input pattern $input_{t+h-1}^p$ and the RoCs for each model in $\mathbb{P}$ is computed. Euclidean distance D_E can now be used to measure the similarity between $input_{t+h-1}^p$ and each $R_j^i, \forall j \in [1, Z], \forall i \in [1, M]$ within each $RoC^i, \forall i \in [1, M]$ buffer. For each model $f_i, \forall i \in [1, M]$, the RoC $\mathcal{R}_i$ satisfying :

$$\mathcal{R}_i = \underset{R_j^i \in RoC^i}{\operatorname{argmin}} D_E(R_j^i, input_{t+h-1}^p) \tag{5.8}$$

is selected to represent f_i for the ensemble pruning at $t + h$.

5.4.4 Online Ensemble Pruning

The pruning decision is performed in a step-wise manner online at each time forecast $t+h, h \geq 1$. For simplicity of notation, we assume $h = 1$. The ensemble generalization error can be written as (See Section 2.6.1):

$$E_{\bar{f}} = \bar{E} - \bar{A} \tag{5.9}$$

The RoCs $\mathcal{R}_i, \forall i \in [1, M]$ of each candidate model indicate the degree of expertise of the corresponding model in forecasting given the most recent input time series sequence pattern that is acquired to forecast the value of X at $t + 1$, i.e., $input_t^p = X_{t-p+1:t}$ (See Equation 5.8). Since this selection is made based on the closeness of $\mathcal{R}_i$ to $input_t^p$, $\mathcal{R}_i$ can also be viewed as an estimate of $input_t^p$ by f_i. In other words, the prediction by f_i of the data points in $input_t^p$ are represented approximately by the data points $r_i^l \in \mathcal{R}_i, \forall l \in [1, p]$:

$$x_{t+l-p} \approx r_i^l, \forall l \in [1, p], t \geq p - 1 \tag{5.10}$$

The ensemble $\bar{f}$ of $\mathbb{P}$ can be expressed as:

$$\bar{f}_{\mathbb{P}}(x_{t+l-p}) = \frac{1}{M} \sum_{i=1}^{M} r_i^l = \bar{r}_l \tag{5.11}$$

The ensemble error on the pattern $input_t^p$ using RoCs approximation $\mathcal{R} = \{\mathcal{R}_1, \cdots, \mathcal{R}_M\}$ by the models in $\mathbb{P}$ can be then written as:

$$
\begin{aligned}
E_{\bar{f}}^{\mathcal{R}} &= \frac{1}{p}\sum_{l=1}^{p}(x_{t+l-p} - \bar{r}_l)^2 \\
&= \frac{1}{p}\sum_{l=1}^{p}(\frac{1}{M}\sum_{i=1}^{M}(x_{t+l-p} - r_i^l)^2) - \frac{1}{p}\sum_{l=1}^{p}(\frac{1}{M}\sum_{i=1}^{M}(\bar{r}_l - r_i^l)^2) \\
&= \bar{E}^{\mathcal{R}} - \bar{A}^{\mathcal{R}}
\end{aligned}
\tag{5.12}
$$

where $\bar{E}^{\mathcal{R}} = \frac{1}{p}\sum_{l=1}^{p}(\frac{1}{M}\sum_{i=1}^{M}(x_{t+l-p} - r_i^l)^2)$ and $\bar{A}^{\mathcal{R}} = \frac{1}{p}\sum_{l=1}^{p}(\frac{1}{M}\sum_{i=1}^{M}(\bar{r}_l - r_I^l)^2)$. It is very intuitive to see from Equation 5.12 that the selection should be made in favor of models whose RoCs are closer to the current pattern (i.e., considering the Euclidean distance) in order to minimize $\bar{E}^{\mathcal{R}}$ and diverse from each other in order to maximize $\bar{A}^{\mathcal{R}}$. To do so, we perform a two-staged pruning procedure.

In the first stage, we start by clustering the candidate models using Euclidean distance into K clusters and select only cluster representatives to take part in the second pruning stage. Models belonging to different clusters are expected to have a higher distance between them compared to models belonging to the same cluster. As a result, clusters' representatives have a higher average distance to the average pattern $\bar{\mathcal{R}}_p = \{\bar{r}_1, \cdots, \bar{r}_p\}$, which contributes to increasing $\bar{A}^{\mathcal{R}}$. In addition, models belonging to the same cluster have more or less the same distance to the current pattern $input_t^p$. Therefore, the average error induced by all the models is expected to be similar to the average 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, $\bar{E}^{\mathcal{R}}$ is approximately preserved. In this way, we promote diversity without increasing the averaged error, which reduces the expected ensemble error $E_{\bar{f}}^{\mathcal{R}}$ on $input_t^p$. The selection of a model $f_r, r \in [1, N_C]$ to be representative of a given cluster of models $\mathcal{C}_m, \forall m \in [1, K]$ where N_C is its size, is done based on its closeness to the current pattern $input_t^p$.

$$
f_r = \underset{\mathcal{R}_i \in \mathcal{C}_m}{\operatorname{argmin}}\, D_E(\mathcal{R}_i, input_{t+h-1}^p), \forall m \in [1, K] \tag{5.13}
$$

This helps to reduce the average error of the base models. The clustering of the RoCs is done using K-means [298] with Euclidean distance as a similarity measure.

In the second stage, a selection of the best-performing clusters' representatives, denoted here as ***top-K*** models, is computed using ranking. As the second stage is dedicated to reducing the averaged error $\bar{E}^{\mathcal{R}}$ while promoting the diversity even further, the ranking is computed using the distance of the candidate clusters' representatives to the current pattern $input_t^p$ and a bound for the radius γ of the p-sphere of center $input_t^p$ enclosing the ***top-K*** models that should be preserved in this stage, is derived:

$$
\frac{1}{2}\sqrt{\frac{\sum_{l=1}^{p}\sum_{i=1}^{K}(\bar{r}_l - r_i^l)^2)}{K}} \leq \gamma \leq \sqrt{\frac{\sum_{l=1}^{p}\sum_{i=1}^{K}(x_{t+l-p} - r_i^l)^2}{K}} \tag{5.14}
$$

where K is the number of clusters (i.e., number of selected models from the first stage). Note that the ensemble's error is always positive, meaning that the ambiguity can be considered as a lower bound of the averaged error of the base models. This applies to the ensemble of the K clusters' representatives $E_K^{\mathcal{R}}$. We have then $\bar{E}_K^{\mathcal{R}} \geq \bar{A}_K^{\mathcal{R}}$ which means that we always satisfy that the upper bound of Equation 5.14 is bigger than the lower bound of the same equation.

Proof 15 *To ensure a reduction of the averaged error, the error $\bar{E}^{\mathcal{R}}_{\textbf{top-}K}$ induced by the subset of the **top-K** models should be lower than $\bar{E}_K^{\mathcal{R}}$ induced by the subset of K of clusters' representatives.*

$$\bar{E}^{\mathcal{R}}_{\textbf{top-}K} \leq \bar{E}_K^{\mathcal{R}}$$

$$\frac{1}{p}\sum_{l=1}^{p}\frac{1}{\mid \textbf{top-}K \mid}\sum_{i=1}^{|\textbf{top-}K|}(x_{t+l-p} - r_i^l)^2 \leq \frac{1}{p}\sum_{l=1}^{p}\frac{1}{K}\sum_{i=1}^{K}(x_{t+l-p} - r_i^l)^2$$

*where $|\textbf{top-}K|$ is the cardinality of the **top-K** models. Since they are contained within the p-sphere of center $input_t^p$, we have $\sum_{l=1}^{p}(x_{t+l-p} - r_i^l)^2 \leq \gamma^2, \forall i \in [1, |\textbf{top-}K|]$. Therefore, every single model f_i contained within the sphere has at the most a distance of γ to $input_t^p$. As a result:*

$$\frac{\gamma^2}{p} \leq \frac{1}{p}\frac{1}{K}\sum_{l=1}^{p}\sum_{i=1}^{K}(x_{t+l-p} - r_i^l)^2$$

$$\gamma^2 \leq \frac{1}{K}\sum_{l=1}^{p}\sum_{i=1}^{K}(x_{t+l-p} - r_i^l)^2$$

$$\delta \leq \sqrt{\frac{\sum_{l=1}^{p}\sum_{i=1}^{K}(x_{t+l-p} - r_i^l)^2}{K}}$$

In addition, we want to ensure that the second stage either preserves the promoted diversity from the clustering stage or enhances it even further by preserving or increasing the ambiguity:

$$\bar{A}^{\mathcal{R}}_{\textbf{top-}K} \geq \bar{A}_K^{\mathcal{R}}$$

$$\frac{1}{p}\sum_{l=1}^{p}\frac{1}{\mid \textbf{top-}K \mid}\sum_{i=1}^{|\textbf{top-}K|}(\bar{r}_l - r_i^l)^2) \geq \frac{1}{p}\sum_{l=1}^{p}\frac{1}{K}\sum_{i=1}^{K}(\bar{r}_l - r_i^l)^2)$$

*The average pattern $\overline{\mathcal{R}}_L$ of the **top-K** models is contained within the p-sphere of center $input_t^p$ and the distance between $\mathcal{R}_i$ and $\bar{\mathcal{R}}_p$ can be then at most 2γ for all the models f_i with $i \in [1, |\textbf{top-}K|]$. Therefore:*

$$\frac{4\gamma^2}{p} \geq \frac{1}{p}\sum_{l=1}^{p}\frac{1}{K}\sum_{i=1}^{K}(\bar{r}_l - r_i^l)^2)$$

$$\gamma \geq \frac{1}{2}\sqrt{\frac{1}{K}\sum_{l=1}^{p}\sum_{i=1}^{K}(\bar{r}_l - r_i^l)^2)}$$

In practice, in order to identify the ***top-K*** models for a fixed number of clusters K, we start by clustering the M models in P and we compute the clusters' representatives (See Equation5.13). Then, we calculate the distance of their RoCs to the current pattern $input_t^p$. We set γ simply to the upper bound of Equation 5.14 (i.e., in this case, the lower bound is naturally verified) and models whose distance is lower or equal to γ are selected as the ***top-K*** models to compose the ensemble that forecasts the value of X at $t+1$. The upper bound of Equation 5.14 can then be interpreted as promoting the selection inside the p-sphere close to the true recent pattern $input_t^p$ which reduces the averaged error while the lower bound can be viewed as a regularization parameter that disallows radius values that would result in a clustering of the base models very closely around $input_t^p$ which decreases the ambiguity.

The models selected to compose the ensemble at $t+1$ (i.e., ***top-K***) are assumed to remain valid for the following time instants $t+h$ with $h > 1$ 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 the pruning decision (i.e.***top-K*** models) is triggered. More details are provided in the following Subsection.

5.4.5 Drift-aware Pruning Update

In order to update the pruning decision at test time in an informed manner, two drift detection mechanisms are deployed.

5.4.5.1 Concept Drift in Time Series

Concept drift in the time series data is detected by monitoring the deviation in the mean of the time series over time, similarly to the way described in Section 5.3.3.4. The detection of a drift in the time series triggers the update of the RoCs. The update of the RoCs triggers, in its turn, the update of the pruning decision. Since new RoCs can be added to the different RoCs buffers, changes in the distances of these RoCs to the most recent pattern $input_{t+h-1}^p$ are expected. As a result, new RoCs representatives of the models can appear (See Equation 5.8). Therefore, a re-clustering of the models and a re-selection of the ***top-K*** models become then necessary.

5.4.5.2 Concept Drift in Models' Performance

Base models' performance is reflected using the distance of their representative RoCs to the current pattern. If we consider forecasting the time series value at $t+1$, the distances measured using the most recent pattern $input_t^p$ can be viewed as a measure of dependencies between the set of candidate models and the target time series. These dependencies can be continuously computed and monitored over time. The distance d_i^t of the model f_i to $input_t^p$ is calculated using its representative RoC $\mathcal{R}_i$ for the forecasting at time $t+1$ (Equation 5.8).

$$d_i^t = D_E(input_t^p, \mathcal{R}_i), \forall i \in [1, M] \tag{5.15}$$

Naturally, with time-evolving data, dependencies between the current pattern and the computed RoCs change over time and follow non-stationary concepts. Stationarity in this context can be formulated as follows:

Definition 16 ***Weak Stationary RoCs Structure*** *Let $D_t = \{d_1^t, \cdots, d_M^t\} \in \mathbb{R}^M$ be a resulting similarity vector between the base models and the target time series $input_t^p$ over a window of size p (i.e., derived from the above similarity evaluation in Equation 5.15). We sort the value of D_t in an ascending order so that $D_t = \{d_{1,t}, \cdots, d_{M,t}\}$ with $d_{1,t} \leq \cdots \leq d_{M,t}$*

. Let δ_d denote the minimum value in D_t at the initial instant of its generation $t_i = t$. The dependence structure is said to be weakly stationary if the true mean of Δdif_t is 0:

$$\Delta dif_t = |d_{1,t} - \delta_d| \tag{5.16}$$

Following this definition, we can assume that the distance between the most similar models to the current pattern within the same pool of models $\mathcal{P}$ sets its boundary under a form of a logical *diameter*. If this boundary diverges in a significant way over time, a drift is assumed to take place. We propose to detect the validity of such an assumption using the Hoeffding Bound ζ similarly to the way suggested for detecting the drift in the deviation of the mean of the time series (Equation 5.7). Once the condition of the *Weak Stationary RoCs Structure* presented in Definition 16 is violated, an alarm is triggered, the ensemble pruning is updated by re-clustering of the models and re-selecting new ***top-K*** models. Afterward, the dependencies monitoring process is continued by sliding the time window to update $input_t^p$ by one value to produce the next forecast. Once a drift is detected at time instant t_d, the reference *diameter* δ_d is reset by setting $t_i = t_d$.

The concept drift detected in the time series is denoted as `Drift Type I` while the drift in the candidate models' performance is denoted as `Drift Type II`. All the steps of OEP-ROC are summarized in Algorithm 3.

5.4.6 Empirical Experiments

We present the experiments carried out to validate OEP-ROC and to answer these research questions:

- **Q1:** How does OEP-ROC perform compared to the State-of-the-Art (SoA) 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 impact of different values of the number of clusters K on the performance of OEP-ROC?
- **Q4:** What is the impact of choosing the size of ***top-K*** automatically using the bound in Equation 5.14?
- **Q5:** What is the benefit of each drift type detection for the performance of OEP-ROC?
- **Q6:** How scalable is OEP-ROC in terms of computational resources compared to the most competitive online model selection methods? What is the computational advantage of the *drift-aware* adaptation of pruning?
- **Q7:** How can different aggregation techniques benefit from our pruning method?

5.4.6.1 Experimental Setup

The methods used in the experiments were evaluated using the root mean squared error (RMSE). The used time series was split into three parts, where the first 50% is used for training (X_ω^{train}), the next 25% for validation (X_ω^{val}), and the last 25% for testing. The results are

Algorithm 3: OEP-ROC

```
Data: Parameters: Parameters: Number of Lags: $p$; size of the validation set: $\omega$;
      size of time windows within the validation set: $n_\omega$; CNNs Pool of size $M$: $\mathbb{P}$;
      Number of clusters: $K$.
1  Models Training and RoCs Computation: ;
2  Train each $f_i \in \mathbb{P}, i \in [1, M]$ on $X_\omega^{train}$. ;
3  Initialize RoC buffers $RoC^i$ for each $f_i, i \in [1, M]$;
4  for each $X_{n_\omega}^{val,j} \in X_\omega^{val}$ do
5      Compute the corresponding $R_j^i$ using PGSMs (i.e. $L_j^i$ Equation 5.3). ;
6      Add $R_i^j$ to the corresponding buffer $RoC^i$ ;
7  end
8  Online Forecasting: Forecasting next $H$ values :;
9  To forecast $t + 1$:;
10 for each $i \in [1, M]$ do
11     Select the representative RoC $\mathcal{R}_i$ for $f_i$ (Equation 5.8).;
12 end
13 Cluster the models into $K$ clusters using their representative RoCs;
14 Select the $K$ clusters' representatives following Equation 5.13;
15 Sort the $K$ representatives according to their distance to $input_t^p$.;
16 Select the top-K models whose distances are lower than the upper bound in Equation
     5.14.;
17 Predict $x_{t+1}$ using the selected top-K models using Equation 2.58 by setting all the
     weights to be equal to $\frac{1}{|\textbf{top-K}|}$;
18 To forecast $t + h$:;
19 for $h \in \{2, \cdots, H\}$ do
20     if Drift Type I detected then
21         Update $X_\omega^{val} = \{x_{t-\omega+h}, x_{t-\omega+2}, \cdots, x_{t+h-1}\}$ ;
22         Recompute and add new RoCs (steps: 4-7) ;
23     end
24     if Drift Type I $\vee$ Drift Type II detected then
25         Use the most recent pattern $input_{t+h-1}^p$ ;
26         Re-cluster and re-select the top-K models (steps: 11-17) ;
27     end
28 end
```

compared using the non-parametric Wilcoxon Signed Rank test [270]. We use the same 100 real-world time series datasets as in the experiments in the previous Section 5.3 to validate OEP-ROC. These datasets are described in the Appendix Table A.3.

OEP-ROC Setup and Baselines We construct a pool $\mathcal{P}$ of CNN-based candidate models using different parameter settings (e.g., the number of filters varies in $\{32, 64, 128\}$, kernel size varies in $\{1, 3\}$) like explained in Section 5.4.2. For the construction of the candidate models, we define four architectural building blocks. Our notation of *layer1-layer2* implies a sequential connection between the two layers. Let *Conv1* be a sequential sub-net made up of one convolutional layer with ReLU activation, followed by a batch normalization layer. *Conv2* is similar, except that we use max pooling instead of batch normalization. *Conv3* is the same as *Conv1*, but the number of filters for the convolutional layer is reduced by half. Lastly, we define a *ResidualBlock* as *Conv-BatchNorm-ReLU-Conv-BatchNorm-ResidualConnection-ReLU*. From these building blocks, we create our base learners as in Table 5.4, where each Dropout layer has a probability parameter of 0.9, and *FCN* refers to a Fully-Connected layer. There, we also show the different configurations we created by varying the number of filters in the convolutional layers as well as the number of hidden units in the LSTM layer. In total, this results in 33 candidate DNNs.

Table 5.4: Configurations and architectures for all candidate DNNs. 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 candidate DNNs.

Base learner	Architecture	Configurations
Shallow FCN	Conv1-Dropout-FC	$n_{filters} \in \{32, 64, 128\}$
Small CNN	Conv2-LSTM	$n_{filters} \in \{32, 64, 128\}$ $n_{hidden} \in \{10, 30\}$
Medium CNN	Conv1-Conv1-LSTM	$n_{filters} \in \{32, 64, 128\}$ $n_{hidden} \in \{10, 30\}$
Large CNN	Conv1-Conv1-Conv1-LSTM	$n_{filters} \in \{32, 64, 128\}$ $n_{hidden} \in \{10, 30\}$
Fewer Filter CNN	Conv3-Conv3-LSTM	$n_{filters} \in \{32, 64, 128\}$ $n_{hidden} \in \{10, 30\}$
One Residual	ResBlock-Dropout-FCN-Dropout-FCN	$n_{filters} \in \{32, 64, 128\}$
Two Residual	ResBlock-Resblock-Dropout-FCN-Dropout-FCN	$n_{filters} \in \{32, 64, 128\}$

OEP-ROC has also a number hyper-parameters that are summarized in Table 5.5.

Table 5.5: Hyperparameters of our method and their values for the experiments.

Parameter	Description	Value
N	Size of the Pool of base models $\mathbb{P}$	33
n_ω	Size of time windows within the validation set	60
z	Number of steps for sliding the n_ω time windows	25
p	Number of lags for training the base models	5
K	Number of base models clusters	$\{5, 10, 15, 20\}$
μ	Hoeffding-Bound parameter	0.05

SoA Methods Setup We compare OEP-ROC against the following approaches, which include SoA methods for forecasting and ensemble pruning methods devised for time series forecasting. Some of them operate in an online fashion. The SoA Forecasting Models include ARIMA [61], LSTM [118], ETS [97], CNN [15], CNN-LSTM [112]: The two best performing base models.

The online SoA pruning Methods include:

- Ran-Pr-m: consists of a random selection of candidate models to construct the ensemble with m indicating the ensemble size, Ens that denotes the ensemble of all the DNNs in $\mathbb{P}$.
- NCL [299]: is the Negative correlation Learning for pruning ensembles of deep learning methods. We use the same pool of base models as $\mathbb{P}$.
- OCL: is the variant of DEMSRC presented in Chapter 4-Section 4.3.5.1. More details about OCL are given in Chapter 4-Section 4.3.5.1. We feed the candidate DNNs in $\mathbb{P}$ to OCL.
- OTOP: is the variant of DEMSRC presented in Chapter 4-Section 4.3.5.1. More details about OTOP are given in Chapter 4-Section 4.3.5.1. We feed the candidate DNNs in $\mathbb{P}$ to OTOP.
- DEMSRC: is the online ensemble pruning method presented in Chapter 4-Section 4.3.5.1 that uses covariance-based model clustering. We feed the candidate DNNs in $\mathbb{P}$ to DEMSRC.
- DEMSRC-K: is the online ensemble pruning method presented in Chapter 4-Section 4.3.5.1 that uses K-Means-based model clustering using DTW distance. We feed the candidate DNNs in $\mathbb{P}$ to DEMSRC-K.
- OS-PGSM: is the online single DNN selection presented in Section 5.3.
- OS-PGSM-Int: is a variant of OS-PGSM (See Section 5.3.4.1).

OEP-ROC Variants :

- OEP-ROC-C: is the variant of OEP-ROC that uses only clustering without ***top-K*** selection.
- OEP-ROC-TOP: is the variant of OEP-ROC that performs ***top-K*** selection without clustering.
- OEP-ROC-ST: is the static variant of OEP-ROC. Pruning is decided at the initial forecasting instant and kept fixed along testing.
- OEP-ROC-Per: is the variant of OEP-ROC where the pruning is updated periodically in a blind manner (i.e., without taking into account the occurrence of the drift).
- OEP-ROC-I: is the variant of OEP-ROC that takes into account only the occurrence of `Drift Type I`.
- OEP-ROC-II: is the variant of OEP-ROC that takes into account only the occurrence of `Drift Type II`.

- OEP-ROC-K: is the variant of OEP-ROC indicating the number K of clusters used in the clustering stage of OEP-ROC.
- OEP-ROC-K-top-K-j: In OEP-ROC and all its above variants the size of ***top-K*** (i.e. |**top-K**|) is decided automatically using the bound in Equation 5.14. In this variant, we set the size of the ***top-K*** models to select to a fixed value j.

Aggregation Methods Setup : We also evaluate how different ensemble aggregation methods (i.e., ensemble weighting methods) can benefit from our pruning strategy. Instead of feeding all the candidate DNNs in $\mathbb{P}$ into the aggregation schema, we only input the models selected by the pruning procedure of OEP-ROC. To do so, we report the evaluation results over various aggregation methods, including:

- OEP-ROC-SW: is the variant of OEP-ROC that uses sliding-window ensemble [12] for aggregation instead of equal weighting.
- SW: Sliding-window ensemble [12] over all the models in $\mathbb{P}$.
- OEP-ROC-OGD: is the variant of OEP-ROC that uses Online Gradient Descent for ensemble aggregation [284].
- OGD: is the Online Gradient Descent aggregation [284] over all the models in $\mathbb{P}$.
- OEP-ROC-FS: is the variant of OEP-ROC that uses the Fixed Share method for aggregation [180].
- FS: is the Fixed Share method for aggregation [180] over all the models in $\mathbb{P}$. Further details about FS are provided in Section 2.6.3.2.
- OEP-ROC-EWA: is the variant of OEP-ROC that uses Exponential Weighting for ensemble aggregation [276].
- EWA: Exponential Weighting aggregation [276] over all the models in $\mathbb{P}$. Further details about EWA are provided in Section 2.6.3.2.
- OEP-ROC-MLPOL: is the variant of OEP-ROC that uses the Polynomial aggregation rule with different learning rates for each candidate DNN for aggregation [173].
- MLPOL: is the Polynomial aggregation rule for aggregation [173] over all the models in $\mathbb{P}$. Further details about MLPOL are provided in Section 2.6.3.2.

5.4.6.2 Comparing OEP-ROCto the State-of-the-Art

Table 5.6 presents the average ranks and their deviation for OEP-ROC, its variants, and SoA 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 the wins and losses for each dataset using the RMSE scores. We use the non-parametric Wilcoxon Signed Rank test [270] to compute significant wins and losses, which are presented in parenthesis (significance level 0.05).

Table 5.6: Comparison of OEP-ROC with $K = 15$ to different SoA methods on 100 time series. The rank column presents the average rank and its standard deviation across different time series for each method. An average rank of 1 means the model was the best performing on all the datasets.

Method	Our Method		
	Losses	Wins	Avg. Rank
OS-PGSM	35(23)	65(59)	4.76 ± 3.11
Ran-Pr-5	20(5)	80(69)	18.35 ± 4.93
Ran-Pr-10	25(6)	75(68)	15.91 ± 5.03
Ran-Pr- 15	31(3)	69(61)	12.82 ± 6.17
CNN	14(7)	86(66)	20.32 ± 6.90
ETS	11(4)	89(71)	24.98 ± 9.36
Ran-Pr-20	32(16)	68(58)	10.42 ± 6.17
ARIMA	16(7)	84(73)	14.91 ± 9.69
OTOP	25(5)	75(60)	15.32 ± 9.89
OEP-ROC-PER	27(9)	73(61)	10.07 ± 5.84
OS-PGSM-Int	31(18)	69(63)	5.30 ± 4.84
DEMSRC-K	19(7)	81(60)	15.52 ± 7.45
CNN-LSTM	26(12)	74(62)	12.63 ± 7.55
DEMSRC	32(19)	68(55)	13.68 ± 7.66
OEP-ROC-I	33(7)	67(35)	8.98 ± 5.69
OEP-ROC-II	30(7)	70(42)	10.08 ± 5.83
LSTM	43(30)	57(49)	8.88 ± 8.41
OEP-ROC-15-top-K-8	33(15)	67(36)	5.59 ± 4.53
NCL	49(25)	51(26)	9.07 ± 9.28
OEP-ROC-15-top-K-6	33(11)	67(33)	6.25 ± 5.19
Ens	47(27)	53(33)	7.73 ± 3.34
OEP-ROC-15-top-K-10	39(10)	61(35)	4.90 ± 4.55
OEP-ROC-TOP	38(13)	62(33)	4.97 ± 5.84
OCL	36(17)	74(61)	6.17 ± 6.66
OEP-ROC-C	39(13)	61(33)	3.79 ± 3.42
OEP-ROC-ST	37(10)	63(34)	6.31 ± 4.83
OEP-ROC	-	-	$\mathbf{3.29 \pm 3.08}$

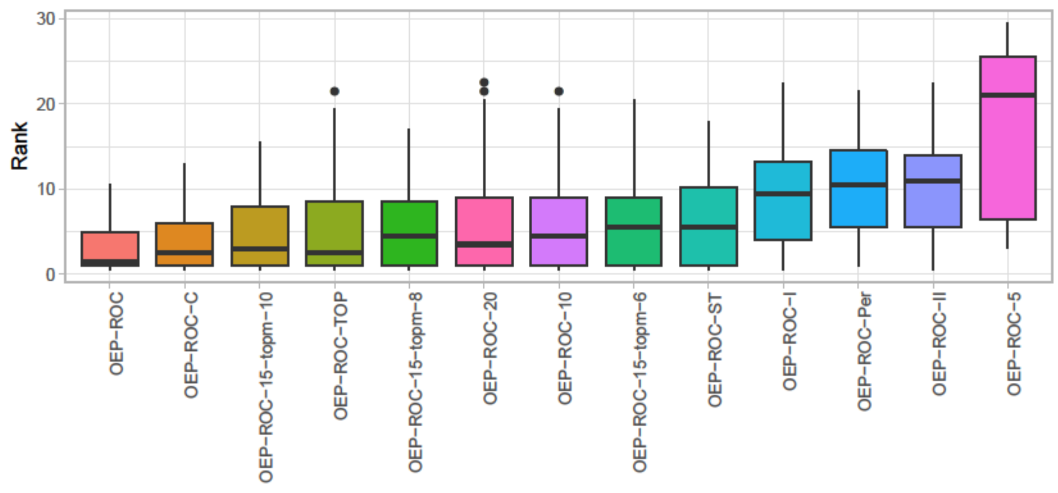


Figure 5.4: Distribution of the ranks of OEP-ROC with $K = 15$ in comparison to its variants across the different time series.

In the results in Table 5.6, OEP-ROC outperforms almost all the baselines in terms of ranks, wins, and losses in the pairwise comparison. Ens, OCL, and NCL seem to have quite good performance. However, their average ranks are approximately more than double the average rank of OEP-ROC. The online ensemble pruning methods, e.g., OCL, OTOP, DEMSRC, and DEMSRC-K, show inferior performance compared to OEP-ROC. ARIMA, ETS, and CNN, SoA methods for forecasting are considerably worse in the average rank compared to OEP-ROC. LSTM and CNN-LSTM show better performance, but still worse than OEP-ROC. The ensemble Ens that uses all of the 33 models also has a worse average rank compared to OEP-ROC which uses on average only 6 DNNs which is almost a fifth of the pool $\mathcal{P}$ size. In addition, applying online pruning methods to the pool $\mathbb{P}$ such as OCL, OTOP, DEMSRC, and DEMSRC-K which are general methods applicable to any family of forecasting models, shows inferior performance compared to OEP-ROC and all its variants. This highlights the usefulness of developing methods that are specific to DNNs and that take into account the information provided by their gradient-based training mechanism. Furthermore, OEP-ROC is better than OS-PGSM in terms of average rank, wins, and losses. This highlights the advantage of the general idea of using an ensemble of experts instead of one single expert since the mixture of many experts covers the weakness of some and mitigates to some extent the wrong selection of some learners. These results address the research question **Q1**.

5.4.6.3 Comparing OEP-ROC to its Variants

Figure 5.4 represents the average rank, and respective standard deviation, of OEP-ROC and its variants. Table 5.7 presents the average ranks and their deviation for OEP-ROC and its variants. It can be seen from Table 5.7 that none of the pruning stages on its own (i.e., OEP-ROC-C or OEP-ROC-TOP) is able on its own to achieve good results similar to OEP-ROC. This highlights the importance of each pruning stage as each of them is carefully designed to promote one of the main ensemble properties, namely *accuracy* and *diversity*. It is also clear that OEP-ROC-C has better performance than OEP-ROC-TOP since proceeding by only ranking the candidate models according to their closeness to the current pattern kills

Table 5.7: Comparison of OEP-ROC with $K = 15$ to its variants on 100 time series datasets. The rank column presents the average rank and its standard deviation across different time series. An average rank of 1 means the model was the best performing on all the datasets.

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

the diversity (i.e., 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). In this way, the ensemble ambiguity is decreased even though the average error of its members is decreased. This shows that our two-staged procedure helps to establish a trade-off between ensemble *diversity* and *accuracy*. This answers the research question **Q2**.

5.4.6.4 Usefulness of the Theoretical Insights

The right set-up of the number of clusters K to be computed seems also to be an important factor for the performance. While it can be seen from Table 5.7 that low values of K like 5 do not help to achieve a good performance, increasing the value of K seems to improve largely the rank of OEP-ROC (i.e., decreasing the rank which means better performance across all the datasets). This can be mainly explained by the fact that a small number of clusters would lead to bigger clusters sizes which means that the selection of the clusters' representatives will no longer be representative of the same error level by all the candidate models belonging to the same cluster, so the control over the average error of these models is lost. Increasing the value of K too much is also not desired since it would lead to small cluster sizes and more similar clusters' representatives, which may alter the diversity by decreasing the ambiguity. This answers the research question **Q3**.

Table 5.7 also shows the usefulness and the benefits of our theoretical insights in setting up the number of ***top-K*** automatically. This number is set up by the derived bound (See Equation 5.14) such that the ensemble members' average error is reduced and the ambiguity is increased. Fixed values for j for $|\boldsymbol{top\text{-}K}|$ could not achieve the same performance as OEP-ROC. The best-fixed model selection configuration is for $j = 10$ OEP-ROC-15-top-K-10 which has a bigger average rank than OEP-ROC. This addresses the research question **Q4**.

5.4.6.5 Importance of the Drift-aware Adaptation

It can be seen from Table 5.7 that none of the drift detection mechanisms (i.e., OEP-ROC-I and OEP-ROC-II) is able on its own to achieve a good performance. The combination of both is necessary to update the pruning at the right moment when it is needed. Further details are given in Sections 5.4.5.1 and 5.4.5.2. It is also clear that performing the updates periodically in a random blind manner with OEP-ROC-Per does not necessarily help in improving the performance even though it is designed to trigger more updates than all the drift-aware methods. This demonstrates that informed adaptation is always beneficial. This answers the research question **Q5**.

Table 5.8: Average run-time plus variance (both in seconds) for three variants of OEP-ROC over 5 datasets.

Method	Mean run-time (in seconds)	Variance of run-time (in seconds)
OEP-ROC	14.41	13.03
OEP-ROC-ST	0.49	0.24
OEP-ROC-Per	27.61	0.66

To compare scalability between our configurations, 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 5.8. As can be seen, OEP-ROC-ST is by far the fastest method on average since it does not adapt its RoCs during the run-time of the algorithm. We noticed that recreating the RoCs takes by far the longest time duration in comparison to other steps of the algorithm, and we plan to address this in future work. OEP-ROC-Per illustrates this problem the best since it blindly and frequently recreates the RoCs and re-clusters the models. Thus, its run-time is always higher, 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 whether or not an adaption to drifts is necessary. As can be seen from Table 5.8, this results in a high variance in the run-time since some datasets contain more concept drifts than others. We see this behavior as a benefit since OEP-ROC outperforms the other two variants, indicating that sometimes a higher run-time can be justified by overall better performance.

5.4.6.6 Impact of Different Aggregation Techniques

Table 5.9 shows the average ranks, and respective standard deviations, of different aggregations methods taking as input at one time the pruned models by OEP-ROC and all the base models in $\mathbb{P}$ at a second time. It can be seen from this table that all aggregation methods achieve better results when combined with OEP-ROC than using all the candidate models in $\mathcal{P}$ as input. 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 $M = 33$ to an average of $|\boldsymbol{top\text{-}K}| = 6$. This makes the regret minimization problem in EWA and MLPOL easier to solve (See Section 2.6.3.2). In addition, the weighting schema is focused more on the most suitable models in terms of *accuracy* and *diversity*. The difference in performance between the methods can be explained by the difference in the weight learning paradigms behind each method. This addresses the research question **Q7**.

5.4.6.7 Final Remarks

The empirical results indicate that OEP-ROC has performance advantages compared to popular forecasting methods and SoA approaches for online ensemble pruning. We show that

Table 5.9: Comparison of OEP-ROC with $K = 15$ combined with different aggregation methods for 100 time series datasets.

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

our method using RoCs-based pruning in two well-studied stages is able to gain excellent and reliable empirical performance in our setting. The informed adaptation and update of the pruning decision following concept drift detection in both time series and candidate models' performance make our method, in addition to having a better predictive performance, computationally cheaper.

The gradient-based saliency maps PGSMs used to compute the candidate models' RoCs can be used for the explainability of online DNN selection and ensemble of DNNs pruning. More details and examples are presented in the following Section.

5.5 Explainable Deep Neural Network Selection

In this section, we show how the computed saliency maps to determine the DNNs' RoCs, inspired by the Grad-CAM, can be exploited to explain the online selection of DNNs and helps in understanding their expertise/competence. Similarly to the previous Chapter 4-Section 4.5 and to the way the saliency maps produced by the Grad-CAM are exploited for explainability in Computer Vision [203], [300], we provide visualization-based explanations.

First, we show how OS-PGSM is used to provide suitable explanations for the reason behind selecting a specific single DNN at a certain time interval or instant in Section 5.5.1. Second, in Section 5.5.2, we present how OEP-ROC is exploited to explain the reason behind selecting specific DNNs to compose the ensemble at a certain time interval or instant. In addition, OEP-ROC is also used to provide reasonable explanations for the performance of a pruned ensemble at a certain time interval or instant.

5.5.1 Single Deep Neural Network Selection

We provide some insights into how OS-PGSM can be used to provide suitable explanations for the reason behind model selection. Note we use the notation C_i for the candidate models in the figures instead of f_i to remind that we deal with a pool of candidate CNN-based models. Figure 5.5 shows a comparison between the current input time series pattern $input_t^p$ (left part in black) and the RoC of the selected model to perform the forecast (right part in blue). 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. This is further validated when also comparing the true time series value (ground truth,

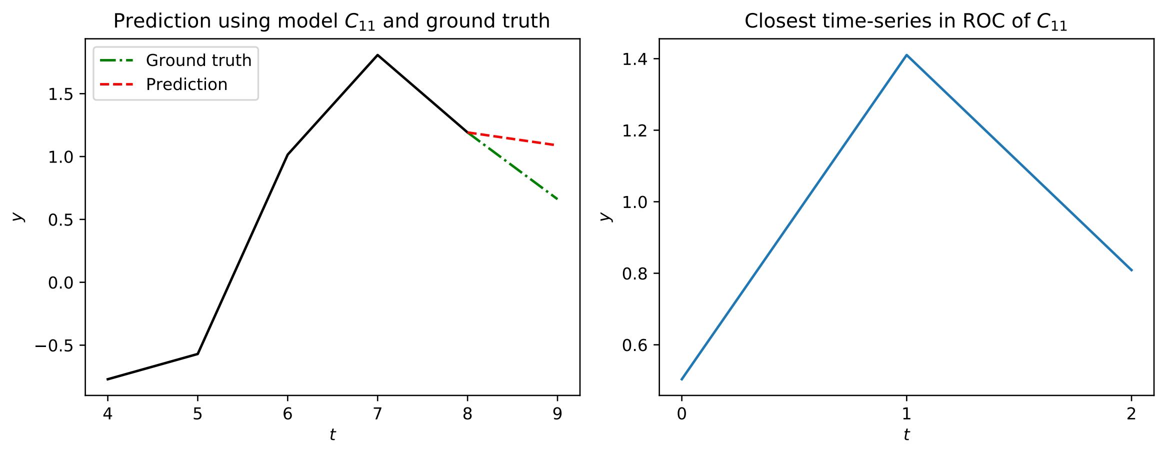


Figure 5.5: Comparison of the current input pattern to the closest RoC (C_{11}). The Forecaster C_{11} was chosen to predict the next value (left plot, green line) because the input (left plot, black line) was closest (in terms of DTW) to the time series on the right side from C_{11}'s region of competence R^{11}

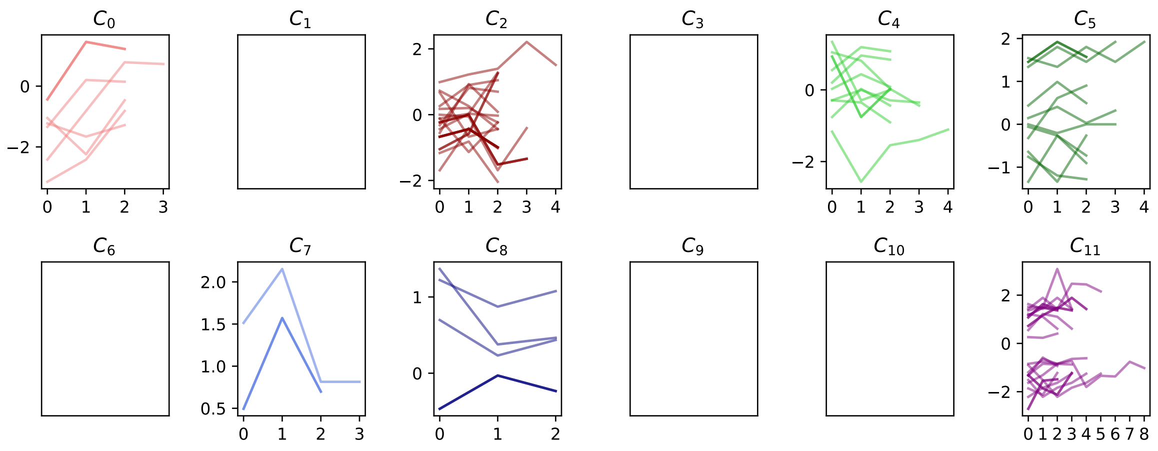


Figure 5.6: Visualization of RoCs for AbnormalHeartbeat data using OEP-ROC.

green) and the predicted value (red). while these two values differ slightly, an evaluation of all the candidates at this point showed that our selected model C_{11} has the smallest error.

A more general overview of the RoCs for AbnormalHeartbeat dataset is shown in Figure 5.6. Some models have quite similar RoCs (patterns). For example, C_0 appears to be an expert in increasing linear trend patterns or in peaks followed by a slight plateau, while C_7 is the best in dealing with sharp peaks. As can be seen with the varied amount of transparency of lines of the RoCs, many identical RoCs are collected for each model, confirming thus the assumption that certain models are experts on specific input regions of the time series. Some models do not have any RoC. This can be explained by the fact that they never get selected in the validation process, or their PGSMs are too small to form a pattern which is why they get filtered out in the smoothing procedure. 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. Practitioners of our method can then use this insight to focus their attention on improving certain poorly performing models or remove the unused models entirely to save run-time.

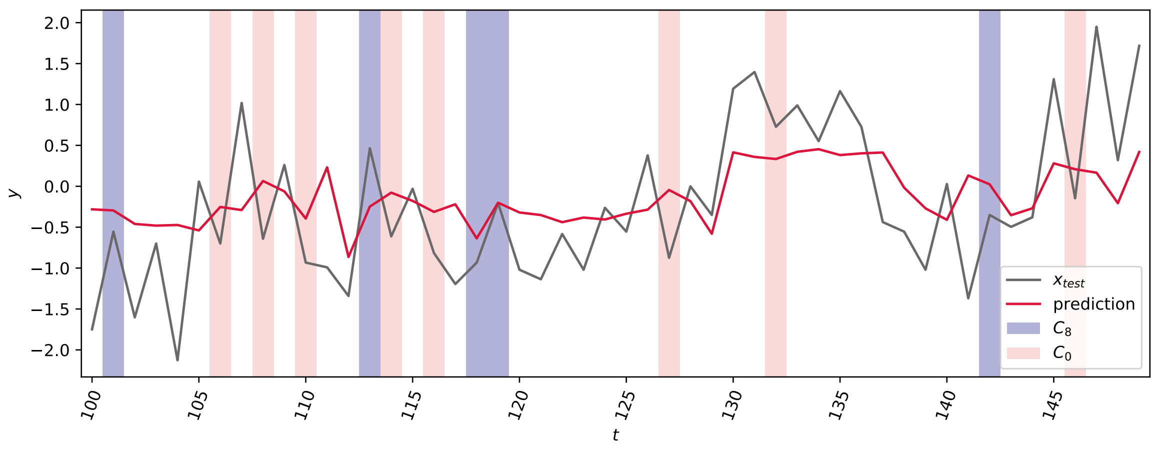


Figure 5.7: A visualization of model selection on the AbnormalHeartbeat dataset.

Another visualization aspect of the forecasting is shown in Figure 5.7. The prediction of OS-PGSM is shown in red in contrast to the ground truth values (black). We focus for visualization clarity reasons on the two models (C_8 and C_0). We highlight regions where they are selected by OS-PGSM. Notice that preceding every decision where C_0 is chosen, the time series exhibits a peak, which corresponds to the model's RoCs in Figure 5.6. The same conclusion can be drawn for C_8, which is picked after valley-shaped parts. While the two models are not picked for all peaks and valleys, our method clearly aligns certain time series regions with specific models.

5.5.2 Ensemble of Deep Neural Networks Pruning

We provide some insights into how OEP-ROC can be used to provide suitable explanations for the reason behind specific candidate models selection to construct the ensemble at a specific time instant of interval. First, we compare the clusters' representatives of OEP-ROC-10 (top row) and OEP-ROC-10-top-K-6 in Figure 5.8. 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 without being too far from the current pattern, which shows the trade-off established by OEP-ROC between ensemble *accuracy* and *diversity*. OEP-ROC-10-top-K-6 promotes the selection of diverse patterns. However, the fixed size results in too many uninformative models (i.e., models outside the p-sphere; See Section 5.4.4 for further details) being picked, leading thus to a higher averaged error.

Second, Figure 5.9 shows a comparison between the current input time series pattern $input_t^p$ (left part in black) with the RoCs of the pruned ensemble to perform the forecast on the right. 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 degree of competence in forecasting using patterns with a similar trend to the input. This is further validated when also comparing the true time series value (ground truth, red) to the predicted value (green). While these two values differ slightly, an evaluation of all the candidates at this point by integrating all of them in an averaged ensemble showed that the ensemble by OEP-ROC has a smaller error.

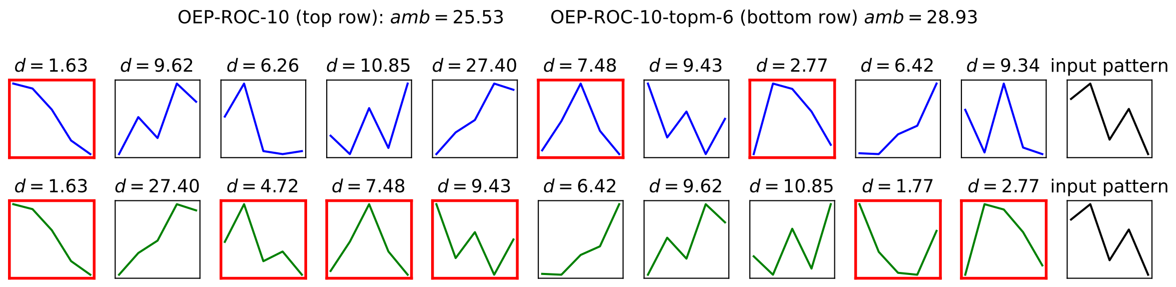


Figure 5.8: Comparison of the clusters' representatives of OEP-ROC-10 (top row) with $K = 10$ and OEP-ROC-10-top-K-6 (low row) (also $K = 10$) on the Abnormal Heartbeat dataset. We report the ambiguity *amb* of the clustered ensemble as well as the Euclidean distance of each RoC to the input pattern, which is shown in the rightmost column. In red, we visualize the models that were chosen by each method for prediction.

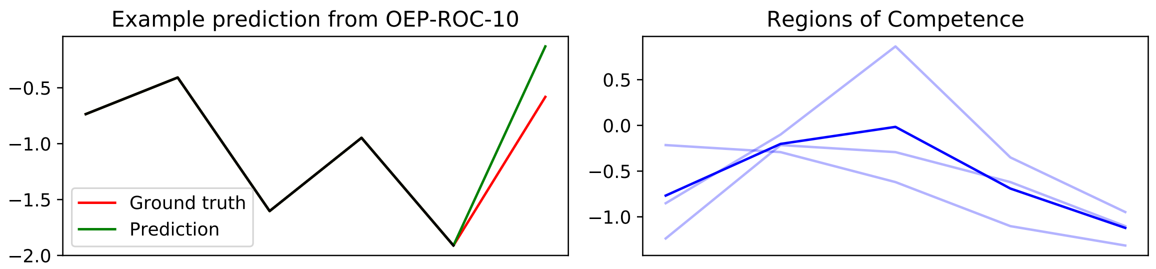


Figure 5.9: Left: For the time series to predict (black), we show the ground truth value (red) as well as the prediction of OEP-ROC-10 (green). **Right**: Visualization of each nearest RoC from the selected models (light blue), as well as the mean RoC (dark blue). Both plots were generated on the Abnormal Heartbeat dataset.

5.6 Concluding Remarks

In this chapter, we have presented two methods for online DNNs selection in the task of time series forecasting.

First, OS-PGSM: a novel, practically useful online CNN-based model selection uses Performance Gradient-based Saliency Maps PGSMs to derive the Region of Competence RoCs of a set of candidate CNNs. These RoCs are updated in an informed manner using concept drift detection in the time series.

Second, OEP-ROC: a novel, practically useful online ensemble of DNNs pruning method also uses the principle of PGSMs. The pruning is updated online in an informed manner using concept drift detection in both time series and ensemble members' performances. The different pruning steps, in addition to the size of the resulting pruned ensemble, are supported by theory.

The PGSMs in these two methods can be used to support the explainability of both online single DNN selection as well as an ensemble of DNNs pruning.

An exhaustive empirical evaluation, including 100 real-world datasets and multiple comparison algorithms, showed the advantages of OS-PGSM and OEP-ROC in terms of performance and scalability.

In the last two chapters, we have investigated the task of ensemble pruning, where we highlighted the main challenges imposed by the nature of this task, the requirements that have to be fulfilled to cope with the time-evolving nature of time series data, and their impact on improving the ensemble's performance once online adaptive pruning methods are developed and deployed. In the following chapter, we will explore the next stage in ensemble model construction, namely ensemble aggregation. More precisely, we develop a novel online adaptive ensemble aggregation schema using Deep Reinforcement Learning.

6 Online Ensemble Aggregation using Deep Reinforcement Learning

In this chapter, we study the last ensemble construction stage, namely ensemble aggregation or combination. More precisely, we develop an online novel weighting schema for linearly-weighted ensembles using the Reinforcement Learning paradigm.

6.1 Introduction

Several approaches, ranging from simple and enhanced averaging tactics to applying meta-learning, have been proposed to learn how to combine individual models in an ensemble. Hence, as we have already detailed in Section 2.6.3, aggregation strategies in an ensemble can be categorized into three main groups. The first group (e.g., bagging [31]) is based on voting schemes that use either a majority or a (weighted) average of the votes to decide the final output. The outputs of the individual models are repeatedly added to the training set once at a time in the second group of methods, which uses the cascading paradigm. The third group is based on a stacking technique [21], and in this context, meta-learning is used to learn the aggregation of actual outputs from prior experiences. That is, individual models' outputs are usually mapped to the true time series values and fed to an ML model that is trained to learn an aggregation rule, i.e., model's parameters, that minimizes the loss between predicted and target values, i.e., true time series values [34]. For more details, see Section 2.6.3.3.

However, there is no single way to estimate the weights of each model in a linearly weighted ensemble; thus, learning the optimal aggregation strategy remains an open research question [10], [36]. The online ensemble aggregation task is also very challenging since high-dimensional continuous weights' values have to be accurately estimated in real-time. In time series analysis, we additionally need to take the temporal properties of the underlying process into account. More specifically, since time series are inherently time-ordered, and the forecasting task hence depends on a sequential analysis that can capture the temporal patterns in this data, we consider a sequential approach based on Reinforcement Learning (RL) for dynamic ensemble aggregation. Hence, we leverage a Deep Reinforcement Learning (DRL) framework for learning linearly weighted ensembles as a meta-learning method.

In this method, we first create a set of individual models that are trained in parallel and separately from each other. Second, we take an aggregation policy based on a weighted average, in which the outcomes of these individual models are averaged linearly to generate the ensemble outcome. Consequentially, the set of optimal weights is learned using a DRL approach in a continuous space of weights that would lead to the most accurate ensemble construction given a finite window of recent time series observations. This method is denoted in the following as OEA-DRL: Online Ensemble Aggregation using DRL.

Since the time series can be subject to significant changes, i.e., concept drifts, the aggregation policy of an ensemble requires to be dynamically adapted to these changes. Therefore, the considered sequential approach based on DRL is devised such that it is able to capture the temporal changes that occur in the data and the ensemble's performance and provide the optimal set of weights in real-time for online applications.

OEA-DRL is based on an actor-critic approach in deep learning settings to learn the aggregation policy of a linearly weighted ensemble. In this context, we exploit the idea from Lillicrap et al. (2015) [301] to learn an RL policy in continuous action space, in which the actions are considered to be the set of weights of the ensemble. These weights are determined online by applying the learned policy to the recently observed sliding window of time series observations. In addition, the policy is updated online each time a concept drift in the performance of the ensemble learned using the weights derived from previous policies is detected.

We further conduct a comprehensive empirical analysis to validate OEA-DRL using the 100 real-world time-series datasets used in the previous experiments of the Chapters. The obtained results demonstrate that OEA-DRL outperforms standard State-of-the-Art methods for ensemble learning and aggregation.

6.2 Related Work

Related works on ensemble aggregation are presented with details in Section 2.6.3. In this section, we present a few works that went in the direction of exploiting RL in the context of ensemble aggregation.

For OEA-DRL, we exploit the paradigm of RL to learn an aggregation policy as a meta-learning technique. However, it should be noted that some approaches address the opposite task of Meta-RL, which denotes meta-learning on RL problems [302], [303]. There are merely a few approaches in the literature that leverage RL in the context of ensemble learning. Partalas et al. [168], [304] propose an algorithm based on Q-learning for ensemble pruning in classification problems, i.e., whether to include or exclude a classifier from the ensemble by searching the space of models. Nevertheless, the approach performs in a static setting and is a search algorithm rather than a learning scheme. RL is used for model selection in a time series forecasting task [305]. In this approach, a tabular Q-learning agent learns the optimal policy of selecting the best model from a pool of forecasting models for every time step. In both methods, the RL algorithm is developed in the finite state and action spaces, which renders a straightforward learning procedure. In the latter case, the discrete action space consists only of binary vectors indicating the fact of switching from a given selected model to a new one.

In OEA-DRL, we aim at directly learning the weights in continuous action space, in which the optimal aggregation policy is updated online following changes in the performance of the ensemble. In other words, the previously learned policy for aggregation is no longer accurate and has to be updated using recently collected time series observations. Once the optimal aggregation policy is learned, it is immediately applied to serve the online forecasting stage.

6.3 Online Ensemble Aggregation

This section presents OEA-DRL which uses DRL for learning the optimal ensemble aggregation policy. The informed update of the aggregation policy in an online learning setting is also explained.

6.3.1 Preliminaries

We keep the same notations as in the previous chapter, Section 5.4.1 for the time series, the pool of forecasting models, and the vector of forecast values at $t+h$.

We remind that an ensemble $\overline{f}$ of the pool of models $\mathbb{P}$ can formally be written at $t+h$ as a convex combination of the individual predictions of the models in $\mathbb{P}$ at $t+h$.

$$\bar{f}_{\mathbb{P}}(\hat{x}_{t+h}) = \sum_{i=1}^{M} w_{i,t+h}\hat{x}_{t+h}^{f_i} = \hat{x}_{t+h}^{\bar{f}_{\mathbb{P}}} \tag{6.1}$$

where $w_{i,t+h}, i \in [1, M]$, determine the weights attributed to the model f_i. The weights are shown to be time-dependent (as shown in Equation 6.1) since the optimal weight setting (i.e., aggregation policy) should naturally be dependent on the dynamics of the time series data that change over time as we have already explained in Section 2.6. We thus aim to find a set of weights for the ensemble that minimizes the expected prediction error for the next forecast

$$\begin{aligned} &\underset{w_{i,t+h},\forall i}{\operatorname{argmin}} \quad \mathbb{E}\big[\Big(f(x_{t+h}) - \overline{f}(x_{t+h})\Big)^2 |X_{1:t+h-1}\big], \\ &\text{s.t.} \quad w_{i,t+h} \geq 0, \forall i \in [1, M], \sum_{i=1}^{M} w_{i,t+h} = 1 \end{aligned} \tag{6.2}$$

In OEA-DRL, the weights are adaptively changed over time following the predicted policy by the DRL component and the detected drifts in the ensemble performance, which show that previous aggregation policies are no longer valid and the DRL component has to be updated by taking into account recently acquired time series observations.

A Reinforcement Learning RL task is formally expressed through a Markov Decision Process (MDP) [306]. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, in which $\mathcal{S}$ are the states, $\mathcal{A}$ the actions, $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ the reward function, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function, and γ is the discount factor. The goal in RL is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the total obtained reward.

In the phase of offline learning, we first split $X_{1:t}$ into $X_{\omega}^{train} = \{x_1, x_2, \cdots, x_{t-\omega}\}$ and $X_{\omega}^{val} = \{x_{t-\omega+1}, x_{t-\omega+2}, \cdots, x_t\}$, with ω a provided window size. X_{ω}^{train} is used for learning and X_ω^{val} is used for meta-learning. Subsequently, we train each of the individual M models on X_ω^{train} or its corresponding embedding (i.e., used for standard regression learning models). In this work, the pool of single models $\mathbb{P}$ is designed to contain a set of *heterogeneous* models, such as Gaussian processes, support vector regression, and neural networks, in order to boost ensemble *diversity*, which is proven in the literature [148], [150], [152] and over the previous Chapters, to be one of the most important aspects in constructing ensemble models. The *diversity* is hence reflected in the distinctive patterns derived from the inductive bias of each individual learner, given various hypotheses on which each learner is built to model the input data and its dependence structure.

6.3.2 Learning the Optimal Ensemble Weights

In this section, we present the details of our framework for online adaptive linear ensemble aggregation using DRL. RL is an iterative process devised to learn the impact of possible taken actions in the context of different states, i.e., situations, in order to determine the actions that result in maximizing the overall reward. Therefore, the application of RL requires first the definition of the corresponding environment composed of the set of states, actions, transitions, and rewards.

In OEA-DRL, our goal is to assign the optimal weights to the ensemble's members, given time series observations in a defined time interval, i.e., window. Since our main goal is to forecast future time series values step by step and monitor the learned policy continuously over time, all the RL environment components are time-dependent. Assume our goal is to forecast the time series value at $t+h$. Consequentially, the MDP is described as follows.

6.3.2.1 The MDP Framework

Before applying RL, the corresponding environment should be well-defined to suit the approached problem. An action $a_{t+h} \in \mathcal{A}$ corresponds to the vector of weights assigned to each of the M base models in the ensemble at time instant $t+h$ (i.e., the time instant at which we want to produce the forecast and the optimal aggregation are required). The action space is a continuous M-dimensional space.

$$\boldsymbol{a}_{t+h} = (w_{1,t+h}, w_{2,t+h}, \cdots, w_{M,t+h})^T = \boldsymbol{w}_{t+h}. \tag{6.3}$$

As previously indicated, current works that employed RL for model selection or ensemble pruning employed a static state specification that mainly consisted of the names or *IDs* of the candidate models to be selected and were not directly related to the data. In other words, these models did not take into account changes within the modeled data. However, because we are dealing with time-evolving data and the performance of the base models is related to the data sequence we are looking at, the state definition should be linked to the time series values or the values of a particular sequence of interest and the combination policy should be tailored to the data characteristics. The validation set at $t+h$ defined by $X_{t+h-\omega:t+h-1}$ includes a window of previous ω values of the time series until time $t+h-1$, i.e. $\{x_{t+h-\omega}, x_{t+h-\omega+1}, \cdots, x_{t+h-1}\}$. We consider a state $s \in \mathcal{S}$ to be the current window of time series that is used for forecasting the next value, i.e., x_{t+h}. However, the next state $s' \in \mathcal{S}$ should be defined in a way that shows the impact of a taken action a_{t+h} at state s. Therefore, we instead take the output of the ensemble in the given window of size ω as the state since it reflects the result of the aggregation and is dependent on $X_{t+h-\omega:t+h-1}$ itself. That means:

$$s = \{\bar{f}_{\mathbb{P}}(\hat{x}_{t+h-\omega}), \bar{f}_{\mathbb{P}}(\hat{x}_{t+h-\omega+1}), \cdots, \bar{f}_{\mathbb{P}}(\hat{x}_{t+h-1})\} \tag{6.4}$$

where $\bar{f}_{\mathbb{P}}(\hat{x}_t)$ is the ensemble output at time t given by Equation 6.1 and using the weight vector a_{t+h}. Note that the transition function $\mathcal{P}(s'|s,a)$ is deterministic in our setting.

We further define the reward of taking action a_{t+h} on the state s as a function $\mathcal{R}(s,a)$, which for simplicity in notation, we denote by r. In the RL setting, the learning process is devised to optimize the resulting reward. Hence, one intuitive way of setting up the reward is to opt for an ensemble error-related measure since our goal is to determine the optimal aggregation that would result in the most accurate ensemble model. Nevertheless, the magnitude of this error does not only depend on its performance or on the performance of the individual models but also relates to the time-varying structure of the time series itself. This can result in a slower convergence rate of the RL algorithm. Therefore, to stabilize the reward, we opt for a rank-based definition instead.

$$r = M + 1 - \rho^{\bar{f}_{\mathbb{P}}}, \tag{6.5}$$

with M is the size of the pool $\mathbb{P}$ of individual models to which we add the ensemble model to form a total of $M+1$ models. Subsequently, a ranked list of all the models (including the ensemble) is compiled using their corresponding forecasting error, in which $\rho^{\bar{f}_{\mathbb{P}}}$ indicates the rank of the ensemble. The lower the rank $\rho^{\bar{f}_{\mathbb{P}}}$ is, the more accurate the ensemble is (i.e., a rank 1 means $\bar{f}_{\mathbb{P}}$ performs the best) and the higher the reward r is. An empirical evaluation of the importance of the adequate choice of the reward function is illustrated in Section 6.4.

6.3.2.2 Learning the Aggregation Policy

Once the meta-learning task is phrased in an MDP framework, a policy π is learned in favor of maximizing the reward r, which positively correlates to a measure of the ensemble accuracy. Consequently, the objective function presented in Equation 6.2 is turned into learning the optimal policy of the MDP via an RL algorithm. We employ the idea from [301] to learn an optimal policy in a continuous action space using an actor-critic approach. This approach is selected since it is well-suited for both continuous and high-dimensional action and state spaces. In fact, enlarging the pool size M of the single models or ω the size of the validation set leads to high-dimensional action and/or state vectors. In this architecture, the actor is accountable for selecting an action given the current state, and the critic estimates a value function that provides adequate evaluation for the actor's action. Both parts can be represented by (Deep) Neural Networks that can be optimized by a gradient descent-based method. As a result, the actor and the critic networks are called the *policy network* and the *value network*, respectively. The value network predicts the value of an action $\boldsymbol{a}_{t+h}$ on the state s via $Q(s, \boldsymbol{a}_{t+h}|\phi)$, where ϕ is the parameter vector of the value network. On the other hand, the policy network learns a policy $\pi(s|\theta)$, which yields a deterministic policy on the state s given the network parameters θ. During the learning, the actor takes the gradients derived from the policy gradient theorem and adjusts the policy parameters θ, and the critic network estimates the approximate value function for the current policy π. The steps of learning the optimal aggregation policy using the actor-critic approach as explained in [301] are detailed in Algorithm 4. For simplicity of notation, we denote $\boldsymbol{a}_{t+h}$ by $\boldsymbol{a}$.

Algorithm 4: Learning the Optimal Aggregation Policy

```
   Data: Parameters: discount factor: $\gamma$; number of episodes: $max.ep$; a number of
         iterations: $max.iter$; learning rate: $\alpha$
1  Initialize randomly the parameters $\theta$ and $\phi$ of the actor and critic networks,
    respectively. ;
2  Initialize a replay buffer $R$;
3  for  $episode = 1 \in max.ep$ do
4      Initialize a random process $\mathcal{N}(0,1)$ for actions exploration.;
5      Receive initial observation state $s^1$.;
6      for  $i = 1 \in max.iter$ do
7          Select action $\boldsymbol{a}^i = \pi(s^i|\theta)$.;
8          Execute action $\boldsymbol{a}^i$ and observe reward $r^i$ and new state $s^{i+1}$.;
9          Store transition $(s^i, \boldsymbol{a}^i, r^i, s^{i+1})$ in $R$. ;
10         Sample a random mini-batch of $N$ transitions $(s^k, \boldsymbol{a}^k, r^k, s^{k+1})$ from $R$.;
11         Update $Q(s, \boldsymbol{a}|\phi)$ using target $y_k = r^k + \gamma Q(s^{k+1}, \pi(s^{k+1}|\theta)|\phi)$.;
12         Evaluate $\hat{A}^{\pi}(s^k, \boldsymbol{a}^k) = r^k + \gamma Q(s^{k+1}, \pi(s^k|\theta)|\phi) - Q(s^k, \pi(s^{k-1}|\theta)|\phi)$.;
13         Update the actor policy using the sampled policy gradient:;
14
                 $$\nabla_\theta J \approx \sum_k \nabla_\theta log\pi(s|\theta)\hat{A}^{\pi}(s^k, \boldsymbol{a}^k)$$
15         Update policy network: $\theta = \theta + \alpha\nabla_\theta J$.;
16     end
17 end
18 Return policy network $\pi(s|\theta)$.
```

6.3.3 Online Aggregation Update and Forecasting

Assume we want to forecast future values of the time series at $t+h$ starting from $t+1$, i.e., $h \geq 1$. After the policy network $\pi(s|\theta)$ is learned, we apply the model for predicting the weights of the ensemble (i.e., actions $\boldsymbol{a}_{t+h}$) that will be used for predicting the future values of time series in an online manner. Starting at $t+1$, the state s is set to $\{\bar{f}(\hat{x}_{t-\omega+1}), \bar{f}(\hat{x}_{t-\omega+2}), \cdots, \bar{f}(\hat{x}_t)\}$, the predicted weights via $\boldsymbol{a}_{t+1}$ are used to predict x_{t+1}. Afterward, the ω-length vector of time-series $X_{t+1-\omega:t}$ (i.e., the state s) is moved forward by one value. That means the oldest value is removed, and the predicted value $\bar{f}(\hat{x}_{t+1})$ is added to the current window. The new state s' and $\pi(s|\theta)$ are employed to predict the weights of the ensemble to forecast the next value of the time series.

The problem consists of forecasting future values step by step in a streaming online manner. Once the value at $t+h$ is collected, a time series of the ensemble's residuals RES can start to be collected in a buffer $B^{\bar{f}_{\mathbb{P}}}$ with:

$$RES_h^{\bar{f}_{\mathbb{P}}} = |x_{t+h} - \bar{f}_{\mathbb{P}}(\hat{x}_{t+h})| \tag{6.6}$$

We keep continuously monitoring the residual time series RES. If a drift is detected in RES, it indicates a significant change in the performance of the ensemble. This can be interpreted by the fact that the old ensemble construction, i.e., aggregation policy, is no longer accurate and unable to capture the new dynamics in the time series. For this purpose, we employ the Page-Hinkley (PH) test [307], a well-known change detection method. Its pseudo-code is introduced in Algorithm 1 in Section 3.3.3. Once a drift alarm is triggered in a given instant $t + h_d$ (i.e., the output of the PH-test), a new aggregation policy has to be recomputed. Naturally, upon an alarm trigger, the error estimators of the PH-test are restarted alongside the overall algorithm. The steps of the online policy update and forecasting are summarized in Algorithm 5.

Algorithm 5: Online Policy Update and Forecasting in OEA-DRL

```
   Data: Validation set $X_{t+1-\omega:t}$; Policy $\pi(s|\theta)$ ; Window size: $\omega$
1  Initialize residuals buffer for the ensemble $B^{\bar{f}_{\mathbb{P}}}$ ;
2  Initialize the Boolean variable $Alarm$ to $False$ ;
3  if $h == 1$: Forecasting the value of $X$ for $t+1$ then
4      Set $s$ to $\{\hat{x}^{\bar{f}_{\mathbb{P}}}_{t-\omega+1}, \hat{x}^{\bar{f}_{\mathbb{P}}}_{t-\omega+2}, \cdots, \hat{x}^{\bar{f}_{\mathbb{P}}}_{t}\}$ ;
5      Learn the combination policy $\pi(s|\theta)$;
6      Predict $\boldsymbol{a}_{t+1} = \boldsymbol{w}_{t+1}$ using $\pi(s|\theta)$;
7      Predict $x_{t+1}$ using Equation 6.1;
8      Compute $RES^{\bar{f}_{\mathbb{P}}}_{t+1}$ using Equation 6.6 ;
9      Add $RES^{\bar{f}_{\mathbb{P}}}_{1}$ to $B^{\bar{f}_{\mathbb{P}}}$;
10 end
11 for $h \in \{2, \cdots, N_f\}$ do
12     update $s$ by removing oldest value $x_{t+h-\omega}$ and adding $x_{t+h-1}$
13     Predict $\boldsymbol{w}_{t+h}$ using $\pi(s|\theta)$;
14     Predict $x_{t+h}$ using Equation 6.1;
15     Compute $RES^{\bar{f}_{\mathbb{P}}}_{h}$ using Equation 6.6 ;
16     Add $RES^{\bar{f}_{\mathbb{P}}}_{h}$ to $B^{\bar{f}_{\mathbb{P}}}$ ;
17     Set $Alarm$ to $True$ if a concept drift in the time series sequence collected in $B^{\bar{f}_{\mathbb{P}}}$ is
         detected.;
18     if $Alarm==True$ then
19         Set $s$ to $\{\hat{x}^{\bar{f}_{\mathbb{P}}}_{t+h-\omega}, \hat{x}^{\bar{f}_{\mathbb{P}}}_{t+h-\omega+1}, \cdots, \hat{x}^{\bar{f}_{\mathbb{P}}}_{t+h-1}\}$;
20         Learn a new aggregation policy $\pi(s|\theta)$;
21         The error estimators of the PH-test are restarted alongside the overall
             algorithm.
22     end
23 end
```

6.4 Empirical Experiments

In this section, we present the experiments that evaluate the performance of OEA-DRL for forecasting to answer the following research questions.

- **Q1**: How does OEA-DRL perform compared to the State-of-the-Art and existing online ensemble aggregation methods for time series forecasting?

- **Q2**: How critical the reward function choice is for the convergence of the reinforcement learning approach?
- **Q3**: What is the impact of the informed update of the aggregation policy in OEA-DRL in terms of computational resources compared to the most successful online ensemble approaches for forecasting?
- **Q4**: What is the benefit of using OEA-DRL as an aggregation method for pruned ensembles using our recently developed online pruning methods, namely DEMSRC, OMS-ROC-Ens, and OEP-ROC?

6.4.1 Experimental Setup

We conduct our experiments on the 100 real-world time series used to validate the previous methods in Chapters 4 and 5. They are also presented and described in the Appendix Table A.3. Each dataset is further split into training and testing sets via a $75\% - 25\%$ ratio. We evaluate various methods in terms of the Root Mean Squared Error (RMSE), and the error is used to create a ranked list of models according to their performance that serves as the evaluation metric in our analysis. The results are further assessed using the non-parametric Wilcoxon signed-ranked test [270] to compare pairs of models across the 100 datasets. Moreover, an embedding dimension of $p = 5$ is used for all the time series. The code is included under this repository [1].

We used the same pool $\mathbb{P}$ of 43 heterogeneous models as the one used for DEMSRC and OMS-ROC in Chapter 4. Except for addressing the research question **Q4**, a pool $\mathbb{P}$ of 33 candidate DNNs is considered, i.e., the same used for the online ensemble of DNNs pruning method OEP-ROC presented in Chapter 5. For clarity of notations and to avoid confusion with the pool $\mathbb{P}$ of the 43 heterogeneous models, we denote it by $\mathbb{P}_{DNNs}$ in the following. We apply first pruning to $\mathbb{P}_{DNNs}$ using each time one of these three methods: DEMSRC, OMS-ROC-Ens, or OEP-ROC. Afterward, selected models by each of the pruning methods are aggregated using OEA-DRL.

6.4.1.1 OEA-DRL Setup

In OEA-DRL setting, both policy and value networks are based on MLPs, which perform simple regression and multi-regression (M-dimensional vector of weights =43 or 33), respectively. In addition, a standard normalization is applied to the output of the policy network so that all the weights are positive and sum to one. The hyperparameters of OEA-DRL are tuned by model selection which results in the discount factor $\gamma = 0.9$, the learning rate $\alpha = 0.01$, and *max.ep* and *max.iter* equal to 100. The parameters of the Page-Hinkley test ν and ϱ are tuned using grid-search in $[0, 1]$ with a step size of 0.1.

6.4.1.2 State-of-the-art Methods

We compare the performance of OEA-DRL against several standard baselines as well as State-of-the-Art approaches: ARIMA [61], LSTM [118], StLSTM [113] (Stacked LSTM), RF

[1] `https://www.dropbox.com/sh/qgi26uhf5yd38a5/AABq4EZ32xMzVTCBK_KK3l5ba?dl=0`

[31], GBM [263], Ens [308], SW-Ens [12], EWA [180], FS [174], OGD [284], MLPOL [173], Stacking [33], OCL, OTOP, and DEMSRC. For more details about these methods, see Section 4.3.5.1. Note that our recently developed methods OCL, OTOP, and DEMSRC presented in Chapter 4, even though they are specifically designed for ensemble pruning, they use an adaptive aggregation schema that sets the ensemble weights using an inverse measure of the recent loss of each ensemble member on a time-sliding window of observations. That is why we compare them to OEA-DRL.

6.4.1.3 OEA-DRL Variants

We also added two variants of OEA-DRL.

- SRL: A method that uses Deep RL for learning linearly weighting aggregation policy offline. The policy is kept static at test time. This is the static version of OEA-DRL developed and presented by the author of this thesis in [36].
- OEA-DRL-Per: is a variant of our method where instead of updating the policy in an informed manner following concept drift detection, the policy is updated blindly in a periodic manner with periodicity for each upcoming 10% data points.

6.4.2 Comparing OEA-DRL to the State-of-the-Art

We first evaluate the rank of all the methods across different time series datasets. Table 6.1 shows the average rank achieved by each model and the standard deviation of the rank across various time series (the lower, the better: a rank of one means that the model is the best performing on all datasets) and the pairwise comparisons between OEA-DRL and the baseline approaches using the non-parametric Wilcoxon signed-ranked test [270]. The results exhibit the number of wins and losses of OEA-DRL compared to the other methods in the table. The numbers in parenthesis represent significant wins/losses with a probability above 95%.

Table 6.1 illustrates that our approach achieves the best performance among the evaluated methods. Hence, OEA-DRL outperforms the baseline methods in terms of wins/losses in the pairwise comparison, however not OEA-DRL-Per. The approaches that are based on combining individual forecasters, including the online ones, e.g., SE, SWE, EWA, and FS, and common ensemble methods, such as RF, GBM, StLSTM (Stacked LSTM), and Stacking, show inferior performance compared to OEA-DRL. ARIMA and LSTM, State-of-the-Art methods for forecasting, have a considerable difference in the average rank compared to OEA-DRL as well. The most competitive approaches to our method are DEMSRC and SRL, which perform well in the pairwise comparison. Nevertheless, both attain a higher average rank. OEA-DRL includes online updates of the aggregation policy/strategy, while SRL is devised offline. The policy is learned once on historical data, and only policy prediction is performed online in a step-wise manner. Per-OEA-DRL outperforms OEA-DRL. However, the number of significant losses of OEA-DRL against Per-OMS-ROCis relatively small, and the difference in their average rank is also small. This answers the research question **Q1**.

Table 6.1: Pairwise comparisons between OEA-DRL and baseline methods averaged over all the 100 datasets ($\omega = 10$).

Method	Pairwise Losses	comparison Wins	Avg. Rank
ARIMA	18(8)	81(76)	15.75 ± 5.2
RF	13(2)	87(85)	16.35 ± 3.5
GBM	11(0)	89(88)	16.61 ± 2.0
LSTM	24(0)	76(72)	13.43 ± 5.5
StLSTM	11(0)	89(88)	14.98 ± 3.4
SE	25(5)	75(70)	10.95 ± 2.5
SWE	14(2)	86(85)	11.85 ± 2.3
EWA	25(7)	75(70)	7.44 ± 3.3
FS	26(9)	74(69)	7.15 ± 2.8
OGD	30(11)	70(69)	6.48 ± 4.3
MLPOL	32(14)	68(66)	6.14 ± 2.5
Stacking	12(0)	88(86)	14.35 ± 3.5
OCL	44(21)	67(65)	6.85 ± 6.0
OTOP	30(12)	70(69)	9.01 ± 3.2
DEMSRC	37(25)	65(55)	4.13 ± 3.8
SRL	38(23)	62(54)	3.01 ± 3.7
OEA-DRL-Per	53(46)	47(39)	$\mathbf{2.15 \pm 3.8}$
OEA-DRL	-	-	$\mathbf{2.35 \pm 3.9}$

6.4.3 Importance of the Rank-based Reward Setup

In the next experiment, we study the convergence of the deep actor-critic Algorithm 4 using two different settings for the reward function. Figure 6.1 (top) shows the results for the reward defined as 1−NRMSE, where NRMSE is the normalized RMSE of the computed ensemble using the corresponding action (i.e., weights) on $X_{t+1-\omega:t}$, while Figure 6.1 (bottom) uses the reward as defined in Equation 6.5. The same setup of OEA-DRL is applied for the second approach using the NRMSE as a reward function. As it is mentioned in Section 6.3.2.1, Algorithm 4 does not converge using the first definition of the reward since the magnitude of forecasting errors does not only depend on the models but is also changing over the course of time. Thus, the choice of the reward function is critical for the convergence of the actor-critic DRL algorithm. This answers the research question **Q2**.

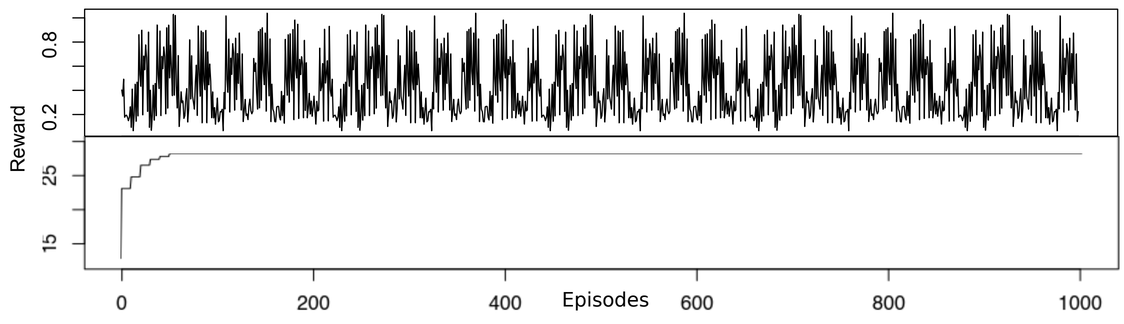


Figure 6.1: Learning curves of Algorithm 4 [301] with two different reward definitions. (a) Reward computed using $1 - NRMSE$. (b) Reward computed using Equation 6.5. On the x-axis is the number of episodes. On the y-axis is the average reward over each episode.

Table 6.2: Number of times of policy update comparison between Per-OEA-DRL and OEA-DRL.

Method	Number of times of policy update
Per-OEA-DRL	57.33 ± 10.05
OEA-DRL	5 ± 3.55

Table 6.3: Empirical run-time comparison between SRL, OEA-DRL-Per, and OEA-DRL.

Method	Avg. Run-time in sec.
SRL	217 ± 30.85
OEA-DRL-Per	10314 ± 1908
OEA-DRL	911 ± 621

6.4.4 Importance of the Drift-aware Policy Adaptation

OEA-DRL-Per relies on much more updates of the policy than OEA-DRL. Table 6.2 shows a comparison between the average number of updates of the ensemble aggregation policy between OEA-DRL-Per and OEA-DRL. While our method relies on informed updates that show that the previously learned policy is unable to deliver accurate ensemble construction that can deal with the current dynamics of time series data, OEA-DRL-Per is based on blind updates that are performed in a periodic manner with relatively high periodicity and therefore able to align better to small changes in the data that are not captured by the drift detection in the ensemble's performance. However, in terms of scalability, it can be seen in Table 6.3 that OEA-DRL-Per has a bigger run-time compared to OEA-DRL and to the static RL approach SRL, which can be considered as a reason for its applicability limitation to online forecasting scenarios that require small forecasting horizon. Similarly, even though OEA-DRL has a considerably lower run-time than OEA-DRL-Per, its applicability depends on whether the forecasting task horizon allows for the retraining of the two deep neural networks, namely the actor and the critic networks to learn new aggregation policy (Algorithm 4). SRL[2] has a lower average run-time since the computation of the policy is done once offline (i.e., we run Algorithm 4 only once offline and keep the policy network parameters fixed afterward) and never gets updated in the online forecasting phase. It only outputs predictions of the action $\boldsymbol{a}$ using the updated state s that contains the recently acquired ω time series observations but with the parameters θ fixed. This answers the research question **Q3**.

6.4.5 Combining Pruning Methods with OEA-DRL

Last but not least, we investigate how our recently developed online pruning methods, namely DEMSRC, OMS-ROC-Ens, and OEP-ROC can be joined together with OEA-DRL. Hence, once the pruning is achieved by one of these three methods (i.e., final individual models to appear in the ensemble are selected), comes the decision on how to aggregate their outputs at each time instant into one single output. This decision is made by the learned ensemble weighting policy in OEA-DRL. Since the only requirement to use OEP-ROC is to input a pool of CNN-based forecasting models, we use the pool $\mathbb{P}_{DNNs}$ of candidate individual models for all the methods (see Section 6.4.1). Table 6.4 shows the average ranks and respective standard deviations of different pruning methods using an averaging technique to combine the individual models at one time, and combined with OEA-DRL that takes care of the aggregation stage at

[2]An RL-based aggregation method that we have developed and published prior to OEA-DRL in [36]

a second time. The notation *MethodX*-OEA-DRL stands for one of the methods DEMSRC or OMS-ROC-Ens or OEP-ROC combined with OEA-DRL. Note that our pruning methods include different drift detection mechanisms that update the selection of the ensemble members, i.e., the individual models. With each of these updates, we also update the aggregation policy in OEA-DRLas new input individual models are present.

Table 6.4: Comparison of OEA-DRL combined with different pruning methods for 100 time series datasets.

Method	Avg. Rank
DEMSRC	4.20 ± 1.74
DEMSRC-OEA-DRL	4.03 ± 1.95
OMS-ROC-Ens	3.83 ± 2.12
OMS-ROC-Ens-OEA-DRL	3.49 ± 2.24
OEP-ROC	2.95 ± 1.92
OEP-ROC-OEA-DRL	2.18 ± 1.75

It can be seen from this table that all the pruning methods achieve better results when combined with OEA-DRL compared to simply using a weighted average aggregation. For DEMSRC and OMS-ROC-Ens, the difference in the average rank is not big compared to DEMSRC-OEA-DRL and OMS-ROC-Ens-OEA-DRL, respectively. This can be explained by the fact that opposingly to OEP-ROC, DEMSRC and OMS-ROC-Ens are not specifically designed to prune an ensemble of DNNs, and the quality of the pruned ensemble naturally has some impact on the aggregation stage. Hence, OEP-ROC-OEA-DRL achieves the lowest average rank, highlighting thus the need for both adequate pruning and combination methods that cope with the nature of the task, the data, and candidate individual models. It should be noted that the combination of these methods in one framework increases the forecasting solution complexity and computational resources. Therefore, the choice of opting for the combination should be based on a trade-off between accuracy and computational costs, which is application-dependent, e.g., in short-horizon forecasting applications, computational resources and execution time should be taken into account and managed efficiently. This is out of the scope of this thesis but will make the subject of future work.

6.5 Concluding Remarks

We introduced OEA-DRL: a novel and practically effective online ensemble aggregation framework for time series forecasting that employs a Deep Reinforcement Learning approach as a meta-learning technique. We used an actor-critic algorithm, in which the actor is taught to learn how to determine the best set of weights given prior experiences on individual model aggregation in the ensemble. The aggregation policy is updated in an informed manner following concept drift detection in the performance of the ensemble constructed based on previously learned policy.

An extensive empirical evaluation of OEA-DRL on 100 real-world datasets demonstrated that OEA-DRL outperformed multiple baseline algorithms and achieved competitive performance with the State-of-the-Art.

In the next Part of this thesis, we present three applications of model-based quality prediction of industrial processes. Some of the time series forecasting methods presented in the previous Chapters are transferred and applied to these applications.

Part III

Applications

7 Real-time quality prediction in NC-Milling

In this Chapter, we focus on the real-time model-based quality prediction of a milling process using Machine Learning. In mechanical engineering, milling is one of the most important machining operations with a wide variety of application use cases, e.g., the machining of structural components for the aerospace industry, dental prostheses, or forming tools in the context of tool and die manufacturing. Milling can be classified as a cutting operation using defined cutting edges. The material removal is performed by superimposing a rotational movement of the tool rotation axis and a translational movement of the tool relative to the workpiece defined by a Numerically Controlled (NC) path, where a chip per cutting operation is removed from the material. Figure 7.1 depicts an example of a pocket milling process.

7.1 Introduction

The optimization of milling operations is one of the most important topics in the context of production engineering [309], [310] since the capability to conduct high-performance cutting processes is indispensable for increasing the manufacturing efficiency in numerous branches of industry, e.g., to reduce tool vibrations in the aerospace industry [47] or to avoid tool wear for processes of the tool and mold manufacturing [311]. Due to different process states, which can change continuously throughout the progression of the regarded process, especially when tool wear characteristics can be expected for the considered process configuration, the monitoring of machining processes and real-time predictions of cutting tool conditions as well as the quality of the machined workpiece are vital for enhancing the manufacturing productivity. However, providing reliable and practicable solutions is still a challenging task for milling operations [312], [313]. The estimation of cutting forces is a crucial aspect since many process and tool-related characteristics are linked to them, such as tool wear [314].

Abrasive tool wear often reaches critical and non-negligible levels during the machining of hardened or difficult-to-cut materials and when long-running processes are required to

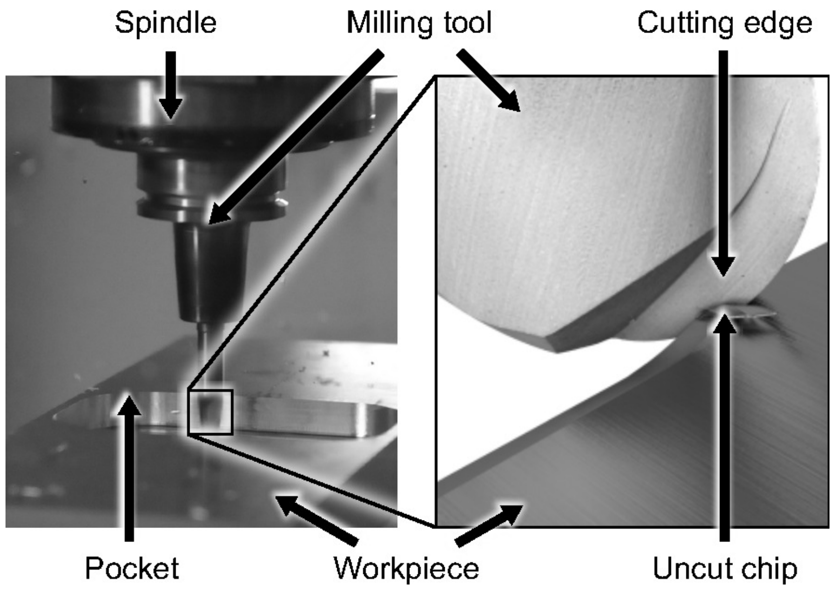


Figure 7.1: Pocket milling process.

manufacture the desired final contour of the component. Figure 7.2 shows a milling tool and optical measurements of a cutting edge for two different wear states, which are correlated by the number of machined slots using the regarded tool. Different values for the width of flank wear, i.e., the wear which occurred on the flank face of a cutting edge, are annotated in the figure and can be used as a measure for the wear state of the tool. The wear induced altered characteristics of the engagement situation between the tool and the workpiece can change the directions and amplitudes of the cutting forces affecting the tool, which can result in an alteration of the excitation of the dynamic behavior of the process and a shift of the stability border [311], [315]. In addition, tool wear can result in a deterioration of the quality of the machined workpiece surface, an increase in the temperature in the area of engagement, and a faster progression of tool wear. As a result, monitoring process forces in real-time is crucial as they enable the adaption of process parameter values [316], monitoring of tool conditions [317] and ensure high-quality and stable milling [318].

In this Chapter, we investigate a slot milling process by predicting active and passive forces in real-time using both sensor and simulation data. The use of simulation is motivated by the fact that it is usually expensive to collect enough data amounts using real-world sensors. In addition, in the case of NC-milling, geometrically-based simulations are able to generate additional information about the process that can not be measured directly using sensors in a reasonable amount of time [47].

Simulations are most often considered as the ground truth because they are based on theoretical knowledge of the particular application domain. Hence, they are used for testing learned models or annotating the data. Simulations are used as experiments that offer scientists the chance to study a range of phenomena in a structured way. This is a standard procedure in computer science [230] as well as in engineering [47], and physics [231], to name but a few. Many investigations of multi-sensor fusion have used simulation environments only for testing their frameworks [232], [319]. Process simulations are also established instruments to investigate processes in a virtual environment prior to deployment [47], [237]. More recently, simulations have been conceived as powerful data generators, providing promising opportunities

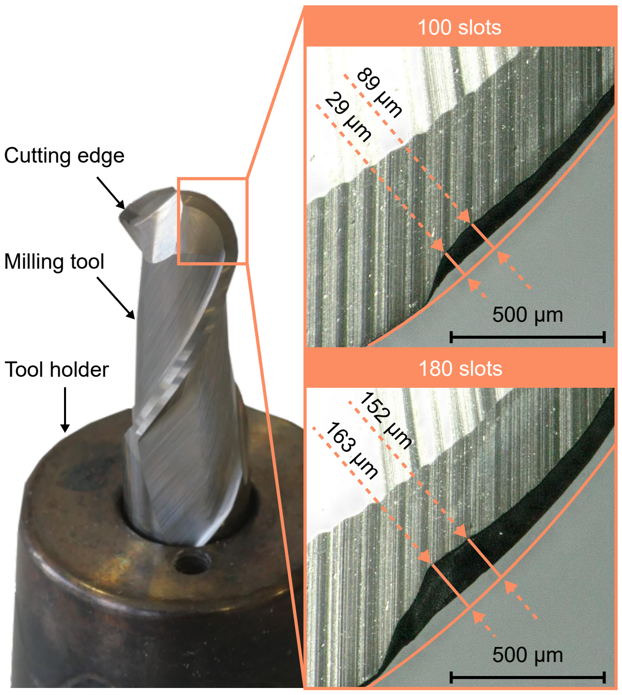


Figure 7.2: Comparison between tool wear states after milling 100 slots and 180 slots.

for simulation data mining [233]. However, simulations have many limitations in practice. They cannot provide a completely accurate representation of reality in real-time [234] and usually have only a limited prediction accuracy when modeling complex relationships [235]. For example, Finite-Element (FE)-based simulations, which are widely used in many mechanical applications, are intensively resource-consuming and not adequate for real-time application [234]. In addition, analytical approaches often provide only a limited prediction accuracy when modeling complex relationships [235], [320].

In contrast, ML models can be applied in real-time and offer the opportunity to predict events based on the analysis of a set of explanatory variables. Therefore, new trends aim at replacing simulation models with surrogate ML models that have been trained on simulation data [236], [237]. Some recent works focus on learning from simulation data to monitor the real-world process and predict upcoming unknown events with a reasonable accuracy [229], [237]. However, none of these approaches used simulation data to enrich sensor data for a given learning task.

In this context, we present a framework that performs simulation-sensor data fusion. First, it formally validates the use of simulation as a synthetic data generator in NC-milling by fulfilling certain conditions, namely *completeness*, *conciseness*, and *correctness*. It also suggests solving possible data mismatches between sensors and simulation using ML-based methods. Additionally, our framework challenges the typical data and model handling for ML applications in the sense that it allows for more flexibility in the fusion of sensor data and simulation. In particular, it automatically selects between two different integration levels.

Data-level fusion The integration of sensor observations and simulation is not restricted to the raw data of observations. Instead, a common representation for both is created and then enhanced by feature extraction, generation, and selection. The performance of

Table 7.1: Experimental process parameter values.

Tool diameter	$D = 10\,\text{mm}$
No. of cutting edges	$z = 2$
Depth of cut	$a_p = 0.2\,\text{mm}$
Width of cut	$a_e = 0.2\,\text{mm}$
Inclination angle	$r_\theta = 30°$
Tooth feed	$f_z = 0.06\,\text{mm}$ or $0.12\,\text{mm}$
Cutting speed	$v_c = 100$–$500\,\text{m/min}$

the model which is trained on these enhanced data is the criterion that guides feature engineering.

Model-level fusion Simulation and sensor data can be used independently for training single models, one on each data source. The resulting models may then be combined to output the prediction using an ensemble model. Weighting the two models adapts the framework to the application at hand.

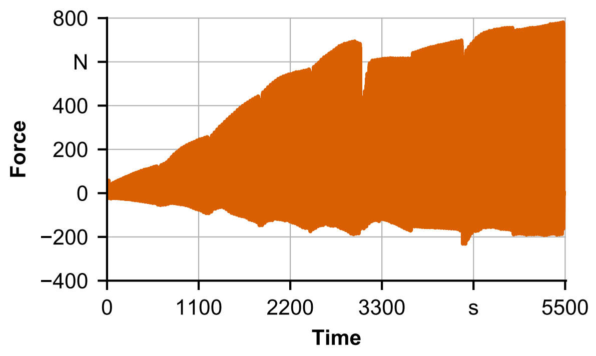
Level choice decision Moreover, the framework automatically decides which level to take based on a derived threshold for the co-variance of the single models. The criterion states that the variance-based error will be reduced by transiting from the data to the model fusion level if this co-variance is lower than the threshold.

The fusion framework is applied to an NC-milling use case for real-time cutting forces prediction where force sensors deliver online streaming time series measurements of the process forces during the conduction of milling experiments, and a geometric physically-based simulation system is used to generate pre-calculated features that cannot be measured by sensors. In addition, once the adequate fusion level is decided automatically using our fusion framework, we apply the online forecasting methods we presented in the previous Chapters.

7.2 Use Case Description

Slot milling experiments were conducted using a ball-end milling tool. For the workpiece material, the high-speed steel 1.3344, with a hardness of 62 HRC, was used, resulting in the occurrence of abrasive tool wear mechanisms throughout the process, which influence the amplitudes of the process forces. The parameter values of the process are summarized in Table 7.1. We vary the values of the cutting speed in 100–500 m/min with a step of 100 m/min. For each value of the speed, two different configurations for the tooth feed were used: 0.06 mm and 0.12 mm, resulting in a total number of 10 different process scenarios or configurations. For each scenario, a time series of cutting forces were measured using a force dynamo-meter Kistler 9257B, which was mounted on the machine table. The forces were measured in a three-dimensional coordinate system using a sampling frequency of 20 kHz, resulting in three force signals, denoted as F_x, F_y and F_z.

Due to the characteristics of the interrupted cut and the low immersion values, there are periods between each tooth engagement visible with no occurring process forces. In addition, there are also no forces during the approach movement between the two slots. As a consequence, two different process states are distinguished: an engagement between the cutting tool and the



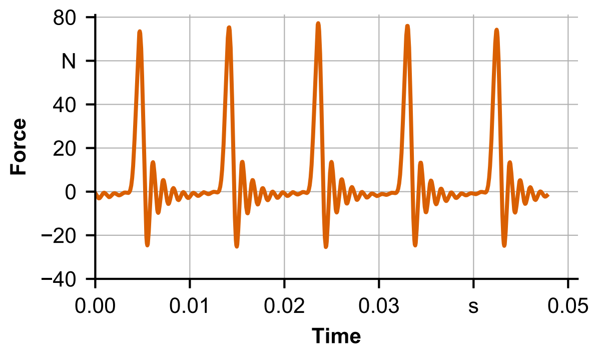


Figure 7.3: Measured process forces in z-direction using a cutting speed of 500 m/min and a tooth feed of 0.06 mm depicted through two perspectives: (a) the total time series, which results from 180 milled slots and is approx. 90 min in duration and (b) five tooth engagements.

workpiece as well as periods of no engagement. To reduce the amount of data, which has to be processed, the active forces F_a^i and passive forces F_p^i for each tooth engagement i are used for the learning procedure and calculated at time t as:

$$F_{a,t}^i = \sqrt{|F_{x,max}^i - F_{x,min}^i|^2 + |F_{y,max}^i - F_{y,min}^i|^2} \tag{7.1}$$

$$F_{p,t}^i = |F_{z,\text{max}}^i - F_{z,\text{min}}^i| \tag{7.2}$$

where $F_{x,min}^i$, $F_{x,max}^i$, $F_{y,min}^i$, $F_{y,max}^i$, $F_{z,min}^i$ and $F_{z,max}^i$ are the minimal and maximal values of the forces of tooth engagement i in x-, y- and z-direction recorded until time t, respectively.

The Forces are aggregated using an aggregation period of 0.1 ms and predicted each 0.1 ms for the next 10 ms. The resulting length of the time series for each scenario depends on the values of the input parameters and varies from 13250 to 54500 observations. Figure 7.3 shows the force progression using a cutting speed of 500 m/min and a tooth feed of 0.06 mm as well as forces for five tooth engagements of this time series. There are no sensor outages and the signal can clearly be distinguished from noise, indicating a suitable signal quality of the measurements.

For the simulation data, a geometric physically-based simulation system is used [47]. Due to the simplifications of the simulation models, simulation data can be generated in a feasible run-time, even for the investigated long-running machining process. A major advantage of the simulation approach is the capability of generating features that typically cannot be measured during the process. For this purpose, the chip volume, the sum of time of engagement, the feed, directions in the three-dimensional coordinate system, which corresponds to the coordinate system of the force dynamo-meter, and the mean of the cutting speeds, which can be calculated at different positions along the cutting edges of the ball-end milling tool, are generated for each point in time to potentially enrich the feature space of the forces measurements. The simulation is therefore used for feature enrichment.

7.3 Learning from Process Simulation

In addition to their major use as virtual test-beds [232], [321], [322], simulations are considered as data sources [233], [237]. However, generating data with simulations is a challenging process [237]. First, simulations are models for reality and they are not perfectly mirroring the exact real-world situation. Even though in many real-world settings, it is possible to rely on simulation models, the mismatch between simulation and real-world situations has to be solved in order to succeed the fusion and avoid data conflicts. Second, some simulation models, like Finite-Element based simulations [237], are intensive-resource consuming and not real-time capable [47], which makes data generation not possible in real-time. However, Fusion is still possible in real-time by using online streaming sensor data together with pre-calculated simulation data.

7.3.1 Major Uses

We can distinguish three major uses of simulations as data generators.

Additional observations generation Simulation is employed to generate more observations of a given process.

Additional features generation Simulation model is used to generate additional computed features that cannot be measured directly using sensors. In this case, simulations are used for feature enrichment. This can be done by using physical models directly or building surrogate ML models that learn the simulation model in order to generate new features for real-world data.

Data annotation Simulation is used as a data annotator to generate data labels. This scenario is mostly required when data annotation by sensors or human oracles is expensive.

7.3.2 Synthetic Data Quality Assessment

We suggest three criteria to which simulation data source should comply, namely *completeness*, *conciseness*, and *correctness*.

Completeness measures the amount of data in terms of the number of attributes and checks for missing values in each data instance. This operation can be performed on the batch data or instance per instance on a stream. Data instances with a high number of missing attributes are removed from the data source. If the number of missing attributes is relatively small, data imputation handles the missing values.

Conciseness measures the importance of an object/item representation in the data. Since we are mainly interested in data enrichment either in terms of features or in terms of creating new instances, we regard simulated data as the more important, the less redundant they are, and we give privilege to real sensor data in the case where simulation and sensor data are measuring the same properties of the same data instance.

Correctness measures whether the simulation data corresponds to the reaL-world. An interaction with domain experts is required to validate whether the simulation model is robust enough to generate data that can be useful for learning real-world process characteristics. Alternatively, Machine Learning itself can be deployed to validate the correctness of simulation data using approaches that have been developed in the context of deep generative models to assess the quality of artificially generated data [323], [324]. For instance, a distribution distance-based measure can be used to evaluate similarities between both simulation and sensor empirical data distributions. In addition, in case a simulation is used for the generation of additional observations, similar to what has been proposed for evaluating the quality of data sequences generated by a GAN model [324], a Train on Real sensor data and test on Simulation data (*TRTS*) and a Train on Simulation data and test on Real sensor data (*TSTR*) strategies can be used to validate the correctness and the usefulness of simulation data in learning an ML model.

Similar criteria can be checked for sensor data to ensure high-quality data acquisition from both sources. Same aspects for *completeness* and *conciseness* could be applied to sensor data. For *correctness*, erroneous measurements and downtimes in sensor data can be detected using anomaly detection techniques. Such measures enable solving data conflicts issues, namely uncertainty, and contradiction. In fact, *completeness* enables to avoid uncertainty which consists of a conflict between a non-null value and one or more null values that are all used to refer to the same characteristic of the same data instance. Hence, uncertainty caused by missing information (i.e., null values in a source or a completely missing attribute) is solved by checking *completeness*. In the meanwhile, *correctness* enables to solve contradiction which is a conflict between two or more *very* different non-null values that are all used to describe the same property of the same data instance. Contradiction is caused by different sources providing different values for the same attribute of a real-world entity. For sensors, erroneous measurements and downtimes are detected using anomaly detection techniques. Simulation *correctness* is assessed as described above. Further conflicts between both sources that may affect the *correctness* of the fused data are addressed by solving mismatches between simulation and sensors by means of synchronization and calibration. Further details are provided in Section 7.4.2.

For our case study, the simulation is used for additional feature generation as we explained in the previous section. An interaction with domain experts enabled us to verify that the geometric physically-based simulation fulfills the above-detailed criteria.

7.4 Simulation-Sensor Data Fusion

The proposed fusion framework is shown in Figure 7.4. The data acquisition process takes raw sensor data and acquires from the simulation the adequate data form depending on the use

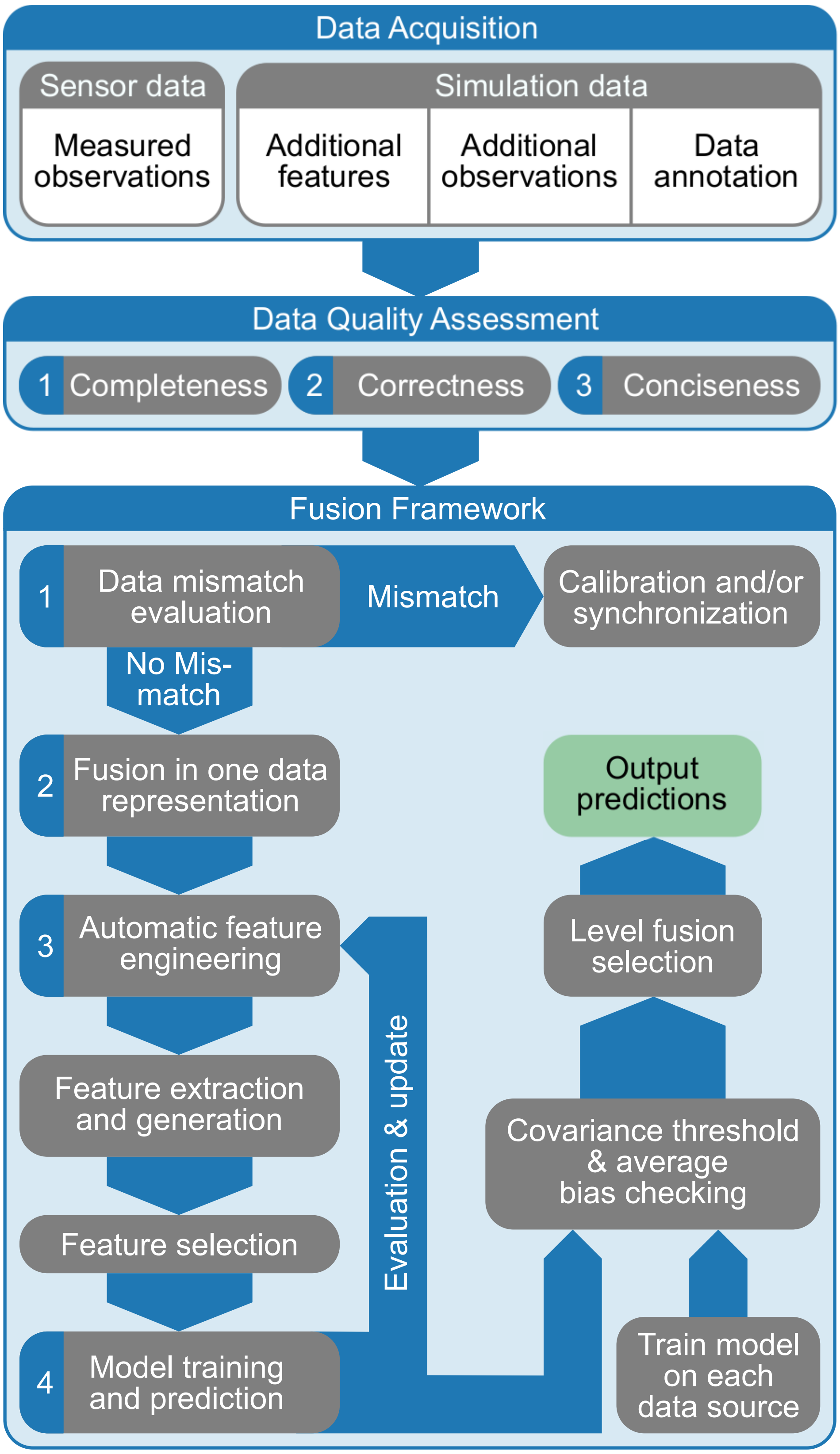


Figure 7.4: Simulation-Sensor Data Fusion Framework.

scenario of the simulation as a data generator. In our case, the simulation generates additional describing features for each data sample, i.e., in this case, a data sample corresponds to a milling experiment described by a set of input parameters.

Then, the quality of all sensor data and simulation data are checked according to three criteria detailed in Section 7.3.2. Simulation-sensor data mismatch is first solved. Feature engineering is used for enhancing the data quality by redundancy and correlations analysis, filtering strategies, new features generation, etc., that can be applied to data once data sources are fused into one unified data representation.

7.4.1 Data Quality Assessment

In case the simulation is used to generate features if the same feature is measured by the sensor, data quality criteria can be used to check which source is more reliable to consider and no further mismatches need to be solved. If the feature can only be generated by the simulation, the *correctness* criterion (Section 7.3.2) is sufficient to check mismatches with reality. In case the simulation is used as a data annotator, due to the absence of ground truth, one possible way is to interact with domain experts as has been already mentioned in the *correctness* criterion for synthetic data. Another possible way is to use an ML model to decide which instances to label. More precisely, an ML model can be used to determine which instances of real-world data are similar to simulation instances. If the similarity is high, the real-world data instance is to be annotated by the simulation (i.e., simulation is treated as an expert in instances that are similar to its own generated instances). In practice, this can be done by training a model to learn the labels using simulated data and using it to predict the labels of real-world data. Instances with high estimated prediction uncertainty are removed.

7.4.2 Simulation-Sensor Data Mismatch Solving

7.4.2.1 Synchronization

The fusion or the combination of data collected from both sources into a single data representation is not straightforward, and data mismatch checking between both sources is required, especially since both sources are heterogeneous and built by different paradigms. Data mismatch is not concerned with different data sources attributing different values to the same instance. A mismatch here is caused essentially by different data alignments due to different data sampling strategies and sampling frequencies of simulation and sensor data. The measurement frequency of a data source is defined by the number of times the source delivers data to the fusion framework per time unit. The simulation sampling frequency is set by domain experts and basically limited by the computational expensiveness. The higher the desired sampling frequency, the longer the simulation runs and the more memory it uses. The association and integration of sensors and simulation data require their synchronization with the environment description. The environment description is characterized by the adequate time of measurement. In order to obtain a precise synchronization, a sufficiently accurate global time of measurement for sensors and simulation and the fusion system is required to be defined or derived using synchronization techniques [325]. In fact, assuming a general sensor-simulation configuration, a data fusion system has to cope with different and varying

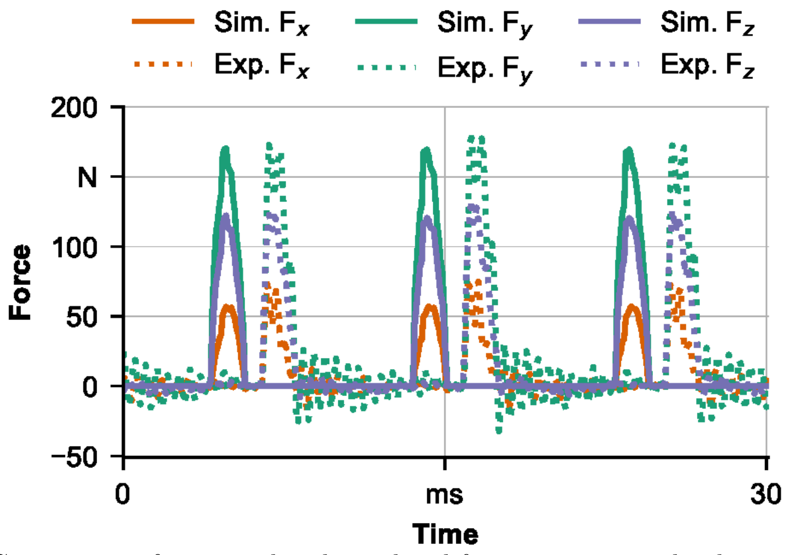


Figure 7.5: Comparison of measured and simulated forces in x-, y- and z-direction with a time-related delay between the time series.

measurement frequencies, measurement latency as well as asynchronous measurement times. Synchronization techniques can be divided into deterministic and non-deterministic ones. In a deterministic setting, the measurement times of each data source (i.e., sensors and simulation) have to be known in advance, and synchronization is performed on the slowest data source using aggregation techniques [325]. In a non-deterministic setting, simulation, and sensors are assumed to be asynchronous, and there is no knowledge about measurement times or latencies. The frequency can be non-constant. In such situations, recursive filtering approaches such as the Kalman filter or recursive auto-regressive filters can be used for synchronization [326].

For our case study, a time-related delay between simulated and measured data is observed. Therefore, data synchronization is required. Figure 7.5 shows a comparison between simulated and measured forces with a visible delay between them.

Both simulated and sensor data are acquired using the same sampling frequency so that only a constant time shift between each of the two time series has to be determined. This can be achieved, e.g., manually, using change points estimated by auto-regressive approaches [328] or by analyzing the continuous wavelet transform of the time series [327] (For more details about the transform See Section 2.1.3.3). Figure 7.6 shows a comparison between the transformed time series of measured and simulated forces of two tooth engagements using the Mexican hat wavelet

$$\Psi(t) = \left(1 - t^2\right) e^{-\left(\frac{t^2}{2}\right)}, \tag{7.3}$$

as mother wavelet [329]. By identifying the points in time where the intensity of the wavelet transforms at the tooth engagement frequency is greater than zero, the time-related delay between the two investigated time series can be identified.

7.4.2.2 Calibration

In addition, different hypotheses for drawing data samples from sensors and simulation may also lead to a mismatch in the sense of learning from different underlying distributions. Therefore,

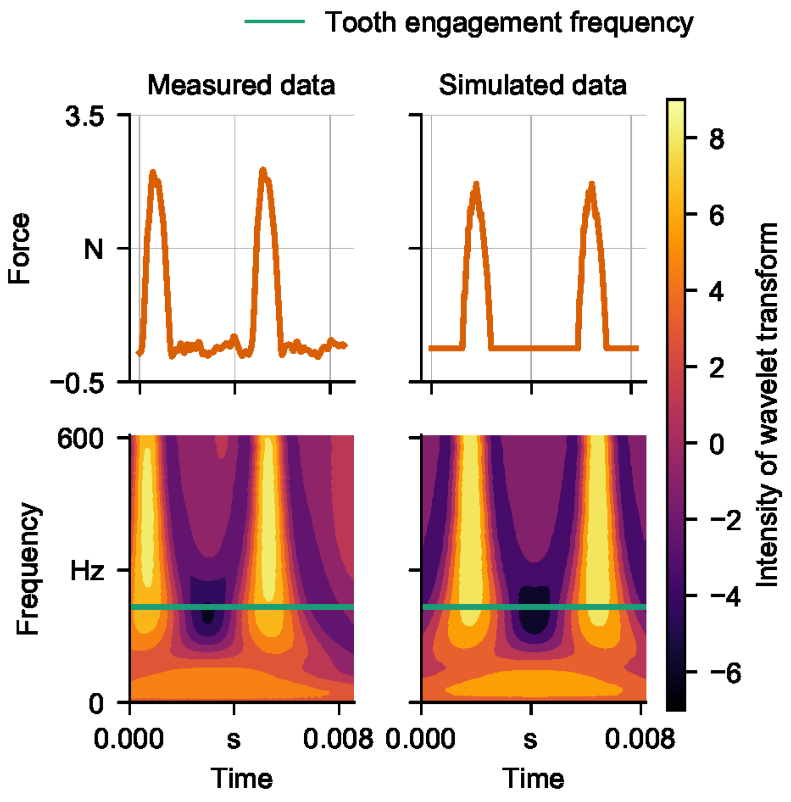


Figure 7.6: Wavelet transform of measured and simulated time series using the Mexican hat mother wavelet [327].

either sensor or simulation calibration is needed. In this setting, since one of the goals is to extract useful information about the real-world process, the calibration of simulation models on the real-world process is generally performed.

For example, in milling processes, the parameter values of different models inside simulation systems have to be calibrated on specific process configurations, e.g., the used workpiece material or tool geometry of the real-world process. Most often, simulation parameters are the ones to be calibrated or estimated as the posterior distribution to reproduce real data. Machine Learning is also useful in this context since the simulation calibration can be treated as a form of data assimilation. Suppose we use both sensors and simulation to monitor a given target variable y. The simulation model usually produces y as a function of some input variables x and simulation parameters θ. To perform the calibration, one of the possible ways would be to express the simulation model $f_{sim}(x;\theta)$ as a combination of a series of kernel mean embedding methods, where θ are the simulation parameters to be calibrated. When Gaussian noise is assumed with the function $f_{sim}(x;\theta)$, the likelihood is expressed as:

$$p(y|x,\theta) = \frac{1}{\sqrt{2\pi\sigma_0^2}} exp(\frac{1}{2\sigma_0^2} \|y - f_{sim}(x;\theta)\|^2), \tag{7.4}$$

where σ_0^2 is a constant of observation noise. More generally, the likelihood can be expressed as:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} (y_i - f_{sim}(x_i;\theta))^2, \tag{7.5}$$

where n is the number of observations, and x_i is the set of input variables corresponding to the output y_i. When the likelihood function is differentiable, a gradient-based optimization method can be solved to determine the simulation parameters θ. However, the likelihood function is most often non-differentiable owing to the simulation model $f_{sim}(x;\theta)$. In this case, the conventional statistical methods of parameter estimation are not applicable to simulation calibration owing to the properties of the likelihood function: intractable or non-differentiable. The posterior mean to be obtained is formulated as $p(\theta|x, y) = p(y|x, \theta)\pi(\theta)/Z(x, y)$, where $\pi(\theta)$ is the prior distribution and $Z(x, y)$ is a regularization constant. Hence, the simulation parameters can be estimated as a kernel mean of the posterior distribution by using, for example, kernel approximated Bayesian computation [330]. After computing the kernel mean of the posterior distribution, a posterior sample can be obtained using kernel herding [331].

For our case study, data calibration is required since measurement deviations between both sources are observed. In fact, due to the simplified models in the simulation system and the negligence of complex engagement behaviors, e.g., frictional effects, which are used to ensure a reasonable run-time, non-negligible deviations between simulation data and measurements of the corresponding process characteristics can occur, especially for process forces or tool vibrations. This deviation can differ for different process parameter values. Due to measurement noise and uncertainties, the quantification of simulation accuracy is a challenging task. Nevertheless, using measurements as ground truth, the simulation models can be calibrated on the used combination of the tool geometry and the workpiece material. For simulated forces, for example, the parameters θ of the force model $f_{\text{sim}}(t, \theta)$ (i.e., time series model) could be determined by applying an optimization procedure to minimize the squared Euclidean distance between simulated and measured forces with sensors $f_{\text{sen}}(t)$ (i.e., equivalent to maximizing the likelihood in Equation 7.4).

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \left(f_{\text{sen}}(t_i) - f_{\text{sim}}(t_i, \theta)\right)^2 , \tag{7.6}$$

The data is acquired using the same process parameter values for both the machining process and the corresponding simulation conduction. In this context, several optimization algorithms can be applied to solve the minimization task. However, in practice, quasi-Newton approaches often outperform other methods. Using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) [332] optimization algorithm, for example, an approximation of the Hessian matrix H_k is estimated, which is updated at each iteration k of the procedure. According to Newton's method, the parameter values of the next iteration:

$$\theta_{k+1} = \theta_k + \alpha_k s_k \tag{7.7}$$

are given by the line search along the descent direction:

$$s_k = -H_k \nabla \mathcal{L}(\theta_k) \tag{7.8}$$

by estimating α_k through:

$$\hat{\alpha}_k = \underset{\alpha}{\operatorname{argmin}} \ \mathcal{L}(\theta_k + \alpha s_k). \tag{7.9}$$

The update of H_k is performed by adding a rank-two correction:

$$H_{k+1} = H_k + auu^T + bvv^T, \tag{7.10}$$

$$u = \delta_k = \theta_{k+1} - \theta_k, v = H_k\gamma_k = H_k\nabla\mathcal{L}(\theta_{k+1}) - \nabla\mathcal{L}(\theta_k) \tag{7.11}$$

are typically chosen so that the quasi-Newton condition:

$$H_{k+1}\gamma_k = H_k\gamma_k + auu^T\gamma_k + bvv^T\gamma_k = \delta_k \tag{7.12}$$

is satisfied, resulting in:

$$H_{k+1} = H_k + \frac{\delta_k\delta_k^T}{\delta_k^T\gamma_k} - \frac{H_k\gamma_k\gamma_k^T H_k}{\gamma_k^T H_k\gamma_k}. \tag{7.13}$$

7.4.3 Unified Data Representation

After checking and solving mismatches between both data sources, the second step consists of unifying them into one unified representation, preparing thus for feature engineering in the following step. The data representation is decided based on the use scenario of the simulation as a data generator. If the simulation is used for generating additional features, these features are added as new columns to the dataset. It is important to mention that in this case, some of the features measured by sensors can also be generated by simulations. However, this format of redundancy will be solved in the feature engineering step by removing redundant features. If the simulation is used to annotate the data, annotations are just an additional column. If the simulation is used to generate new data instances, these are just new rows of the dataset.

In our case study, due to the difficulty of taking tool wear mechanisms into account reasonably for the calculation of process forces using a geometric approach for the representation of the tool-workpiece-engagement, simulated forces are not accurate for the regarded milling process and, therefore, not used as additional features for the learning task in this contribution. We rely exclusively on the measured real forces by the dynamo meters. Nevertheless, simulation is used to generate additional process features. In fact, utilizing the feed directions, the determination of the engagement time, and the cutting speed, generated by the simulation system is analytically accurate. Using the NC path of the milling process and modeling the tool using the corresponding tool properties, e.g., the tool diameter and the number of cutting edges, are also necessary. The simulated chip volume is chosen to be qualified as an additional feature since it contains crucial information about the engagement situation between the tool and the workpiece, which is not represented by any other simulated or measured feature. Therefore, after solving possible mismatches between both sources, a unified feature set was created by joining new features generated by the simulation together with the lagged sensor measurements of the cutting forces as sensor features. The target is simply the future time series observations values of both active and passive forces (Equations 7.1 and 7.2).

7.4.4 Automated Feature Engineering

One of the issues in data fusion while reducing data mismatch is underestimating the fused objects' co-variance due to the existing correlations inside one data source as well as inter-source correlations [333]. This leads to redundancies and sub-optimal knowledge extraction. In this context, feature engineering appears as one of the efficient ways not only to improve the accuracy and efficiency of a learning model but also to succeed in the fusion task since most of the preparation techniques of data fusion, such as filtering, redundancies, and correlation analysis, are the essence of features engineering paradigms [334], [335]. Feature engineering plays a key role in the success of ML algorithms that are trained to model the underlying relationship between a set of features and some defined target variable [336]. Feature engineering consists of the process of transforming raw data into features that provide an enhanced representation of the underlying problem to the predictive models, resulting in an improved model accuracy on the unseen data using a variety of operations. These operations include feature transformation, generation, extraction, selection, and evaluation. The traditional approaches consist of manual feature engineering to build features once at a time using data analysis and domain knowledge. Such approaches are tedious, time-consuming, error-prone, and usually not adaptive to data changes. Hence, some approaches have been conducted for automated feature engineering [337] and some tools have already been published [338]. Many widely used programming languages and software for data mining and Machine Learning, support tools, and libraries for automated feature engineering. For example, `RapidMiner` [339] enables automated feature engineering using *Automatic Feature Engineering operator* based on a multi-objective evolutionary algorithm. `Python` supports *featuretools*, which is an open-source library for feature engineering based on "Deep Feature Synthesis" [338]. For a principled view of fully automated feature engineering for time series, see [49]. The user can decide on the adequate approach by evaluating its *accuracy* using validation techniques and its *stability*. The *accuracy* of an approach is decided based on feature analysis and evaluation, which is a crucial step for assessing the usefulness of selected features in relation to the performance of the ML model in question, also giving feedback to the user about the choice of the feature engineering method. The *stability* of a feature selection method refers to the robustness of its features preference, with respect to data sampling and to its stochastic nature [340]. An algorithm is 'unstable' if a small change in data leads to large changes in the chosen feature subset. Therefore, a desired property in feature selection is to be 'stable'. Many quantifications of *stability*, each with different motivations and justifications, have been proposed in literature [340].

7.4.5 Model Learning

In our use case, we use 5 lagged values for the time series of the forces as sensor features. For each time step t, lagged sensor values are joined together with the simulation features for the current time step (i.e., simulation features were pre-calculated and stored, and only sensor data was streaming). In addition to these features, we have also devised a binary feature based on the simulation features called '*activity feature*', which indicated the engagement situation of the tool (0: no engagement, 1: engagement) and is added to the fused set of features. The target variables are the values of the forces F_a and F_p at $t + h$. As mentioned in Section 7.2, the Forces are aggregated using an aggregation period of 0.1 ms and predicted each 0.1 ms for the next 10 ms. So, in this application, the forecasting horizon h is set to 10 ms, which is different from the step size between two subsequent observations and the forecasting frequency, i.e., 0.1 ms. However, using this joined set of features mapped to the target variables in the format of tabular data, the forecasting task can be approached as a general regression task as explained in Section 2.4.4.

7.4.6 Automated Fusion Level Selection

7.4.6.1 General Methodology

Once adequate features are extracted and selected. A model is trained on these features to solve a given learning task. We refer to this in our work as *data-based* fusion. However, this is not always sufficient to build a robust model with enriched data. Therefore, we propose to automatically select between the *data-level* and the *model-level* fusion. The *model-level* fusion builds an ensemble of individual models trained separately on each data source. Several approaches for data fusion use different fusion levels without giving any justification [319], [341], [342]. However, it is known that a good ML model should establish a trade-off between bias and variance. Such a statement is derived from the error decomposition schema of an ML model, but it gives a guide on how to automate the decision for the fusion level. In fact, given a learned model $\hat{f}$ that approximates an unknown true model f, the expected mean square error between the target variable $y = f(x)$ and the model's predictions on an unseen sample x, can be decomposed into bias, variance, and an irreducible error term :

$$\mathbb{E}\Big[\big(f(x) - \hat{f}(x)\big)^2\Big] = \mathrm{Bias}\,[\hat{f}(x)]^2 + \mathrm{Var}\,[\hat{f}(x)] + \sigma^2 \tag{7.14}$$

where $\mathrm{Var}\,[\hat{f}(x)] = \mathbb{E}[\hat{f}(x)^2] - \mathbb{E}[\hat{f}(x)]^2, \mathrm{Bias}\,[\hat{f}(x)] = \mathbb{E}[\hat{f}(x)] - \mathbb{E}[f(x)]$. The expectation ranges over different choices of the training set $\{x_1, \ldots, x_n,\, y_1, \ldots, y_n\}$, all sampled from the same joint distribution $\{P(x, y)\}$ and σ^2 is an irreducible error term that will be ignored in the following. One way to reduce the variance-type error is to use an ensemble technique based on bagging-based techniques [343] (See Section 2.6.4.1). Such a statement can also be derived from the ensemble error decomposition. For more details, see Section 2.6.1 where we showed that for an average-based ensemble model $\overline{f}$ equally weighted (i.e. $\overline{f} = \frac{1}{M}\sum_{i=1}^{M} f_i$, $f_i, i \in [1, M]$), the expected error can be decomposed into $\overline{Bias}$, $\overline{Variance}$, and $\overline{Covariance}$.

The variance in this decomposition is the average variance divided by the number of single models M. When M is big enough, the variance term will diminish. However, the averaged bias and co-variance should be taken into account while adding more and more models. In our setting, we are concerned with a small number of single models (i.e., mainly 2 one built on sensor data and the other on simulation data). In addition, the decision of the transition from a single model to an ensemble has to be made. Therefore, it is more straightforward to deal with the whole term $\frac{1}{M}\overline{Var} + (1 - \frac{1}{M})\overline{Covar}$, as a variance-type error for the ensemble model and the corresponding bias as the average bias of single models $\overline{Bias}$ since

$$Bias(\overline{f}) = (\mathbb{E}[\overline{f}] - f) = (\mathbb{E}[\sum_{i=1}^{M} \frac{1}{M} f_i] - f) = \overline{Bias}. \tag{7.15}$$

While empirically evaluating the decomposition of the expected error term for a given model, there is no rule or reference for deciding whether the variance term is high or not. It is only possible to compare variance and bias terms for two candidate models to decide which one is better. However, a decision for the transition from a *data-based* fusion to a *model-based* fusion should be based on the level of the expected variance-type error together with the expected bias of the *data-based* fusion model without the need to explicitly compare it with the ensemble model (i.e., without the need for computing the ensemble model, considering it as a single model and checking its variance). Therefore, we propose to compute the normalized versions of the expected variance type-error of the *data-based* fusion and the single models trained separately using each data source and derive a threshold for the co-variance between the single models that guarantees that the transition to the *model-based* fusion will reduce the variance-type error and check the average bias later.

Lemma 17 *Given a set of models $f_i, i \in [1, M]$ each trained on a single data source and a model f_{fus} trained using a data-based fusion approach. The model-based fusion using an averaged ensemble with equal weights contributes to reducing the variance-type error compared to the data-based fusion model if and only if the average co-variance between the single models is lower than*

$$\overline{Covar} \leq \frac{M}{M-1}(Var(f_{fus}) - \frac{1}{M}\sum_{i=1}^{M} Var(f_i)). \tag{7.16}$$

where M is the number of single models.

Proof 18 *$\Rightarrow$ Start with Equation 7.16 and rearrange to obtain the variance term in Equation 2.61 lower than $Var(f_{fus})$.*

$\Leftarrow$ Suppose that $\frac{1}{M}\overline{Var} + (1 - \frac{1}{M})\overline{Covar} \leq Var(f_{fus})$ and rearrange to obtain Equation 7.16.

Corollary 18.1 *Let f_{sim} and f_{sen}, two models each trained on simulation and sensors data, respectively. The model-based fusion using an averaged ensemble with equal weights has a lower variance-type error than the data-based fusion model f_{fus} if*

$$Covar(f_{sim}, f_{sen}) \leq \tau, \tag{7.17}$$

where the threshold $\tau = 2(Var(f_{fus}) - \frac{Var(f_{sim})+Var(f_{sen})}{2})$

Equation 7.16 complies with the decomposition in Equation 2.61, stating that a lower covariance is always desired to reduce the overall ensemble error. It also confirms that enforcing a degree of diversity between the ensemble members through low covariance is always favorable [148]. Furthermore, once the single models are trained and built, the average co-variance term can be estimated entirely without any knowledge of the true data labels or the real function f to be approximated. From a practical point of view, this result confirms the usefulness of using simulation for enriching data with samples that reflect different patterns than the ones observed with sensor data, either with new features that can not be measured by sensors or with observations that can not be detected with sensors.

A good ML model should establish a trade-off between bias and variance. The bias of the ensemble model will be equal to the average bias of the single models in case of equal

Table 7.2: Comparison between the NRMSE of predicted active and passive forces using different methods.

Method	**NRMSE** F_a	**NRMSE** F_p
$\mathrm{RF}_{\mathrm{Simu}}$	$15.29\,\% \pm 3.70\,\%$	$21.50\,\% \pm 11.15\,\%$
$\mathrm{RF}_{\mathrm{Sen}}$	$24.24\,\% \pm 5.95\,\%$	$33.63\,\% \pm 5.38\,\%$
$\mathrm{RF}_{\text{data-level fusion}}$	$9.95\,\% \pm 2.90\,\%$	$13.30\,\% \pm 6.30\,\%$

weights. So, it will achieve a lower bias than the single model with the highest bias. If the covariance between single models is lower than the derived threshold τ in Equation 7.17, the system then compares their average bias with the bias of the *data-based* fusion model to check if the variance-type error reduction with the average bias will establish a better bias-variance trade-off or not. if it is not, the system sticks to the *data-level* fusion. For instance, in the case where the covariance is lower than the threshold, a reduction in the variance type error is expected. However, we may have an averaged bias that is significantly higher than the bias of the *data-level* fusion model. In this case, we earn in terms of reducing the variance-type error, but the bias is altered by the ensemble, and the bias-variance trade-off doesn't hold anymore. This is possible, especially in the case of small-size ensembles (i.e., small number of single models, like our setting $M = 2$) where the weakness of one model can not be totally covered by the other ones.

7.4.6.2 Application to Simulation-Sensor Data Fusion in NC-Milling

In the following, we showcase how the automated fusion level selection is performed in our use case. To do so, in order to forecast the future values of the forces, namely F_a and F_p, over time, we select as an example the Random Forest (RF) [31] as a forecaster since it presented a relatively good performance compared to other forecasting models. However, a similar analysis can be conducted for any forecasting model. We used the $p = 5$-lagged values for each force as input to the forecasting model in addition to the features generated by the simulation and devised '*activity feature*' as exogenous variables (see Section 7.4.5). Since we have 10 different realizations of the milling process given by 10 different input parameters configurations, i.e., 10 different time series, in order to model the effect also of the input parameters on the future values of the forces so that the model learns to predict the forces even with unseen new input parameters, we use a 10-fold cross-validation procedure where 9 scenarios (i.e., time series) are kept for training and 1 scenario for testing for each fold. The prediction error is evaluated using the Normalized Mean Squared Error (NRMSE):

$$NRMSE = \frac{RMSE}{|Max_X - min_X|} \tag{7.18}$$

where Max_X and min_X are the maximum and minimum values, respectively, of the predicted time series X, i.e., in this case, F_a and F_p. The results, which can be seen in Table 7.2 show that the data-level fusion model outperformed single models trained separately on each data source. The data-level means, in our case, one single model, i.e., RF, is trained on the fused set of features from the simulation and the sensors. The error rates are reduced by 35% and 39% for F_a and F_p compared to the single source model with the lowest error rate for each case.

Table 7.3: Comparison between the NRMSE of predicted active and passive forces using data-level and model-level fusions.

Method	**NRMSE** F_a	**NRMSE** F_p
RF $_{\text{Feature-based fusion}}$	$9.95\,\% \pm 2.90\,\%$	$13.30\,\% \pm 6.30\,\%$
RF $_{\text{Model-based fusion}}$	$16.25\,\% \pm 4.27\,\%$	$19.69\,\% \pm 5.37\,\%$

Table 7.4: Comparison between different measures for the level fusion selection.

Measure	F_a	F_p
Covar ($\text{RF}_{\text{Simulation}}$, $\text{RF}_{\text{Sensors}}$)	21.18	4750.80
Threshold τ	109.92	-13092.95
Average bias ($\text{RF}_{\text{Simulation}}$, $\text{RF}_{\text{Sensors}}$)	782.50	13714.16
Bias (RF $_{\text{data-level fusion}}$)	121.21	3204.13
Var (RF $_{\text{data-level fusion}}$)	330.40	8614.31
Var ($\text{RF}_{\text{Model-level fusion}}$)	140.86	13367.52

The fusion level is set up automatically to the data-level fusion after empirically checking the threshold τ derived in Equation 7.17 for the covariance and the average bias of the single models trained separately on each data source. To show the validity of our decision and theoretical insights, we have conducted experiments using an ensemble of both single models only for comparison reasons. These results can be seen in Table 7.3 and show that the data-level fusion model outperformed the model-level fusion using an averaged ensemble, confirming the validity of our automated decision about selecting the fusion level.

Furthermore, examples of empirical evaluations of the covariance, the threshold τ derived in Equation 7.17, the average bias of single models, and the empirical bias of the data-level fusion model for the predictions of F_a and F_p are shown in Table 7.4 to describe how the fusion level decision is made in detail. In addition, the model variances of the model-level fusion (i.e., ensemble) and the data-level fusion are reported to show the validity of our theoretical insights, which indicated that the ensemble computation is not necessary and the decision can be made based on the covariance and average bias of single models. For F_a, the value of the covariance between single models is lower than the threshold, which guarantees that computing the ensemble model will reduce the variance type error, and this is confirmed by the reported variance values in Table 7.4. However, validating the covariance threshold is not sufficient. The average bias of single models should also be compared to the bias of the data-level fusion model. Comparing these values, it is clear that the model-level fusion will contribute to reducing the variance type error but will also alter the bias by increasing it by more than a factor of six. For F_p, the value of the covariance between single models was higher than the threshold, which indicated that computing the ensemble model will not help to reduce the variance-type error, and this is confirmed by the reported variance values in Table 7.4. In addition, the bias of the data-level fusion is lower than the reported average bias. This observation confirmed that the model-level fusion would not improve the variance or the bias.

7.5 Online Cutting Forces Prediction

In this stage, we construct a pool $\mathbb{P}$ of forecasting models composed of RF [31], MLP [268], GBM [263], GP [281], SVR [264], PPR [265], LSTM [118] and CNN-LSTM [115]. For a detailed

Table 7.5: Comparison between the NRMSE of predicted active and passive forces using different forecasting methods.

Method	NRMSE F_a	NRMSE F_p
GBM	$11.97\,\% \pm 3.92\,\%$	$15.42\,\% \pm 7.82\,\%$
RF	$9.95\,\% \pm 2.90\,\%$	$13.30\,\% \pm 6.30\,\%$
Ens	$9.37\,\% \pm 3.92\,\%$	$13.01\,\% \pm 5.21\,\%$
SW-Ens	$8.37\,\% \pm 5.28\,\%$	$12.19\,\% \pm 7.92\,\%$
KNN-RoC(5)	$8.25\,\% \pm 4.78\,\%$	$12.05\,\% \pm 6.32\,\%$
DEMSRC	$8.07\,\% \pm 5.28\,\%$	$11.95\,\% \pm 5.56\,\%$
OEA-DRL	$7.83\,\% \pm 4.38\,\%$	$11.34\,\% \pm 5.86\,\%$
OMS-ROC-Single	$8.11\,\% \pm 3.89\,\%$	$12.01\,\% \pm 7.81\,\%$
OMS-ROC-Ens	$7.27\,\% \pm 2.97\,\%$	$10.98\,\% \pm 7.62\,\%$

description of the models, see Section 4.3.5.1. A pool $\mathbb{P}$ of size 34 is obtained using different parameter settings for each model. We found that the adequate fusion level for each of the models in $\mathbb{P}$ is the data-level fusion like the case for the RF model shown in Section 7.4.6.2. Therefore, all the models in $\mathbb{P}$ are trained similarly to the way explained above for the RF model, i.e., using the set of fused features.

Next, we apply some of the online forecasting methods for single model selection as well as for ensemble learning that we presented in the previous Chapters, namely DEMSRC,OMS-ROC-Single, OMS-ROC-Ens, and OEA-DRL. Some State-of-the-Art SoA methods such as Ens, RF, GBM, SW-Ens, and KNN-RoC(5) are also used for comparison. Note also that further details about these methods can be found in the previous Chapters, and SoA methods are described in Section 4.3.5.1.

Table 7.5 shows a comparison between the NRMSE of predicted active and passive forces using these methods. The results of Table 7.5 demonstrate that our developed methods for online adaptive time series forecasting clearly outperform the SoA methods in this use case. This can be explained by the dynamic nature of the NC-milling process cutting forces that show a clear non-stationary behavior (See Figure 7.3) and change their patterns drastically when different tool engagement situations are present (i.e., especially in the change between engagement and no-engagement situations). Hence, this dynamic nature of the time series of F_a and F_p requires the use of online adaptive methods and justifies the success of our methods. It can also be seen that DEMSRC, OEA-DRL and OMS-ROC-Ens have relatively better performance than OMS-ROC-Single. In this case, the use of a heterogeneous ensemble schema seems to be beneficial. This can be explained by the fact that using a pool of different families of forecasting models in DEMSRC, OEA-DRL and OMS-ROC-Ens empowers the *diversity* aspect of the ensemble implicitly and contributes thus to a better performance by combining the predictions of well-selected experts efficiently by our methods. Hence, DEMSRC, OEA-DRL and OMS-ROC-Ens perform ensemble pruning and/or aggregation over a pool of 34 models. Finally, performing this online prediction task as accurately as possible is crucial for this application, where cutting forces predicted values are to be used in a sub-module for a recommendation system that will produce smart live recommendations for automatically adjusting the process parameter values to ensure good quality and stable milling process.

7.6 Concluding Remarks

In this chapter, a novel approach for real-time cutting forces prediction in NC-milling, which incorporates ML methods, is presented. To achieve improved prediction accuracy, a data fusion approach utilizing pre-calculated simulation features and measured data, which can be acquired by sensors in real-time, is developed. The simulation data is generated using a geometric physically-based simulation system, providing non-measurable process characteristics, which add additional knowledge and information for the analysis of the engagement situations between the tool and the workpiece while training the model. The fusion framework decides automatically for the adequate fusion level, namely data-level and model-level. In the presented use case, the data-level fusion is more efficient, and by fusing features from both simulation and measurements, improved accuracy can be achieved compared to incorporating only one data source. Using the developed methodology, cutting active and passive forces can be predicted in real-time with a reasonable forecast horizon in order to achieve the possibility of adapting or stopping the process even before unwanted events would occur. Some of the online adaptive forecasting methods presented in this thesis are applied successfully to this use case, highlighting thus the usefulness of online adaptive approaches in a complex non-stationary environment.

Further research will be conducted in the context of transferring these approaches to more complex engagement situations. In addition, the interaction between tool wear, process stability , and the online adaptation of process parameter values will be investigated in succeeding research activities.

8 Explainable Quality Prediction in Industrial Applications

In the context of applied ML to quality prediction, we investigate two additional use cases from Industry 4.0. Previously, we have developed approaches that provide explainability for ML model selection and performance. In this chapter, we dive further into ML models' output explainability for quality prediction-related tasks. Note the presented use cases do not involve a forecasting task, but one of them consists of a time series classification task. In addition, we investigate both of them to dive further into the use of the Grad-CAM method [203] (i.e., the method from which we got inspired to develop the Performance Gradient-based Saliency Maps-PGSMs) for tabular and time series data to serve explainability purposes.

8.1 Introduction

Nowadays, in the context of Industry 4.0, the linkage of production environments through Information and Communication Technologies (ICT) to cyber-physical systems with the goal of monitoring, controlling, and optimizing complex manufacturing systems, enables real-time capable approaches for process data acquisition, analysis, and knowledge discovery [8], [224]. This is achieved in practice by collecting and analyzing sensor data. As a result, data-driven approaches for predicting process quality in real-time and deriving adequate process control interventions in a timely manner can be developed [8], [45], [225], [229], [344]. In some cases, sensors are able to generate mass quantities of sensor data as responses to some types of inputs from the physical production environment. Most often, each of these data points is captured at specific time stamps, effectively transforming sensor data into time series data that can be analyzed across this additional dimension [8], [18].

Machine Learning algorithms trained on sensor data are able to ensure model-based quality prediction [224]. To do so, the description of the product or process quality and all the related information should be done in the first step, especially in highly complex dynamic production systems with non-linear interactions between their steps. The second step consists of building and training a predictive model that maps the available quality-related information, e.g., operating states, sensor data, and process input parameters, to the resulting process/product quality [18]. This model can be used afterward for predicting the expected quality given a set of input values.

More recently, DNNs have been successfully applied in the context of process quality prediction with high accuracy [229], [345], [346]. Their success is mainly due to their ability to learn new complex enriched feature representations in an automated manner from the input data [109]. Thus, they achieve a good performance in solving a wide variety of complex tasks such as quality prediction where the expected process/product quality is affected by multi-interacting features in complex manufacturing settings [8], [45], [224]. as we mentioned in previous chapters, DNNs are known to be complex black-box models that mostly give better accuracy in their predictions, at the cost of a low explainability [40]. However, likewise, high prediction accuracy, both interpretability, and explainability can be of the same importance and sometimes even more substantial for quality prediction-related applications [347]. Hence, understanding the model's predictions, i.e., "Why" a certain quality label is predicted, and the model's dynamics, i.e., which model's parameters are taken into account or if the model contains any bias, can be of great help for subsequent decision-making by process experts on process optimization, such as adjusting process parameters, or early stopping of the process if desired quality standards will not be reached, etc.

In this Chapter, we study the task of model-based quality prediction using DNNs for two industrial use cases, and we use explainability tools to provide an understanding of the process features that led to expected process quality. We also show how explainability results can be exploited by domain experts.

First, early quality prediction of a bolting process in a real-world automotive industry use case is performed using 1D-CNN. The quality of a bolt is described by eight discrete labels indicating whether it is "defective" or "non-defective". In the case of "defective", seven types of defects can be identified. The quality label is described by a time series feature. The learning task can formally be described as a multi-class Time Series Classification (TSC). We devised a training mechanism such that a 1D-CNN is trained on the training set composed of the full-length time series features. Then, a heat-mapping gradient-based explanation method, namely Grad-CAM is used to highlight the most important discriminative patterns on input time series on a validation set. These maps are used not only for providing explanations but also to determine where the discriminative patterns can be localized so that a reliable early quality prediction can be achieved by means of sampling, i.e., deriving a shorter length of the time series window to be used for early prediction of the quality. The 1D-CNN is retrained using the input time series with the newly derived length. We demonstrate how this training mechanism can be used to achieve better prediction accuracy using carefully selected time series subsequences on the described use case.

Second, Surface Mount Technology (SMT) in electronics manufacturing use case is presented. More specifically, we employ a 1D-CNN for quality prediction of well-defined Fields Of View (FOVs) of Printed Circuit Boards (PCBs). Explanations for the model's predictions are provided under various perspectives using a Grad-CAM to highlight the contribution of both local and global PCBs' characterizing features to the quality predictions. This helps to reveal the most decisive features for a given quality assignment and understand which process parts are the most responsible for such a decision. Finally, the deployment of the model-based predictive analytics and parts of the prescriptive analytics supported by the provided explanations is achieved using Edge Cloud Computing technology.

8.2 Related Works

Several industrial applications utilize already existing ML methods and algorithms to solve actual problems in manufacturing from an engineering point of view [224]. These applications cover a wide range of industrial fields, including electronics [224], metal [8], [45], [225], and process industries [229], [344]. Likewise, the adopted ML solutions are not limited to a specific family of models but include, amongst others, Artificial Neural Networks (ANNs) [229], Support Vector Machines (SVMs) [8], and Decision Trees (DTs) [45]. Most of the aforementioned works focused on quality prediction at the end of the production chain with the goal of reducing the costs of quality inspection by humans or special machines [8], [45], [224]. However, only a few studies have achieved and investigated the impact of early quality prediction in the very early stages of multi-staged processes or within the execution of the process [225], [229].

8.3 Explainable Early Quality Prediction in Automotive Manufacturing

In manufacturing systems that are organized according to the flow principle, quality deviations that are only detected at the end of the production chain may potentially result in high amounts of rejected products that require laborious and costly rework or need to be scrapped [224]. To prevent such events, early quality prediction has to be achieved. Hence, corrective actions are expected to have the largest impact if they are executed as early as possible in the process, avoiding thus costly rework and waste of resources through further processing of defective components [8], [45], [225]. A key requirement for early quality prediction is the full coverage of the product quality in the early stages of the manufacturing process or within the execution of some stages.

8.3.1 Use Case Description

Bolted connections are commonly used mechanical connections in industrial products. The non-linear tightening process of the bolts is generally distributed into four different zones that can be visualized using the torque diagram, which is a commonly used monitoring tool for the quality of bolted connections in the industry. The four zones are the rundown, alignment, elastic clamping, and post-yield zone. Figure 8.1 shows an example of these zones. Note that the torque is shown as a function of the angle of the turn. However, the angle itself is a function of time, and a nearly similar curve of the torque can be observed in the course of time.

By strengthening and plastic deformation of the bolts, a clamping force is generated between the connected parts. However, under realistic process conditions, many different complex factors influence the quality of the part connections: e.g., connected parts might move during the screwing, the equipment condition might be deficient, or the quality of the bolts might differ,

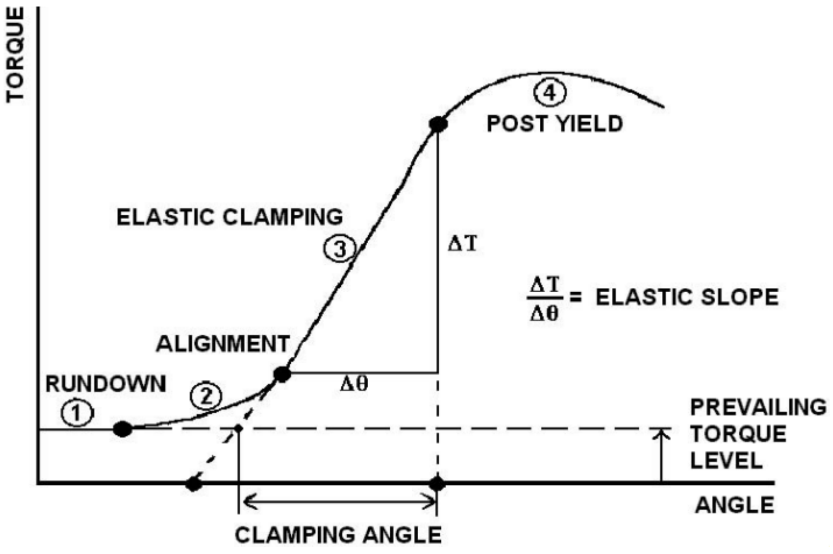


Figure 8.1: The Four zones of a bolt tightening process [348].

leading to quality problems in the final product. When the occurrence and the causes of the fault in the bolt connection are detected at an early stage in the process, countermeasures can be taken so that potential quality problems can ideally be avoided. However, monitoring methods are mainly based on the comparison of the torque with a preset standard value, ignoring the analysis of the torque diagram. As a result, various errors remain hidden. For this reason, the use of more sophisticated methods, such as data mining is desirable to achieve an improvement in quality analysis within bolting processes. A historical data set covering 233138 industrial bolting cases represented by time series torque sequences from the same workplace and product type is used for this case study. The data set consists of eight different classes, with one class representing the "non-defective" cases while the remaining depict seven different types of defects. Only 358 cases are labeled as "defective" which induces a highly imbalanced multi-class classification problem. The sensory data is described by a univariate time series feature of the normalized torque that is measured during the execution of the bolting process.

8.3.2 Modeling

8.3.2.1 Preliminaries

Let the input data denoted as a multi-set $\mathcal{D} = (\mathbf{X}, \mathbf{Y}) = \{(X_{1:t}^{(i)}, y^{(i)})\}_{1 \le i \le n}$ which consists of $|\mathcal{D}| = n$ input data points $X_{1:t}^{(i)}$ and their corresponding label $y^{(i)}$. Each input data point $X_{1:t}^{(i)}$ consists of a univariate time series $X^{(i)}$ recorded till time t. The time series subsequence of $X^{(i)}$ starting from time instant t and can be denoted as $X_{t:l}^{(i)} = \{x_t^{(i)}, x_{t+1}^{(i)}, \cdots, x_{t+l-1}^{(i)}\}$, where l is the length of the subsequence.

The goal of predictive ML models is to approximate some true function $f^* : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ denotes the input data space and $\mathcal{Y}$ is the target variable to be predicted space. In our case, the predictive ML model is built to predict a discrete quality label given an input univariate time series. Therefore, $\mathcal{X} = \mathbb{R}^d$ is the input time series feature space, where the dimension d can be equal to t if the whole recorded time series is used for training the predictive model or l if only time series subsequences of length l are alternatively used. $\mathcal{Y} = \{c_1, c_2, \cdots, c_j\}$ is the target variable that can take j possible values mutually exclusive. The learning task can be formalized as a multi-class TSC. The prediction of $\mathcal{Y}$ is carried out by the application of a model $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$. Usually, a fixed structure of f_θ is chosen that is parametrized by some vector $\theta \in \mathbb{R}^p$. Learning from data $\mathbf{X} \in \mathcal{X}$ consists then of basically fitting θ to X, so that $f_\theta \approx f^*$. Convolutional Neural Networks (CNNs) are a special class of these predictive models (More details about CNNs are provided in Section 2.4.5.6).

8.3.2.2 1D-CNN Architecture

The architecture of the CNN used in this work is inspired by [349]. The basic block is composed of a convolutional layer followed by a batch normalization layer and a ReLU activation layer. The the convolution operation is fulfilled by 1-D kernels. The blocks are repeated four times, varying at each time the number of applied filters and their corresponding length. The basic convolution block excludes any pooling operation to prevent overfitting. Batch normalization is applied to speed up the convergence. After the convolution blocks, the features are fed into a Global Average Pooling (GAP) layer. However, opposingly to [345], instead of directly connecting the GAP layer to the final softmax layer, we feed its output to three dense layers. Even though this strategy would lead to increasing the number of weights, it results in better generalization [120]. The final discriminative layer L takes the new representation of the input time series, i.e., resulting from the applied convolutions, and gives a probability distribution over the class variables in the data set using a softmax operation:

$$\hat{y}_k(X) = \frac{e^{f^{(L-1)} * w_k + b_k}}{\sum_{z=1}^{j} e^{f^{(L-1)} * w_z + b_z}} \tag{8.1}$$

with $\hat{y_k}$ denoting the probability of X having the class y equal to class c_k out of the j classes and w_k the set of weights (and the corresponding bias b_k) for each class c_k connected to each previous activation in layer $L-1$.

8.3.2.3 Grad-CAM for Extracting Explanations

Heat maps are one of the tools for providing visual explanations for CNNs, applied widely in computer vision-related applications [203], [208], [300]. Grad-CAM is one of the most popular methods for producing heat maps [120], [208].

Grad-CAM is applied to an already-trained neural network after training is completed and the parameters are fixed. We feed the time series input into the network to calculate the Grad-CAM heat map for that input belonging to a given class of interest. The output of Grad-CAM is a "class-discriminative localization map", i.e., a heat map where the hot part corresponds to a part in the input that is of the most importance for assigning a particular class to this input by the classifier. For example, in the case of images, this part corresponds to the most important regions in the image for classifying an object within the image. In our case, it highlights the input time series subsequence that is of high relevance for deciding the class of the whole time series. The length and the beginning of the subsequence are then automatically decided by the Grad-CAM.

In order to obtain this map for any class $c_k, \forall k \in [1, j]$, the gradient of y^{c_k} before applying the softmax (i.e., the score of the class c_k) with respect to feature maps $A_m, \forall m \in [1, f_{maps}]$ of the last convolutional layer, is computed. These gradients flowing back are global average-pooled to obtain the neuron importance weights:

$$\alpha_m^{c_k} = \frac{1}{Z} \sum_u \frac{\partial y^{c_k}}{\partial A_m^u} \tag{8.2}$$

where Z is the total number of units u in A_m. Note that in the $2D$ case, the activation unit u has $2-D$ coordinates $\{i, j\}$. In our case, it has only a $1-D$ dimension corresponding to the time dimension. This weight $\alpha_m^{c_k}$ represents a partial linearization of the deep network downstream from A_m, and depicts the importance of this feature map for the class c_k. Afterward, a weighted combination of forward activation maps is computed:

$$L_{Grad-CAM}^{c_k} = ReLU(\sum_{m=1}^{f_{maps}} \alpha_m^{c_k} A_m) \tag{8.3}$$

The ReLU is applied to remove the negative contributions since we are mainly interested in the input part that has a positive influence on the class of interest c_k, i.e., in the case of images, pixels whose intensity should be increased in order to increase the distinguishability of c_k. Without this ReLU, localization maps sometimes highlight more than just the desired class and achieve lower localization performance [208]. $L_{Grad-CAM}^{c_k}$ is used to find the subsequences in the input time series that has mainly contributed to the decision of the network for the class c_k.

The Grad-CAM computation can result in highlighted sparse time series data points. In order to highlight subsequences so that clear patterns can be observed and distinguished, we apply a smoothing procedure. Hence, for an input time series X belonging to the class c_k, i.e., $y = y^{c_k}$, we normalize the $L_{Grad-CAM}^{c_k}$ values between 0 and 1 and we apply a moving average smoothing procedure of window size 3. It should be noted that many subsequences can be highlighted in one single time series. This can be explained by the fact that different regions in the time series are of similar importance to the classifier, as it can be the case that the classifier needs the joint contribution of these subsequences to make its decision.

8.3.2.4 Important Time Series Subsequences Identification

The produced heat maps by the Grad-CAM can be used to identify the length and the temporal location of the most discriminative subsequences within the input time series. This would also help to check whether a reliable decision on performing early quality prediction can be made or not. In other words, these maps can be used to decide if the early subsequences in the input time series are, on average, the most important subsequences for correctly assigning the time series to its true class. To do so, we split $\mathcal{D}$ into two sets, $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$. $\mathcal{D}_{train}$ is used to train the 1D-CNN. $\mathcal{D}_{val}$ where the class labels for each time series are assumed to be known, is used for producing the heat maps. Note that by means of random data shuffling, the data split and the 1D-CNN training are repeated until we ensure the condition $\mathcal{C}_1$.

Condition 7 (*Condition $\mathcal{C}_1$*) *The condition $\mathcal{C}_1$ corresponds to the two following points:*

- *All the classes $c_k, \forall k \in [1, j]$ are represented in $\mathcal{D}_{val}$.*
- *At least one time series sample from each class is correctly classified by the 1D-CNN.*

Then, we select time series samples from $\mathcal{D}_{val}$ that are correctly classified by the 1D-CNN, i.e., time series samples $X^{(s)} \in \mathcal{X}_{\mathcal{D}_{val}}$ that fulfill the condition $\mathcal{C}_\mathbf{2}$.

Condition 8 (*Condition $\mathcal{C}_2$*) *Construct with $\mathbf{X}^{(s)}$ containing all the series $X^{(i)} \in \mathcal{X}_{\mathcal{D}_{val}}$ fulfilling:*

$$\hat{y}^{(i)}_{X^{(i)}} = y^{(i)} \tag{8.4}$$

For each selected time series $X^{(s)} \in \mathbf{X}^{(s)}$ with $y^{(s)} = c_k$, we compute $L^{c_k}_{Grad-CAM}$ to highlight the most important subsequence $X^{(s)}_{t_i:l}$ of length l starting at time t_i, for assigning the correct class membership to $X^{(s)}$. Then, we compute the average length l_a of all the computed subsequences starting from $t_i = t_0$, where t_0 is the initial instant of the process time series generation. That means we determine the furthest time point in each time series $X^{(s)} \in \mathbf{X}^{(s)}$ at which the last highlighted subsequence ends. These points are denoted by $t^{(s)}_{end}$. Then, l_a gets the average value of these points.

$$l_a = \frac{1}{|\mathbf{X}^{(s)}|} \sum_{X^{(s)} \in \mathbf{X}^{(s)}} t^{(s)}_{end} \tag{8.5}$$

Finally, we compare l_a to the average length l_m of the original full-length input time series in $\mathcal{X}_\mathcal{D}$:

$$l_m = \frac{1}{n} \sum_{X \in \mathcal{X}_\mathcal{D}} length(X) \tag{8.6}$$

We verify afterward if the condition $\mathcal{C}_\mathbf{3}$ is fulfilled.

Condition 9 (*Condition $\mathcal{C}_3$*) *The condition $\mathcal{C}_\mathbf{3}$ is given by:*

$$l_a < l_m \text{ and } l_m - l_a \geq \tau \tag{8.7}$$

where τ is the admissible time duration required to perform process optimization, e.g., corrective measures and process parameters adjustment. τ is a user-defined hyperparameter as it is application-dependent, and interaction with domain experts is required to set up its value. If the condition $\mathcal{C}_\mathbf{3}$ is fulfilled, it is possible then to reliably perform early quality prediction before the termination of the process. The 1D-CNN is then retrained on the input time series subsequences of $\mathcal{D}_{train}$ of length l_a. This operation can be viewed as an input feature selection where only l_a input time series points are fed to the 1D-CNN. The above steps are summarized in Algorithm 6.

8.3.3 Quality Prediction Results

8.3.3.1 Experimental Setup

The available data set is split into 75% for training and 25% for testing. The 75% are split into 75% for $\mathcal{D}_{train}$ and 25% for $\mathcal{D}_{val}$. A 10-fold cross-validation procedure is employed for the evaluation of our method. The reported results are the averaged metrics values over all the folds. Since the data set reveals high imbalance ratios towards some classes, both random over-sampling and under-sampling are employed to mitigate this issue [350]. More precisely, we under-sampled the "non-defective" class and over-sampled the seven different defect classes. In addition, the time series data have different lengths. Zero-padding is employed to bring the

Algorithm 6: Important **T**ime **S**eries **S**ubsequences **I**dentification: **ITSSI**

```
   Data: Data set: $\mathcal{D}$; Admissible time duration for process adaption: $\tau$.
1  Repeat $\mathcal{D}$ split into $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ and the 1D-CNN training until $\mathcal{C}_1$ is fulfilled. ;
2  Select time series samples from $\mathcal{D}_{val}$ that fulfill $\mathcal{C}_2$ and put them in $\mathbf{X}^{(s)}$.;
3  for $X^{(s)} \in \mathbf{X}^{(s)}$ do
4      We compute $L^{y(X^{(s)})}_{Grad-CAM}$ to highlight $X^{(s)}_{t_i:l}$.;
5      Set up all the $t_i$ to $t_0$.;
6      Calculate the lengths of the subsequences starting from $t_0$ including the computed
         $X^{(s)}_{t_i:l}$.;
7  end
8  Compute the average length $l_a$ (Equation 8.5).;
9  Verify the validity of $\mathcal{C}_3$: $l_a < l_m$ and $l_m - l_a \leq \tau$ ;
10 if $\mathcal{C}_3$ is fulfilled then
11     The 1D-CNN is retrained on the input time series subsequences of $\mathcal{D}_{train}$ of length
         $l_a$.
12 end
```

input time series to the same length. For the 1D-CNN, all convolutions have a stride equal to 1 with zero padding to preserve the exact length of the time series after the convolution. The first convolution contains 128 filters with a filter length equal to 10, followed by a second convolution of 128 filters with a filter length equal to 8, then a third convolution of 256 filters with a filter length equal to 5, which in its turn fed to a fourth and final convolutional layer composed of 128 filters, each one with a length equal to 3. The three dense layers are composed of 500, 300, and 100 neurons, respectively, each one using the ReLU activation function. The number of neurons in the final softmax classifier is equal to 8, i.e., the number of classes in the data. The model's weights are learned using a variant of Stochastic Gradient Descent (SGD), namely, Adam [351]. The number of training epochs is set to 2000.

8.3.3.2 Evaluation Metrics

To evaluate the achieved results, the confusion matrix 8.1 is utilized. This matrix represents all prediction results of a given model for multi-class as follows: The *Recall* metric calculates

Table 8.1: Confusion Matrix

		Predicted Value		
		Class A	Class B	Class C
Actual Value	Class A	Aa	Ab	Ac
	Class B	Ba	Bb	Bc
	Class C	Ca	Cb	Cc

the proportion of actual positives that are correctly identified. For example, for class A:

$$Recall(A) = \frac{TruePredictedA}{TotalTrueA} = \frac{Aa}{Aa + Ab + Ac} \tag{8.8}$$

The *Precision* describes what proportion of positive identifications are actually correct:

$$Precision(A) = \frac{TruePredictedA}{TotalPredictedAsA} = \frac{Aa}{Aa + Ba + Ca} \tag{8.9}$$

The *F1-score* represents the trade-off between Precision and Recall. They are calculated as follows for class A:

$$F1 - score = 2 \times \frac{Precision * Recall}{Precision + Recall} \tag{8.10}$$

These metrics are calculated per class. There are two ways to compute their average: *Macro Average* (*M.Avg*), which is a simple arithmetic mean of the metrics per class, and *Weighted Average* (*W.Avg*), which is a weighted mean based on the number of samples per class. For example:

$$\begin{aligned} M.AvgRecall &= \frac{\sum RecallPerClass}{TotalNumberOfClasses} \\ &= \frac{Recall(A) + Recall(B) + Recall(C)}{3} \end{aligned} \tag{8.11}$$

$$W.AvgRecall = \frac{\sum (RecallPerClass * NumberOfSamplesPerClass)}{TotalNumberOfSamples} \tag{8.12}$$

The *Micro Average* (*Mi.Avg*) is a metric that is used for the whole model. For example:

$$Mi.AvgRecall = \frac{\sum TruePredictedPerClass}{\sum TotalTruePerClass} \tag{8.13}$$

Following the same idea, the macro, weighted, and micro averages definitions can be extended to *Precision* and *F1-Score*. However, in the case of micro average, it should be noted that based on its definition, the following equation is valid:

$$\begin{aligned} Accuracy = Mi.AvgRecall = & \quad Mi.AvgPrecision = Mi.Avg\text{F1-Score} \\ = \frac{TotalTruePrediction}{TotalNumberOfSamples} & \end{aligned} \tag{8.14}$$

8.3.3.3 Results

Table 8.2 encloses descriptive statistics of the results on the prediction performance of the 1D-CNNs trained on the full-length time series data and on the identified subsequences by Algorithm 6, denoted CNN_{full} and CNN_{ITSSI}, respectively. The length of the input time series for the CNN_{full} is 727 time steps. However, the average length l_a derived following the procedure explained in Algorithm 6 is 250 time steps. So, CNN_{ITSSI} is trained on the input subsequences of length l_a.

Following Table 8.2, it can be seen that feeding the most important, i.e., discriminative time series patterns to the 1D-CNN improves its predictive performance. Since the data is highly imbalanced, the micro and the weighted average measures are biased towards the majority classes that are more accurately predicted by the classifier and much more represented in the data set. A better overview of the performance on minority classes can be seen using the macro measures that reflect the averaged measures independently from their representation in the data. A clear improvement in the Precision is achieved by the CNN_{ITSSI} while preserving a similar Recall. This results in an increase of 8% in the F1-score compared to the CNN_{full}. With respect to the CNN_{ITSSI}, the prediction accuracy is improved using shorter time series inputs (i.e., lower dimensionality). The dimension is reduced up to the third from 727 time points to

250. Therefore, our method can also be viewed as input feature selection (i.e., specific time series data points selection), which also helps in reducing the general resource consumption required for model training.

Relying only on the first 250 time steps to make a decision within the execution of the process enables to do corrective actions, thus avoiding costly rework and waste of resources through further processing of defective components, and also anticipating the type of defect helps to estimate the reworking time to correct it which varies from 5 seconds to 5 hours. From a practice point of view, if $\mathcal{C}_3$ is fulfilled with $l_m = 500$ and $l_a = 250$ which means some reactivity time τ is left, the process can be stopped to execute these actions so that expected quality deviations can be corrected.

Table 8.2: Average Performance Comparison of CNN_{ITSSI} vs. CNN_{full}

Metric	Precision		Recall		F1-score	
	CNN_{full}	CNN_{ITSSI}	CNN_{full}	CNN_{ITSSI}	CNN_{full}	CNN_{ITSSI}
M.Avg	58%	**66%**	77%	77%	63%	**68%**
W.Avg	**99%**	98%	94%	**97%**	96%	**97%**
Mi.Avg	–	–	–	–	94%	**97%**

8.3.4 Explainable Quality Prediction

Figures 8.2 and 8.3 show the localization of some examples of the identified subsequences for different classes and the differences between them.

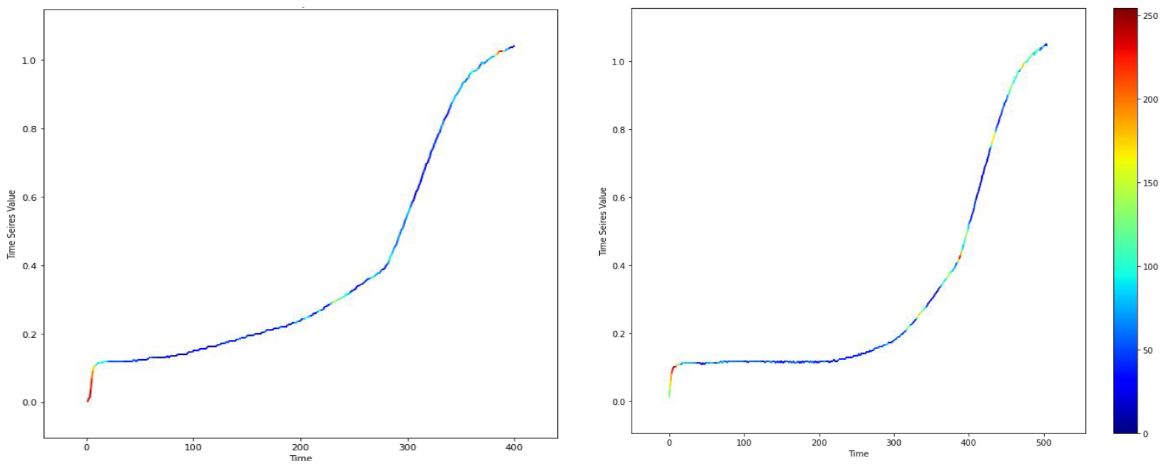


Figure 8.2: Examples of heat-maps produced by the Grad-CAM for two examples of input time series for the "non-defective" class. The x-axis is the time, while the y-axis is the recorded torque value as the time series value. The red color is used for highlighting the most important subsequences and the blue for less important parts.

In Figures 8.2 and 8.3, the most important subsequences are highlighted in red, while the least important are present in dark blue. It can be seen that the "non-defective" class presents some regularity in the patterns (Figures 8.2). The most important subsequences are almost localized at the beginning of the process. Clear different patterns can be distinguished in the highlighted subsequences for the "defective" classes compared to the "non-defective" class patterns. They are also different from each other, confirming thus that they represent two different types of defects. Their localization, on average also confirms the validity of the

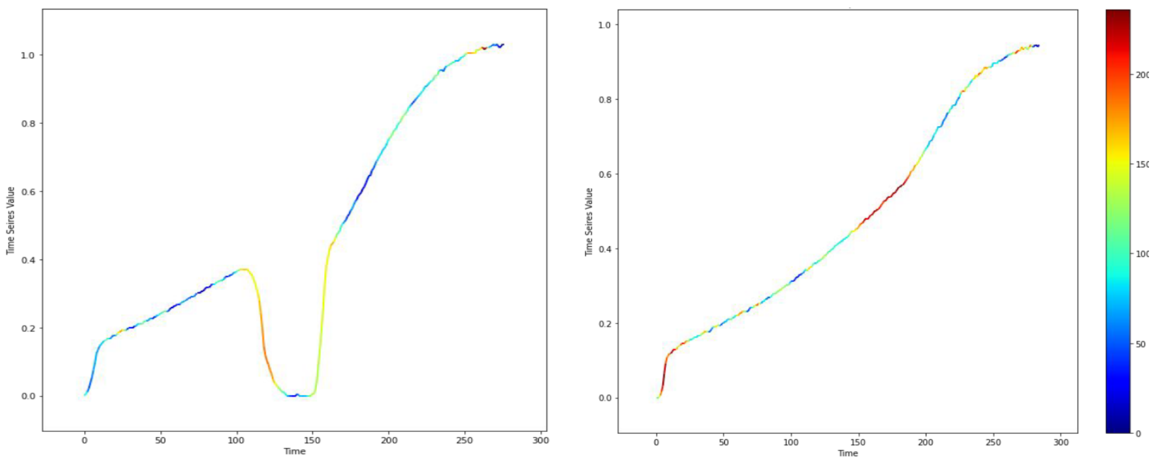


Figure 8.3: Examples of heat-maps produced by the Grad-CAM for two examples belonging to two different "defect" classes. The x-axis is the time, while the y-axis is the recorded torque value as the time series value. The red color is used for highlighting the most important subsequences and the blue for less important parts.

derived length $l_a = 250$ by the **ITSSI** Algorithm 6 since almost all of them are located before time $t = 250$.

These patterns can be shown to domain experts so that they can determine which part of the torque is affected the most. For example, in the left torque curve in Figure 8.3, it can be seen that the defect is located in the alignment zone, while in the right torque curve, it is located in the elastic clamping zone. Such localization visualization validates the robustness of the 1D-CNN as it builds its decision on clear anomalous subsequences when comparing Figure 8.3 to Figure 8.2. In addition, different torque zone identification helps in determining necessary corrective measures and corresponding resources in an adequate manner.

The most important subsequence localization can be further optimized by also tuning t_i instead of setting it to t_0 and by ensuring equal length sequences to be fed to the 1D-CNN using zero padding.

8.4 Explainable Quality Prediction in Electronics Manufacturing

Due to a combination of different effects such as increasing competitive pressure, globalization, and supply shortages resulting from external factors (i.e., lockdown in different countries along the supply chain, chip shortages, etc.), the production of zero-defect products is becoming an important competitive factor for modern and successful electronic manufacturing companies. In SMT manufacturing, expensive high-end inspection systems (such as X-ray machines) are usually used to inspect the quality of high-volume products at the End-Of-Line (EOL) to ensure the delivery of zero-defect products [224]. These optical inspection systems provide quality assessments that subsequently lead to conclusions about corrective actions to correct or enhance the quality of the product [224]. While these measurements are necessary to prevent the delivery of defective products, quality testing is often prone to become a bottleneck of the production line because of the highly time-consuming inspection process [224], [352], [353].

Recently, through the use of low-cost sensors and storage devices in SMT manufacturing, extensive amounts of data have been collected and stored by manufacturers [224], offering enormous potential not only for real-time process monitoring but also for gaining valuable insights and knowledge about processes [8]. Consequently, model-based quality prediction is employed to replace the offline time-consuming traditional testing procedures in SMT manufacturing. The prediction of the final product quality in conjunction with sample-based physical testing is then used to derive corrective measurements for quality control [224]. Thus, a paradigm shift from descriptive (monitoring and summarising process data) to predictive analytics (using ML models to predict possible future quality indicators) has been observed [8], [45], [225]. The application of model-based quality prediction combined with an interlinked production environment leads to real-time capable approaches to resolve quality deviations through corrective measurements early in the process and to improve the overall product quality. Quality prediction in the electronics manufacturing industry is a widely discussed topic in the literature (e.g., [354]–[356]), reflecting the wide diffusion of ML applications in electronics manufacturing.

In this context, we employ a 1D-CNN for quality prediction in SMT manufacturing. We compare the obtained results with conventional methods such as MLP [264], GBM [263] and RF [31]. For explainability, in order to investigate the process from different perspectives, we use the 1D-CNN given the flexibility that it offers with regard to data reshaping so that a distinction between *global* and *local* features and their corresponding importance can be easily made. The task is to map process features of previous processing steps to binary quality labels of the EOL-testing, differentiating between the *Ok*- and Not-Ok (*NOk*) pieces and can therefore be described as a binary classification. We use a heat map gradient-based visualization technique, again Grad-CAM, to make the output of the 1D-CNN explainable to the user. As a further step, an edge cloud computing architecture is described for the model deployment to give the reader a guideline on how explainability methods can be applied in Industry 4.0 for process optimization and recommendation systems building.

8.4.1 Use Case Description

The case study is conducted on an SMT production line where an X-ray inspection system is used for quality control at the End Of the production Line, i.e., EOL quality testing. Since the X-ray inspection induces high resource consumption, quality control is to be made faster by the application of ML model-based quality prediction. This use case has been part of other publications that serve as a reference for our experiments [224]. The SMT assembly process is a process chain consisting of the following consecutive steps: First, a raw Printed Circuit Board (PCB) is inserted into a printer via a conveyor belt, then the solder paste is printed onto the PCB. Following this, Solder Paste Inspection (SPI) is applied in a visual inspection station to assess the quality of the solder paste position. Afterward, individual components are automatically placed on the board and are then transported by conveyor belts to the re-flow soldering process, where the applied solder paste is melted in various heat zones to connect the components with the PCB. After the soldering process, the components of the PCBs are examined by an Automatic Optical Inspection (AOI). Depending on the product type, subsequent additional X-ray inspection is carried out. The AOI captures surface properties.

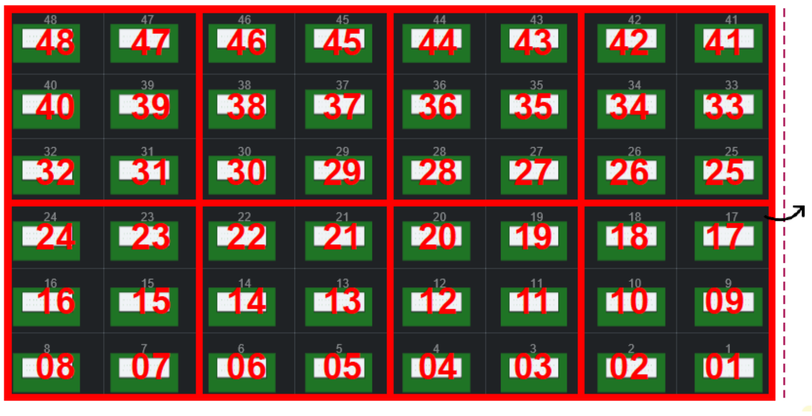


Figure 8.4: Field of Views (FOVs) of the selected panel variant in $X2$.

Opposingly, the X-ray inspection, which is used to detect pins located beneath the surface of the components, must be performed in a separate batch process. As a result, a long inspection time is expected.

The case study covers a specific product variant of a connector PCB and considers the corresponding manufacturing process in the SMT line as well as the information from the SPI and the X-Ray inspection. During the manufacturing process, the individual PCBs are grouped together by 48 units as one panel. Depending on the orientation of the panel, each PCB has a different number of pins, 79 for $X1$ orientation (Top) and 52 for $X2$ (Bottom). Since it would take too much time to assess the quality at the pin level, the quality information of the panels is aggregated at a Field-Of-View (FOV) level, which corresponds to the aggregation level of the X-Ray inspection. One FOV consists of 6 PCB boards and is denoted as *NOk* if one PCB is detected as defective, whereas it is declared as *Ok* when all PCBs are defect-free. Similarly, Each PCB is labeled as defective if at least one pin in at least one orientation is defective. Otherwise, the PCB is considered defect-free. Figure 8.4 shows an example panel of PCBs and corresponding FOVs in the orientations $X2$. The FOVs are the six big squares with red borders. Each data point corresponds to one pin described by a set of numerical features from the SPI (see Table 8.3).

Table 8.3: Descriptive PCB features on the pin level.

SPI feature	Height	Shape 2D	Shape 3D	Surface	Volume	Offset X	Offset Y
Unit	%	%	%	%	%	μm	μm

For the case study, the dataset covers a period of five production months containing a total of $1,461,037,321$ data points. Because decisions on dynamic X-ray inspection or alternative routings can be made only on higher aggregation levels and because the supervision of the PCBs on the pin level is not feasible, the solder pins are aggregated to a PCB level leading to a relatively higher dimensional quality prediction task, i.e., For the $X1$ orientation 7×79 and for the $X2$ orientation 7×52. Afterward, decisions for the quality of FOVs can be derived as explained above.

8.4.2 Modeling

8.4.2.1 Preliminaries

Let the input data be denoted as a multi-set $\mathcal{D} = (X, Y) = \{(x^{(i)}, y^{(i)})\}_{1 \leq i \leq n}$ which consists of $|\mathcal{D}| = n$ input data point $x^{(i)} \in \mathbb{R}^d$ and their corresponding label $y^{(i)}$. The data is transformed such that each data point corresponds to one PCB in a given orientation. Each data point is described by $x^{(i)}$, which consists of a d-dimensional vector of feature values and a corresponding quality label, denoting whether the PCB is *OK* or *NOK*. The goal is to train a predictive ML model to approximate some true function $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ denotes the input feature space and $\mathcal{Y}$ is the target variable to be predicted. In our case, the predictive ML model is built to predict a discrete binary quality label given an input feature vector. Therefore, $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$ is the target variable that can take two possible values mutually exclusive. If $y_i = 0$, the corresponding PCB is considered as *NOK*. In the opposite case, it is treated as *OK*. The learning task can be formalized as a binary classification task. The prediction of $\mathcal{Y}$ is carried out by the application of a model $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$. Usually, a fixed structure of f_θ is chosen that is parametrized by some vector $\theta \in \mathbb{R}^p$. Learning from data $X \in \mathcal{X}$ consists then of basically fitting θ to X, so that $f_\theta \approx f$. Random Forest (RF) [31], Gradient Boosting Trees (GBT) [263], Multi-Layer Perceptron (MLP) [268], and Convolutional Neural Networks (CNNs) [120] are special classes of predictive models.

8.4.2.2 1D-CNN Architecture

The convolutional block of the 1D-CNN consists of several convolution layers followed by a dropout layer, then a pooling layer, and a flatten layer. Dropout is a technique used to prevent the model from overfitting. Dropout works by randomly setting the outgoing edges of the hidden units (neurons that make up the hidden layers) to 0 at each update of the training phase. The pooling layer performs a pooling operation like average or maximum computation to each of the received feature maps from the convolutional layers. After the convolution blocks, the features are fed into a Max Average Pooling (MAP) layer. Then, flattening is applied to convert the data into a 1-dimensional (1-D) array for inputting it to the next layer. The second block is a Fully-Connected (FC) block that includes the concatenation layer for merged features of the convolutional block using two dense layers and an output layer with 1 neuron with a softmax activation function to output a class probability.

8.4.2.3 CNN Visual Explanations

We are mainly interested in predicting and understanding the resulting quality of the FOVs for each orientation of the panel, namely X_1 and X_2. Each orientation has a different number of pins, and each pin is described by the 7 various physical quantities in Table 8.3. As a result, the two orientations have a different number of features. Therefore, we train a specific ML model for each orientation. We refer to these physical quantities in Table 8.3 that describe all the pins as *global features* and to the set of pins describing each PCB in a given orientation with their corresponding physical quantities as *local features*. It is interesting and more practical to explain the models' decisions from the perspective of both *global* and *local* features so that engineers can understand which physical quantities need to be considered more closely and which pins are more prone to contain defects (i.e., which process parts are responsible for such pins). Feature importance analysis can be considered as a tool to provide these explanations [40]. Some ML models like RF [31] and GBT [263] are equipped with feature importance measures. Since the features describing the PCBs are the set of *local features*, we can get feature importance analysis exclusively on the pin level. Also, due to the high number of features, it is very difficult to assess their importance with quick analysis using the standard feature importance analysis and visualization methods, e.g., box-plots [40]. Opposingly, the 1D-CNN can provide different overviews of the input data by allowing convolutions either along the physical quantities or along the pins for each orientation. This would result in two different 1D-CNNs for each orientation. Afterward, a visualization heat-mapping-based explanation method is used for each model to highlight the most important features for the prediction of each class. In this way, feature importance analysis can be conducted for both *global* and *local* features. In addition, heat maps are much easier to investigate visually and interpret. More specifically, Grad-CAM [203] is employed to produce these heat maps. More details about Grad-CAM are provided in the previous Section 8.3.2.3.

8.4.2.4 Model Deployment

The technical details of the deployment for this particular use case are already described in [224]. We, therefore, focus on the conceptual details and briefly summarize the architecture.

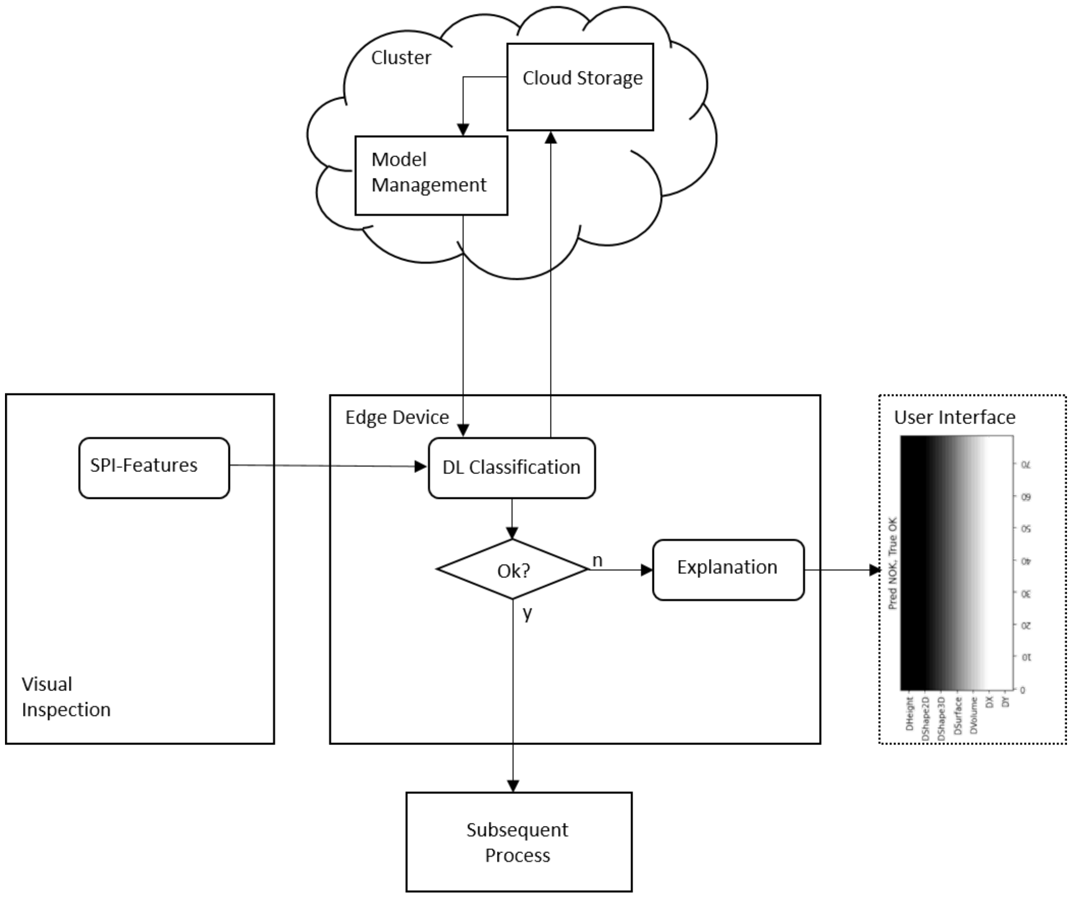


Figure 8.5: Explainable predictive quality inspection in SMT manufacturing

Then, we discuss the extension part that is necessary for the use of the explainable predictive quality inspection. The overall architecture is illustrated in Figure 8.5. ML model management, i.e., training, updating, and storage, takes place on a computing cluster, which in this case is a Spark cluster, while the storage of the data is handled in a datalake or data warehouse. The SPI features are transmitted directly from the visual inspection machines to the edge device, which in this case, is an industrial PC. The three components, i.e., the datalake or data warehouse, the computing cluster, and the edge device, are connected via a network layer. To reduce latency and bandwidth-related issues, the model is stored on the edge device. The ML model classifies the PCBs of the FOV level to enable rerouting of the product. If the classifier detects that a PCB is defective (*NOk*), the graphical user interface visualizes two perspectives, i.e., the *local* as well as the *global* features to support explainability by depicting the location of the defective pin areas and the deviating SPI features.

8.4.3 Quality Prediction Results

8.4.3.1 Experimental Setup

The available dataset $\mathcal{D}$ is split into 75% for training and 25% for testing. A 10-fold cross-validation procedure is employed to evaluate the models. The reported results are the average metric values over the 10 folds. The dataset reveals a high class-imbalance ratio. Therefore, random over- and under-sampling strategies are combined and applied to the training data [350]. For the GBT and RF, we use a grid-search procedure to tune their hyper-parameters. For the MLP, we use five hidden layers. The MLP's weights are learned using a variant of the Stochastic Gradient Descent (SGD), namely, Adam [351]. The number of training epochs is set to 1000. For the 1D-CNN, depending on the orientation and whether we perform the convolution along the pins or the SPI (*global*) feature, the first convolution contains filters with a number equal to the pins with a filter length equal to 7 (i.e., number of the SPI features) or the way around. The size and lengths of the filters of the following layers depend on the orientation ($X1$ or $X2$) and the convolution direction (i.e., pins or SPI features). The number of neurons in the final softmax classifier is equal to 1, outputting a class membership probability. Similarly to MLP, The model's weights are learned using Adam [351] with 2000 training epochs.

8.4.3.2 Evaluation Metrics

The commonly used metrics for evaluating a binary classification task are derived using the confusion matrix with the true predicted values TP = True Positive, TN = True Negative and the false predicted values False Positive and False Negative. In this notation, the positive class corresponds to the non-defective class and the negative class to the defective class.

The most commonly used metric for classification tasks is accuracy:

$$Accuracy = \frac{\text{True Predicted Classes}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + FP + TN + FN} \tag{8.15}$$

However, the accuracy is insensitive to class imbalance, and therefore misleading [357]. Therefore, the recall measures the partition of correct prediction to the total amount of true values per class is also reported:

$$Recall = \frac{\text{True Predicted Values}}{\text{Total True Values}} \tag{8.16}$$

$$Recall(NOk) = \frac{TN}{TN + FP} \tag{8.17}$$

$$Recall(Ok) = \frac{TP}{TP + FN} \tag{8.18}$$

The recall of the *NOk* class is of the highest importance since the manufacturer can not tolerate *NOk* being classified as *Ok*.

8.4.3.3 Results

The results are presented in Table 8.4 and are averaged over X_1 and X_2 orientations. The Table encloses descriptive statistics trained per PCB so that in the X_1 orientation one data point consists of $79 * 7 = 553$ features and in the X_2 orientation of $52 * 7 = 364$ features. Afterward, the results of the PCBs prediction are aggregated to the FOV level. Referring to Table 8.4, it can be seen that the Recall on the *NOK* class has the highest value for the 1D-CNN trained using convolutions over the pins. It also maintains a relatively good recall on the *OK* class. The reshaping of the data to highlight the global physical SPI features has slightly worse performance since aggregation of these features over all the pins is performed. However, a good trade-off of the recall on both classes is maintained. Traditional baselines like RF and GBT have comparable performance. The difference in performance between our GBT and the GBT in [224] is explained by the application of a class re-balancing strategy on the training data in our case before training the ML models, which seems to be necessary since a high-imbalance ratio introduces a high bias towards the majority class. The MLP has a higher recall compared to the remaining methods on the *OK* class but a lower recall (except for GBT [224]) which has a very poor performance on the class of interest *NOK*).

Table 8.4: Average Performance Comparison of different classifiers

Metric	Recall(*NOk*)	Recall(*Ok*)	Accuracy
RF	69.83%	53.76%	73.46%
GBT	73.98%	39.70%	43.26%
MLP	16.08%	**95.13%**	80.15%
$1D - CNN^{global}$	66.92%	42.78%	65.12%
$1D - CNN^{local}$	**75.4%**	43.85%	**74.71%**
GBT [224]	7.6%	29.4%	29.63%

Based on the performance of both 1D-CNNs on the *NOK* class, we make the argument that they can be used to give insight to the user into which physical quantities (*global features*) and which specific pins (*local features*) influence the most the quality prediction. To do so, we collect samples that are correctly classified by both 1D-CNNs per class and we compute the Grad-CAM values and average them over all these samples for each class. Computing the maps in an averaged manner over many samples leads to conclusions about process optimization or product-specific measurements that can be taken by domain experts.

8.4.4 Explainable Quality Prediction

Heat maps for both classes for both CNNs are shown in Figures 8.6 and 8.7 . From these Figures, a user can locate and infer the most discriminative features for each class and can get a better understanding of the importance of *global* physical SPI features (the sub-figures at the bottom of each Figure) and localize the main influencing pins, i.e., the *local* features (the sub-figures at the top of each Figure). It can be seen that different *global* and *local* features are highlighted for each class. Most importantly, for correctly predicting the *NOK* class, the focus of the 1D-CNN is mainly on the SPI features DX and DY, a little less on DVolume and DSurface. The remaining features seem to be irrelevant for making the decision toward assigning, on average, the *NOK* pieces correctly to the *NOK* class. This may give some hints to domain experts to investigate further DX and DY signals and, more importantly, to discover the causes of deviations between these signals for the *OK* and the *NOK* classes. Looking also at the pin level, it can be seen that the model's focus is on some particular pins, namely from 1 to 5 and from 15 to 25. This shows that some pins are more prone to contain defects than others. This may also give domain experts some hints on investigating which process parts or machines are most often responsible for these particular pins and need to be tracked back.

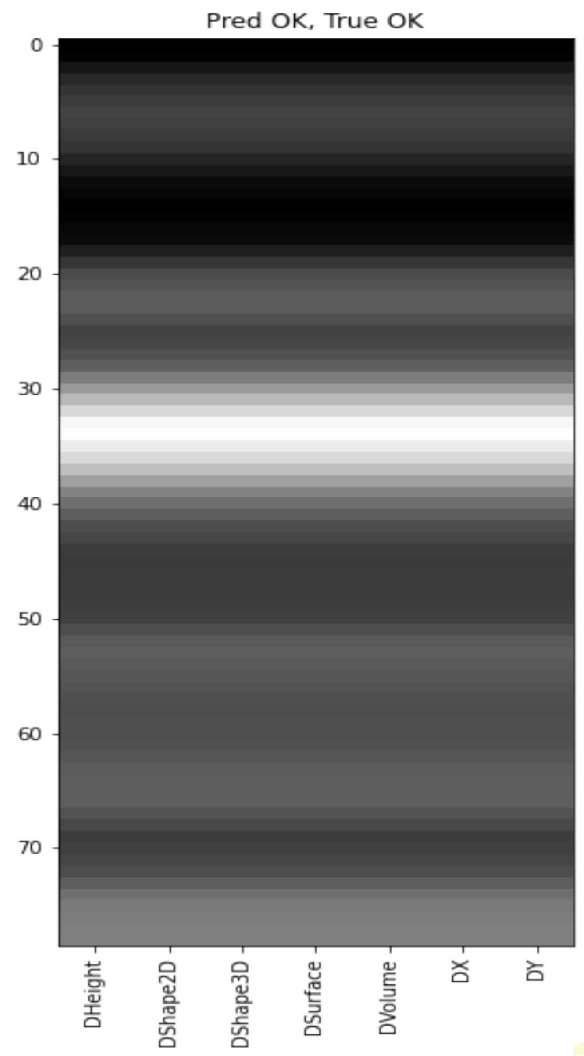
8.5 Concluding Remarks

In this chapter, we present two applications of explainable quality prediction in Industry 4.0.

First, a novel approach for important time series subsequences identification for time series classification is presented. This is achieved using a 1D-CNN model and a gradient-based heat-mapping approach, namely Grad-CAM. The method demonstrates a satisfactory performance not only in improving the classification accuracy but also in achieving early quality prediction in a real-world use case in the automotive industry. Quality prediction explainability is also promoted using the produced heat maps.

Second, an explainable model-based quality prediction in SMT manufacturing is presented. This is also achieved using a 1D-CNN model and the Grad-CAM method. First, the predictive model demonstrates satisfactory performance in the detection of quality deviations. Second, the heat-mapping-based method is used to reveal the causes of quality deviations from two different perspectives yielding corrective measures for these deviations. Lastly, the deployment of the ML solution within an edge cloud computing-based architecture is described.

In both cases, further interaction with domain experts is required in order to qualify and quantify the impact of our solution on a larger scale. In addition, the quantification should go beyond standard ML measures such as recall or accuracy to include more application-specific measures or indicators, e.g., energy consumption reduction, material savings, customer satisfaction level, etc.



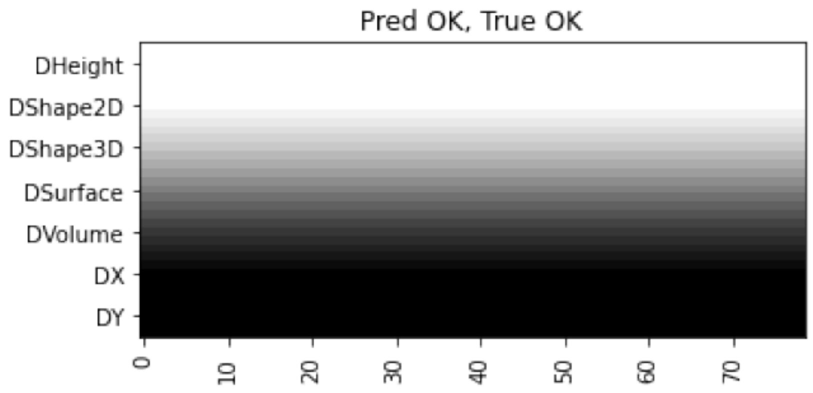
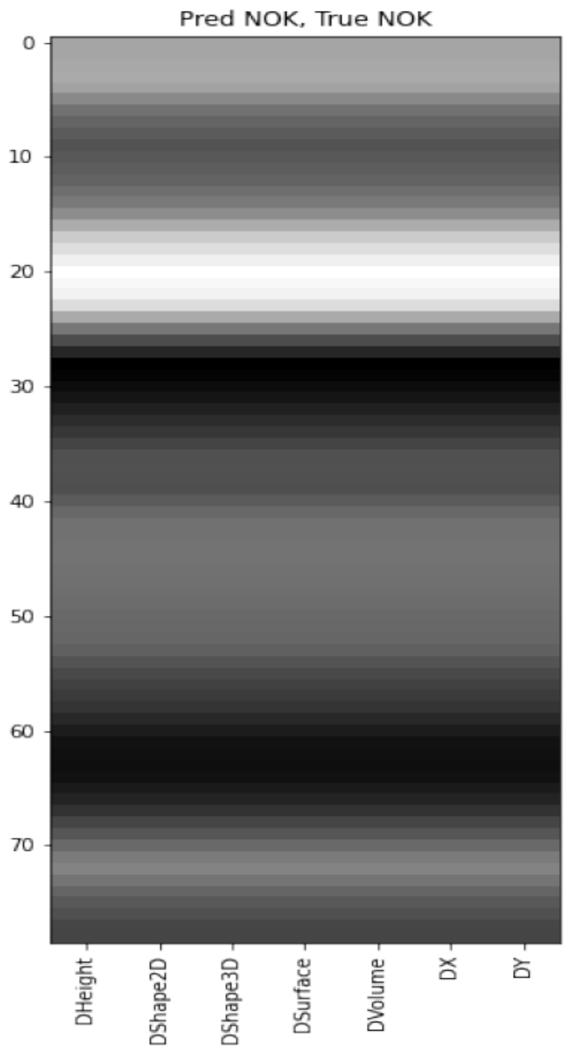


Figure 8.6: Heat-maps averaged over many samples for $X1$-direction.x-axis shows the SPI-features, y-axis the numbering of the pins, and the white coloring highlights the most discriminative regions of the PCB (most important input features), i.e., highlighting *local* features (pins)



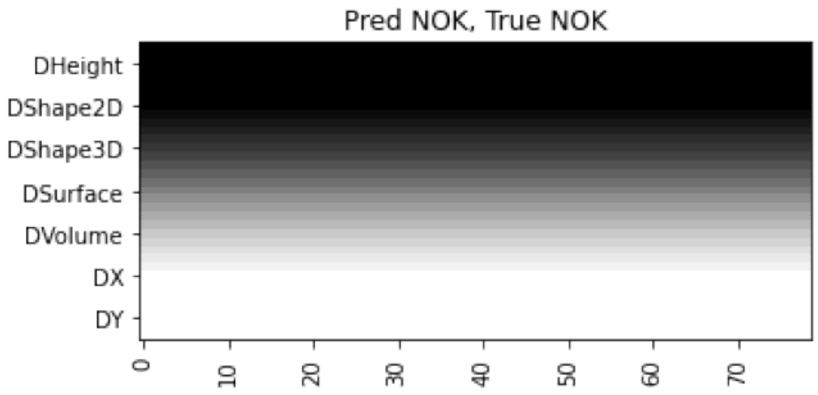


Figure 8.7: Heat-maps averaged over many samples for $X1$-direction. y-axis shows the SPI features, x-axis the numbering of the pins, and the white coloring highlights the most discriminative regions of the PCB (most important input features), i.e., highlighting the *global* features (physical SPI).

Part IV

Conclusions

9 Conclusions

9.1 Main Conclusions

Time series are omnipresent in a wide variety of fields and thus can be exploited to describe many real-world phenomena in several domains of application, including industrial processes, transportation, finance, and healthcare, to name but a few. The way we forecast how the time series will evolve in the future extremely influences decision-making in the present. However, due to the uncertainty related to the future of a time series, organizations are nowadays focusing on data-driven approaches to build forecasting systems for decision-making.

In this context, the goal of this thesis is focused on devising methods for forecasting the future behavior of time series online. That is, our objective is to leverage historical time series data that is collected over time and support organizations in making accurate data-driven decisions by forecasting the future in an accurate, timely manner. In particular, we framed our objectives in two folds:

1. to develop new methods for automatically forecasting time series in an online adaptive manner such that they are able to efficiently cope with potentially different types of non-stationarities that most often characterize this data type and must be involved in its modeling process. These methods are further promoted for the trustworthy application of Machine Learning by being supported with explainability tools.
2. to build new methods for model-based quality prediction for Industry 4.0 applications in a timely manner.

9.1.1 Forecasting

The thesis was split into two main parts according to the objectives stated above. In the first main part dedicated to (Forecasting), we handled the task of time series forecasting, denoting the prediction of the next value of a numeric time series using historical data. In the introductory chapter of this thesis, we divided the first objective into four research questions. We now recapitulate them and answer them in turn:

- **RQ1** Given the time-evolving spatio-temporal dependencies among time series variables in MTS, what is the most appropriate way of selecting the most relevant variables in a timely manner to enhance the predictive performance of forecasting MTS models?

  In Chapter 3, we address the problem of time series variables selection for MTS forecasting by developing OAMTS: an online adaptive framework that performs both input time series variables and adequate forecasting model selection. Input variables selection is made in two stages using relevance and redundancy analysis. The selection is made dynamically and adaptively in an informed manner following concept drift detection. The concept drift detection covers the two MTS dimensions, namely spatial and temporal. Spatial dependencies indicate the similarity between the input variables at one time instant. We monitor the change in the similarity values over time. Temporal dependencies indicate the patterns discovered within the same spatial dimension over time. The drift detection within the temporal dimension is ensured by tracking the change in the estimated model's performance on a given target time series variable. In addition, the choice of adequate relevance and redundancy measures, as well as the forecasting model, is done in an automated fashion using meta-learning on well-devised MTS meta-features. OAMTS is validated using a comprehensive empirical analysis with 66 real-world MTS datasets from different domains. We have created separate meta-data which cover a collection of real-world and synthetic MTS with various characteristics for the meta-learning task. The obtained results show that OAMTS achieves excellent results in comparison to the SoA approaches for MTS forecasting.

- **RQ2** How can we dynamically select one or multiple forecasting models and cope with non-stationary sources of variation that are frequently at play in time series and the time-varying models' performance?; **RQ3** How can we provide suitable timely explanations for the reason behind model selection at a given time instant or interval?

  Several machine learning models have been used to solve the task of time series forecasting. However, it is generally accepted that none of these models is universally valid for every application and over time. Therefore, adequate and adaptive real-time model selection is often required to cope with the time-evolving nature of time series and the fact that models have specific Regions of Competence (RoCs) in the time series data. In Chapter 4, we have developed two methods that are suitable for both online single forecasting model selection and an ensemble of heterogeneous forecasting models pruning.

  The first method, called DEMSRC consists of a drift-aware meta-learning approach for adaptively selecting and combining forecasting models. It is based on the assumption that different forecasting models have different areas of expertise and varying relative performance over time. Therefore, DEMSRC ensures the dynamic selection of initial ensemble members through a performance drift detection mechanism. Since diversity is a fundamental component in ensemble methods, a second stage selection of models' clustering is added to DEMSRC and updated with each drift detection in the performance. Predictions of the final selected models are combined into a single prediction. Exhaustive empirical testing of DEMSRC was performed, evaluating both its generalization error and scalability using time series from several real-world domains. Empirical results show the competitiveness of the method in comparison to State-of-the-Art approaches for combining forecasters.

Based on the fact that different forecasting models have different specific RoCs in the time series data, the second method OMS-ROC performs online single model selection for time series forecasting by using an adaptive clustering approach to compute the RoCs of candidate models. This method can be extended to ensemble pruning by combining clustering with a rank-based approach. In OMS-ROC, the selection of the appropriate model(s) is made online, and the update of the RoCs responsible for the model selection is done adaptively after the detection of concept drift in the structure of the RoCs. Moreover, the computed RoCs can be used to provide appropriate explanations for the reason for selecting certain model(s) in a certain time interval or instant and for observing certain performances/outcomes. Since the RoCs are computed independently of the family of forecasting models in question, the explanations we provide are model-agnostic. An extensive empirical study on various real-world datasets shows that our method achieves excellent or on-par results compared to state-of-the-art approaches and various baseline solutions.

Finally, in Chapter 5, we have specifically investigated the online selection of DNNs for time series forecasting. The reason behind this specific attention is that DNNs are nowadays widely used in the context of time series forecasting. In addition, DNNs are usually known to be extensive resource-consuming models making thus their online selection and management very challenging, especially in resource-constrained environments. These models are also known to be very complex and usually referred to as black-box models, making thus their explainability very challenging but also highly required. To do so, we first developed a method for online single DNN selection called OS-PGSM which consists of a novel approach for online deep CNN selection using saliency maps in the task of time series forecasting. We start with an arbitrarily set of different CNN forecasters with various architectures. Then, we outline a gradient-based technique for generating saliency maps with a coherent design to make them able to specialize the CNN forecasters across different regions in the input time series using a performance-based ranking. In this framework, the selection of the adequate model is performed in an online fashion, and the computation of saliency maps responsible for the model selection is achieved adaptively following drift detection in the time series. In addition, the saliency maps can be exploited to provide suitable explanations for the reason behind selecting a specific model at a certain time interval or instant.

The extension of OS-PGSM to an ensemble of DNNs pruning is presented by OEP-ROC.The same principle of performance-based saliency maps is applied to prune the ensemble by taking into account both aspects of *accuracy* and *diversity*. In addition, the saliency maps can be exploited to provide suitable explanations for the reason behind selecting specific models to construct an ensemble that plays the role of a forecaster at a certain time interval or instant.

An extensive empirical study on various real-world datasets demonstrates that both methods achieve excellent or on-par results in comparison to the State-of-the-Art approaches as well as several baselines.

- **RQ4** In the case of many models selection, how can we dynamically and adaptively combine them to cope with the mentioned variations?

  Several approaches, ranging from simple and enhanced averaging tactics to applying meta-learning methods, have been proposed in the ML literature to learn how to combine individual models in an ensemble. However, finding the optimal strategy for ensemble aggregation remains an open research question, particularly when the ensemble needs to be adapted in real-time. In Chapter 6, we leveraged a Deep Reinforcement Learning framework for learning linearly weighted ensembles as a meta-learning method. In this framework, the combination policy in an ensemble is modeled as a sequential decision-making process, and an actor-critic model aims at learning the optimal weights in continuous action space. The policy is updated following a drift detection mechanism for tracking performance shifts of the ensemble model. An extensive empirical study on many real-world datasets demonstrated that our method achieves excellent or on-par results in comparison to the State-of-the-Art approaches as well as several baselines.

9.1.2 Model-based Quality Prediction

- **RQ5** How can the developed methods be transferred efficiently to industrial case studies to perform quality prediction in real-time? And how can the online management of many models be exploited to integrate sensor and simulation data for Machine Learning applications efficiently?

  In Chapters 7 and 8, we investigated three different industrial applications of model-based quality prediction. First, we applied our forecasting methods to predict the cutting forces of an NC-milling process in real-time. The prediction can subsequently be used to monitor the quality of the process online. But before, we enriched the time series sensor data of the cutting forces with simulation data. To do so, we devised a fully automated framework for simulation-sensor data fusion that efficiently integrates the pre-calculated simulation data into sensor time series data and automatically selects the suitable fusion level, namely data or model level. Our forecasting methods applied to the fused data showed a better performance compared to the State-of-the-Art approaches as well as several baselines. We also investigated the usefulness of exploiting DNNs' heat maps in explaining model-based quality predictions for two case studies belonging to the electronics and automotive industries, respectively. The produced explanations were proven to be helpful in moving one step ahead in linking predictive analytics to perspective analytics in Industry 4.0 applications.

9.2 Open Issues and Future Directions

The methods presented in the thesis can be improved in a number of ways. In this section, we outline some potentially interesting research directions.

9.2.1 Towards a Fully Automated Forecasting Framework

ML plays an increasingly influential role in science and industry. However, ML-based solutions application in the real world still requires huge efforts of technical expertise. Therefore, many research efforts in the ML field are oriented toward automating ML to facilitate its use by non-ML experts. We proposed a method OAMTS that selects the time series variables and the forecasting models for MTS data automatically using meta-learning. The update of the selection is also decided automatically in an informed manner following concept drift detection. The quality of automation depends on the meta-learning model and the drift detection mechanism accuracy. For this reason, further work is required to increase the amount of the meta-data used in the meta-learning task and deploy further experimental efforts in tuning the hyper-parameters of the drift-detection algorithms to reduce the rate of false alarms triggered by the detection of false drifts.

Regarding DEMSRC and OMS-ROC which perform online model selection and/or ensemble pruning and automatically adapt themselves to changes in the time series and candidate models performance in an informed manner, there are still some important decisions left for the user, for example, the number of final models to be kept in the case of ensemble pruning, the size of the time series validation windows, or the hyper-parameters of the drift detection mechanisms. In OS-PGSM and OEP-ROC the task of base CNNs generations can be automated by considering a wide variety of architectures obtained from the automatic variation of, for example, the number of convolution blocks, the type of the pooling layer or the architecture of the fully-connected layer. Finally, the online aggregation framework OEA-DRL decides automatically for the ensemble weights and updates the aggregation policy as well as the weights in an informed fashion. The automation of this framework can be further enhanced by incorporating the pruning into the aggregation using constraints on the weights to set some of them to zero.

Last but not least, the choice between all the developed methods can be automated by means of meta-learning, for example, which offers a higher level of automation when it comes to the decision of the best online adaptive frameworks for time series forecasting.

We believe that automating all these tasks and developing a completely auto ML framework for forecasting is important for increased adoption of ML solutions, especially by non-ML experts [358]. We started the first step by developing an R-Shiny App that is devised to make our methods easier to use and evaluate on new time series datasets. It is also straightforward to use by non-ML domain experts, especially since online time series forecasting is required in a wide range of application domains. The App can also be used by ML researchers to reproduce our experiments easily. Our first plan is to finish the design of the App and publicly host it. In addition, we plan to add an extension to the same App for time series explainability tools, including the presented approaches in this thesis. In Appendix B, we aim to give an overview of the first version of the App and our research vision for automating time series forecasting in an understandable manner.

9.2.2 Beyond One-step ahead Forecasting

In the Forecasting part of the thesis, we formalized the forecasting task as predicting the next value in a time series, i.e., one step ahead, in a step-wise manner. In other words, the task in almost all the cases except for the milling use case is reduced to predicting x_{t+1} using $\{x_1, \cdots, x_t\}$ which can be viewed as explanatory variables in the predictive models. However, it should be noted that in the decision-making process, it is essential to quantify the uncertainty behind predictions. Therefore, using probabilistic models in this context can be helpful. More precisely, instead of forecasting the next time series point value $\hat{x}_{t+1}$ to approximate x_{t+1}, it may be more beneficial to give an estimate of the predictive probability distribution over the future values [359], or a prediction time interval [360]. In addition, one may be interested in predicting more than one step ahead in the future, for example, due to delayed feedback or to the minimum required time of reactivity for decision-making which is application dependent. Throughout this thesis, we based our work on the assumption of immediate availability of feedback from the environment of applications, i.e., the $t-th$ observation is assumed to be known when predicting the next observation at $t+1$. Indeed, forecasting multiple steps ahead is generally a more difficult task due to the increased related uncertainty [361]. In future research, we plan to extend our methods to these scenarios. The reformulation of the forecasting task as a general regression task with lagged time series values as explanatory variables allows more flexibility to model $x_{t+h}, h > 1$ using $\{x_1, \cdots, x_t\}$ We note that we address a simpler predictive task because we focus on the dynamic online selection of a set of forecasting models and their aggregation into one model, which is one of the main goals of one part of this thesis.

9.2.3 Beyond Univariate Time Series Forecasting

In Chapter 3, we showed the importance of input time series variables selection for MTS data before starting the forecasting task. Indeed, after careful selection of the appropriate variables using the selection procedure detailed in OAMTS, our online forecasting methods, including the ones that use a single model or ensemble of models, can be extended to MTS data. For DEMSRC and OMS-ROC, the extension is straightforward since these methods need just as input the predictions of the forecasting models. Therefore, we train many models on the selected variables of the MTS data in question. The selection is decided by OAMTS. Then, we apply DEMSRC and OMS-ROC.

Regarding the methods that are DNNs-specific, namely OS-PGSM and OEP-ROC, further efforts are required to transform the Performance-based Gradient Saliency Maps PGSMs into multi-dimensional time series data. Therefore, important time series sequences that are responsible for certain behavior of the MTS forecasting should be highlighted over multiple spatial dimensions. That is, instead of outputting a vector of importance values by the Grad-CAM, a matrix of importance values over time and space should be output. The resulting RoCs will be thus in the format of MTS sequences. Such RoCs can potentially be used to explain the dynamics of the relationships between the variables composing the MTS.

9.2.4 Towards More Time Series Specific Explainability Tools

We investigated the explainability of the online model selection process and, in some cases, the output of the ML models using visualization approaches that rely on gradient-based saliency maps or clustering visualization. However, a wide variety of explainability tools are presented in the ML literature [40]. Therefore, future work focused on the adaptation of these tools to the time series domain will be conducted in order to investigate which tools are more suitable than others for this data type. This also requires, in its turn, further elaborated work in the direction of evaluating different explainability methods.

9.2.5 On Empowering Process Simulation for Machine Learning Application

In the application part of the thesis, we investigated the task of model-based quality prediction. In one case, we showed the advantages of integrating simulation data into the learning process. However, simulations are most often simplistic representations of reality. Therefore, simulation model calibration to the particular application is crucial to reduce the simulation-reality gap. The computational cost of simulation calibration is high owing to the complexity of reality. In this context, ML can be used to improve the simulation. First, it can be used to facilitate the calibration process by turning it into a learning problem, i.e., learning simulation parameters that reduce the gap between simulation results and true observed values. In future work, we aim to apply Reinforcement Learning as a calibrator in the simulation environment. A second aspect we consider studying in the future is the replacement of weak simulation parts, i.e., parts with a high gap, with ML generative models.

References

[1] R. Godahewa, C. Bergmeir, G. I. Webb, R. J. Hyndman, and P. Montero-Manso, "Monash time series forecasting archive," in *Neural Information Processing Systems Track on Datasets and Benchmarks*, forthcoming, 2021.

[2] P. Esling and C. Agon, "Time-series data mining," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–34, 2012.

[3] C. A. Ralanamahatana, J. Lin, D. Gunopulos, E. Keogh, M. Vlachos, and G. Das, "Mining time series data," in *Data mining and knowledge discovery handbook*, Springer, 2005, pp. 1069–1103.

[4] J. D. Cryer, *Time series analysis.* Springer, 1986, vol. 286.

[5] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: A survey and empirical demonstration," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 102–111.

[6] N. R. Council *et al.*, *When weather matters: Science and services to meet critical societal needs.* National Academies Press, 2010.

[7] T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka, "Stock market prediction system with modular neural networks," in *1990 IJCNN international joint conference on neural networks*, IEEE, 1990, pp. 1–6.

[8] M. Stolpe, H. Blom, and K. Morik, "Sustainable industrial processes by embedded real-time quality prediction," in *Computational sustainability*, Springer, 2016, pp. 201–243.

[9] V. Cerqueira, L. Torgo, F. Pinto, and C. Soares, "Arbitrated ensemble for time series forecasting," in *Joint European conference on machine learning and knowledge discovery in databases*, Springer, 2017, pp. 478–494.

[10] V. Cerqueira, L. Torgo, F. Pinto, and C. Soares, "Arbitrage of forecasting experts," *Machine Learning*, 2018, ISSN: 1573-0565.

[11] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, 2017.

[12] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Predicting taxi-passenger demand using streaming data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1393–1402, 2013, ISSN: 15249050.

[13] T. Liebig, N. Piatkowski, C. Bockermann, and K. Morik, "Route planning with real-time traffic predictions.," in *LWA*, Citeseer, 2014, pp. 83–94.

[14] A. G. Salman, B. Kanigoro, and Y. Heryadi, “Weather forecasting using deep learning techniques,” in *2015 international conference on advanced computer science and information systems (ICACSIS)*, Ieee, 2015, pp. 281–285.

[15] E. Hoseinzade and S. Haratizadeh, “Cnnpred: Cnn-based stock market prediction using several data sources,” *arXiv preprint arXiv:1810.08923*, 2018.

[16] A. Saadallah, L. Moreira-Matias, R. Sousa, J. Khiari, E. Jenelius, and J. Gama, “Bright-drift-aware demand predictions for taxi networks,” *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[17] R. S. Tsay, *Multivariate time series analysis: with R and financial applications.* John Wiley & Sons, 2013.

[18] M. Stolpe, “The Internet of Things: Opportunities and challenges for distributed data analysis,” *SIGKDD Explorations*, vol. 18, no. 1, pp. 15–34, Jun. 2016.

[19] A. Saadallah, F. Priebe, and K. Morik, “A drift-based dynamic ensemble members selection using clustering for time series forecasting,” in *Joint European conference on machine learning and knowledge discovery in databases*, Springer, 2019.

[20] P. Geurts, “Pattern extraction for time series classification,” in *European conference on principles of data mining and knowledge discovery*, Springer, 2001, pp. 115–127.

[21] D. H. Wolpert, “The lack of a priori distinctions between learning algorithms,” *Neural computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

[22] F. Priebe, “Dynamic model selection for automated machine learning in time series,” 2019.

[23] Y.-A. Le Borgne, S. Santini, and G. Bontempi, “Adaptive model selection for time series prediction in wireless sensor networks,” *Signal Processing*, vol. 87, no. 12, pp. 3010–3020, 2007.

[24] D. V. Oliveira, G. D. Cavalcanti, and R. Sabourin, “Online pruning of base classifiers for dynamic ensemble selection,” *Pattern Recognition*, vol. 72, pp. 44–58, 2017.

[25] R. M. Cruz, D. V. Oliveira, G. D. Cavalcanti, and R. Sabourin, “Fire-des++: Enhanced online pruning of base classifiers for dynamic ensemble selection,” *Pattern Recognition*, vol. 85, pp. 149–160, 2019.

[26] R. Argiento, A. Guglielmi, and A. Pievatolo, “Bayesian density estimation and model selection using nonparametric hierarchical mixtures,” *Computational Statistics & Data Analysis*, vol. 54, no. 4, pp. 816–832, 2010.

[27] I. Rivals and L. Personnaz, “On cross validation for model selection,” *Neural computation*, vol. 11, no. 4, pp. 863–870, 1999.

[28] D. D. Margineantu and T. G. Dietterich, “Pruning adaptive boosting,” in *ICML*, Citeseer, vol. 97, 1997, pp. 211–218.

[29] N. Li, Y. Yu, and Z.-H. Zhou, “Diversity regularized ensemble pruning,” in *Machine Learning and Knowledge Discovery in Databases*, P. A. Flach, T. De Bie, and N. Cristianini, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 330–345, ISBN: 978-3-642-33460-3.

[30] G. Tsoumakas, I. Partalas, and I. Vlahavas, “An ensemble pruning primer,” in *Applications of supervised and unsupervised ensemble methods*, Springer, 2009, pp. 1–13.

[31] L. Breiman, “Bagging predictors,” *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[32] F. Zhang, J. Bai, X. Li, C. Pei, and V. Havyarimana, “An ensemble cascading extremely randomized trees framework for short-term traffic flow prediction,” *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 13, no. 4, pp. 1975–1988, 2019.

[33] D. H. Wolpert, “Stacked generalization,” *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[34] J. Khiari, L. Moreira-Matias, A. Shaker, B. Ženko, and S. Džeroski, “Metabags: Bagged meta-decision trees for regression,” in *Joint european conference on machine learning and knowledge discovery in databases*, Springer, 2018, pp. 637–652.

[35] A. Saadallah and K. Morik, “Online ensemble aggregation using deep reinforcement learning for time series forecasting,” in *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2021.

[36] A. Saadallah, M. Tavakol, and K. Morik, “An actor-critic ensemble aggregation model for time-series forecasting,” in *IEEE International Conference on Data Engineering (ICDE)*, 2021.

[37] S. Thiebes, S. Lins, and A. Sunyaev, “Trustworthy artificial intelligence,” *Electronic Markets*, vol. 31, no. 2, pp. 447–464, 2021.

[38] H. AI, *High-level expert group on artificial intelligence*, 2019.

[39] N. Bussmann, P. Giudici, D. Marinelli, and J. Papenbrock, “Explainable ai in fintech risk management,” *Frontiers in Artificial Intelligence*, vol. 3, p. 26, 2020.

[40] C. Molnar, *Interpretable machine learning.* Lulu. com, 2020.

[41] J. Petch, S. Di, and W. Nelson, “Opening the black box: The promise and limitations of explainable machine learning in cardiology,” *Canadian Journal of Cardiology*, 2021.

[42] A. Sanders, C. Elangeswaran, and J. P. Wulfsberg, “Industry 4.0 implies lean manufacturing: Research activities in industry 4.0 function as enablers for lean manufacturing,” *Journal of Industrial Engineering and Management (JIEM)*, vol. 9, no. 3, pp. 811–833, 2016.

[43] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, “Intelligent manufacturing in the context of industry 4.0: A review,” *Engineering*, vol. 3, no. 5, pp. 616–630, 2017.

[44] C. A. Escobar, M. E. McGovern, and R. Morales-Menendez, “Quality 4.0: A review of big data challenges in manufacturing,” *Journal of Intelligent Manufacturing*, vol. 32, no. 8, pp. 2319–2334, 2021.

[45] D. Lieber, M. Stolpe, B. Konrad, J. Deuse, and K. Morik, “Quality prediction in interlinked manufacturing processes based on supervised & unsupervised machine learning,” *Procedia Cirp*, vol. 7, pp. 193–198, 2013.

[46] M. Stolpe, K. Bhaduri, and K. Das, “Distributed support vector machines: An overview,” *Solving Large Scale Learning Tasks. Challenges and Algorithms*, pp. 109–138, 2016.

[47] P. Wiederkehr and T. Siebrecht, “Virtual machining: Capabilities and challenges of process simulations in the aerospace industry,” *Procedia Manufacturing*, vol. 6, pp. 80–87, 2016.

[48] J. Cuzick, “A wilcoxon-type test for trend,” *Statistics in medicine*, vol. 4, no. 1, pp. 87–90, 1985.

[49] I. Mierswa and K. Morik, “Automatic feature extraction for classifying audio data,” *Machine learning*, vol. 58, no. 2, pp. 127–149, 2005.

[50] A. A. Cook, G. Mısırlı, and Z. Fan, “Anomaly detection for iot time-series data: A survey,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2019.

[51] M. F. Ghalwash, V. Radosavljevic, and Z. Obradovic, “Extraction of interpretable multivariate patterns for early diagnostics,” in *2013 IEEE 13th International Conference on Data Mining*, IEEE, 2013, pp. 201–210.

[52] J. Asafu-Adjaye, “The relationship between energy consumption, energy prices and economic growth: Time series evidence from asian developing countries,” *Energy economics*, vol. 22, no. 6, pp. 615–625, 2000.

[53] C. Chatfield, *The analysis of time series: an introduction.* Chapman and hall/CRC, 2003.

[54] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice.* OTexts, 2018.

[55] P. S. Cowpertwait and A. V. Metcalfe, *Introductory time series with R.* Springer Science & Business Media, 2009.

[56] G. Rudebush, “On the power of dickey-fuller tests against fractional alternatives,” *Economics letters*, vol. 35, pp. 155–160, 1991.

[57] Y.-W. Cheung and K. S. Lai, “Lag order and critical values of the augmented dickey–fuller test,” *Journal of Business & Economic Statistics*, vol. 13, no. 3, pp. 277–280, 1995.

[58] J. Breitung and P. H. Franses, “On phillips–perron-type tests for seasonal unit roots,” *Econometric Theory*, vol. 14, no. 2, pp. 200–221, 1998.

[59] G. Elliott, T. J. Rothenberg, and J. H. Stock, *Efficient tests for an autoregressive unit root*, 1992.

[60] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, “Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?” *Journal of econometrics*, vol. 54, no. 1-3, pp. 159–178, 1992.

[61] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[62] E. Zivot and D. W. K. Andrews, “Further evidence on the great crash, the oil-price shock, and the unit-root hypothesis,” *Journal of business & economic statistics*, vol. 20, no. 1, pp. 25–44, 2002.

[63] S. Kanaya, “A nonparametric test for stationarity in continuous-time markov processes,” *Job Market Paper, University of Oxford*, 2011.

[64] A. van Delft, V. Characiejus, and H. Dette, “A nonparametric test for stationarity in functional time series,” *arXiv preprint arXiv:1708.05248*, 2017.

[65] R. J. Hyndman, G. Athanasopoulos, C. Bergmeir, G. Caceres, L. Chhay, M. O’Hara-Wild, F. Petropoulos, S. Razbash, and E. Wang, “Package ‘forecast’,” *Online] https://cran. r-project. org/web/packages/forecast/forecast. pdf*, 2020.

[66] C. Croarkin, P. Tobias, J. Filliben, B. Hembree, W. Guthrie, *et al.*, "Nist/sematech e-handbook of statistical methods," *NIST/SEMATECH, July. Available online: http://www. itl. nist. gov/div898/handbook*, p. 24, 2006.

[67] G. E. Box and D. R. Cox, "An analysis of transformations," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 26, no. 2, pp. 211–243, 1964.

[68] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: A novel symbolic representation of time series," *Data Mining and knowledge discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[69] D. Criado-Ramón, L. Ruiz, and M. Pegalajar, "Electric demand forecasting with neural networks and symbolic time series representations," *Applied Soft Computing*, vol. 122, p. 108 871, 2022.

[70] S. Elsworth and S. Güttel, "Time series forecasting using lstm networks: A symbolic approach," *arXiv preprint arXiv:2003.05672*, 2020.

[71] S. Lawrence, A. C. Tsoi, and C. L. Giles, "Noisy time series prediction using symbolic representation and recurrent neural network grammatical inference," Tech. Rep., 1998.

[72] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, 2003, pp. 2–11.

[73] S. Lhermitte, J. Verbesselt, W. W. Verstraeten, and P. Coppin, "A comparison of time series similarity measures for classification and change detection of ecosystem dynamics," *Remote sensing of environment*, vol. 115, no. 12, pp. 3129–3152, 2011.

[74] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[75] G. M. Mimmack, S. J. Mason, and J. S. Galpin, "Choice of distance matrices in cluster analysis: Defining regions," *Journal of climate*, vol. 14, no. 12, pp. 2790–2797, 2001.

[76] P. C. Mahalanobis, "On the generalized distance in statistics," National Institute of Science of India, 1936.

[77] T. K. Vintsyuk, "Speech discrimination by dynamic programming," *Cybernetics*, vol. 4, no. 1, pp. 52–57, 1968.

[78] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.

[79] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. Keogh, "Dynamic time warping averaging of time series allows faster and more accurate classification," in *2014 IEEE international conference on data mining*, IEEE, 2014, pp. 470–479.

[80] W. Nikolai, T. SCHLEGL, and J. DEUSE, "Feature extraction for time series classification using univariate descriptive statistics and dynamic time warping in a manufacturing environment," in *2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, IEEE, 2021, pp. 762–768.

[81] T. Górecki and M. Łuczak, "Multivariate time series classification with parametric derivative dynamic time warping," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2305–2312, 2015.

[82] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review," *Information systems*, vol. 53, pp. 16–38, 2015.

[83] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and information systems*, vol. 7, no. 3, pp. 358–386, 2005.

[84] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 262–270.

[85] J. Klemming, *Python Extension for the UCR-Suite*, 2018. [Online]. Available: `https://github.com/klon/ucrdtw`.

[86] R. N. Bracewell and R. N. Bracewell, *The Fourier transform and its applications*. McGraw-Hill New York, 1986, vol. 31999.

[87] J. C. Schlimmer and R. H. Granger, "Beyond incremental processing: Tracking concept drift.," in *AAAI*, 1986, pp. 502–507.

[88] V. Losing, B. Hammer, and H. Wersing, "Tackling heterogeneous concept drift with the self-adjusting memory (sam)," *Knowledge and Information Systems*, vol. 54, pp. 171–201, 2018.

[89] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[90] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, Springer, 2000, pp. 1–15.

[91] L. I. Kuncheva, "Classifier ensembles for changing environments," in *International Workshop on Multiple Classifier Systems*, Springer, 2004, pp. 1–15.

[92] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Locally adaptive dimensionality reduction for indexing large time series databases," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, 2001, pp. 151–162.

[93] E. J. Keogh and M. J. Pazzani, "An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback.," in *Kdd*, vol. 98, 1998, pp. 239–243.

[94] J. Lin, E. Keogh, P. Patel, and S. Lonardi, "Finding motifs in time series, in proceedings of the 2nd workshop on temporal data mining, at the 8th acm sigkdd international conference on knowledge discovery and data mining," *Edmonton, Alberta, Canada*, 2002.

[95] E. Keogh, S. Lonardi, and B.-c. Chiu, "Finding surprising patterns in a time series database in linear time and space," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 550–556.

[96] D. Dasgupta and S. Forrest, "Novelty detection in time series data using ideas from immunology," in *Proceedings of the international conference on intelligent systems*, Citeseer, 1996, pp. 82–87.

[97] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, "A state space framework for automatic forecasting using exponential smoothing methods," *International Journal of forecasting*, vol. 18, no. 3, pp. 439–454, 2002.

[98] R. Klinkenberg and T. Joachims, “Detecting concept drift with support vector machines.,” in *ICML*, 2000, pp. 487–494.

[99] R. Klinkenberg and S. Rüping, “Concept drift and the importance of examples,” in *Text mining–theoretical aspects and applications*, Citeseer, 2002.

[100] L. Noakes, “The takens embedding theorem,” *International Journal of Bifurcation and Chaos*, vol. 1, no. 04, pp. 867–872, 1991.

[101] G. Schwarz, “Estimating the dimension of a model,” *The annals of statistics*, pp. 461–464, 1978.

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

[103] T. O. Kvålseth, “Cautionary note about r 2,” *The American Statistician*, vol. 39, no. 4, pp. 279–285, 1985.

[104] T. Hastie, R. Tibshirani, and M. Wainwright, “Statistical learning with sparsity,” *Monographs on statistics and applied probability*, vol. 143, p. 143, 2015.

[105] R. G. Brown, “Statistical forecasting for inventory control,” 1959.

[106] E. Zivot and J. Wang, “Vector autoregressive models for multivariate time series,” *Modeling Financial Time Series with S-Plus®*, pp. 385–429, 2006.

[107] B. Lim and S. Zohren, “Time-series forecasting with deep learning: A survey,” *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2194, p. 20 200 209, 2021.

[108] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[109] P. E. Utgoff and D. J. Stracuzzi, “Many-layered learning,” *Neural computation*, vol. 14, no. 10, pp. 2497–2529, 2002.

[110] Q. Sun, M. V. Jankovic, L. Bally, and S. G. Mougiakakou, “Predicting blood glucose with an lstm and bi-lstm based deep neural network,” in *2018 14th Symposium on Neural Networks and Applications (NEUREL)*, IEEE, 2018, pp. 1–5.

[111] E. Hoseinzade and S. Haratizadeh, “Cnnpred: Cnn-based stock market prediction using a diverse set of variables,” *Expert Systems with Applications*, vol. 129, pp. 273–285, 2019, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2019.03.029. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419301915.

[112] P. Romeu, F. Zamora-Martınez, P. Botella-Rocamora, and J. Pardo, “Time-series forecasting of indoor temperature using pre-trained deep neural networks,” in *International conference on artificial neural networks*, Springer, 2013, pp. 451–458.

[113] J. C. B. Gamboa, “Deep learning for time-series analysis,” *arXiv preprint arXiv:1701.01887*, 2017.

[114] I. E. Livieris, E. Pintelas, and P. Pintelas, “A cnn–lstm model for gold price time-series forecasting,” *Neural Computing and Applications*, pp. 1–10, 2020.

[115] T.-Y. Kim and S.-B. Cho, “Predicting residential energy consumption using cnn-lstm neural networks,” *Energy*, vol. 182, pp. 72–81, 2019.

[116] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[117] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[118] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying lstm to time series predictable through time-window approaches," in *Neural Nets WIRN Vietri-01*, Springer, 2002, pp. 193–200.

[119] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5929–5955, 2020.

[120] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[121] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Transfer learning for time series classification," in *2018 IEEE international conference on big data (Big Data)*, IEEE, 2018, pp. 1367–1376.

[122] V. Cerqueira, L. Torgo, and I. Mozetič, "Evaluating time series forecasting models: An empirical study on performance estimation methods," *Machine Learning*, vol. 109, no. 11, pp. 1997–2028, 2020.

[123] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[124] J. Tayman and D. A. Swanson, "On the validity of mape as a measure of population forecast accuracy," *Population Research and Policy Review*, vol. 18, no. 4, pp. 299–322, 1999.

[125] C. Sammut and G. I. Webb, *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[126] Z. Chen and Y. Yang, "Assessing forecast accuracy measures," *Preprint Series*, vol. 2010, pp. 2004–10, 2004.

[127] C. Bergmeir and J. M. Benıtez, "On the use of cross-validation for time series predictor evaluation," *Information Sciences*, vol. 191, pp. 192–213, 2012.

[128] C. Bergmeir, R. J. Hyndman, and B. Koo, "A note on the validity of cross-validation for evaluating autoregressive time series prediction," *Computational Statistics & Data Analysis*, vol. 120, pp. 70–83, 2018.

[129] L. J. Tashman, "Out-of-sample tests of forecasting accuracy: An analysis and review," *International journal of forecasting*, vol. 16, no. 4, pp. 437–450, 2000.

[130] V. Cerqueira, L. Torgo, and C. Soares, "Model selection for time series forecasting: Empirical analysis of different estimators," *arXiv preprint arXiv:2104.00584*, 2021.

[131] S. M. Abdulrahman, P. Brazdil, J. N. van Rijn, and J. Vanschoren, "Speeding up algorithm selection using average ranking and active testing by introducing runtime," *Machine learning*, vol. 107, no. 1, pp. 79–108, 2018.

[132] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.

[133] J. Friedman, T. Hastie, and R. Tibshirani, “Regularization paths for generalized linear models via coordinate descent,” *Journal of statistical software*, vol. 33, no. 1, p. 1, 2010.

[134] J. Gama, P. P. Rodrigues, and R. Sebastiao, “Evaluating algorithms that learn from data streams,” in *Proceedings of the 2009 ACM symposium on Applied Computing*, 2009, pp. 1496–1500.

[135] R. Hyndman and Y. Yang, *Tsdl: Time series data library*, 2019.

[136] A. Benavoli, G. Corani, and F. Mangili, “Should we really use post-hoc tests based on mean-ranks?” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 152–161, 2016.

[137] L. Breiman and P. Spector, “Submodel selection and evaluation in regression. the x-random case,” *International statistical review/revue internationale de Statistique*, pp. 291–319, 1992.

[138] Y. Sakamoto, M. Ishiguro, and G. Kitagawa, “Akaike information criterion statistics,” *Dordrecht, The Netherlands: D. Reidel*, vol. 81, no. 10.5555, p. 26 853, 1986.

[139] D. L. Weakliem, “A critique of the bayesian information criterion for model selection,” *Sociological Methods & Research*, vol. 27, no. 3, pp. 359–397, 1999.

[140] V. Vapnik, “Principles of risk minimization for learning theory,” *Advances in neural information processing systems*, vol. 4, 1991.

[141] S. I. Vrieze, “Model selection and psychological theory: A discussion of the differences between the akaike information criterion (aic) and the bayesian information criterion (bic).,” *Psychological methods*, vol. 17, no. 2, p. 228, 2012.

[142] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, “The m4 competition: 100,000 time series and 61 forecasting methods,” *International Journal of Forecasting*, vol. 36, no. 1, pp. 54–74, 2020.

[143] W. Jamil and A. Bouchachia, “Model selection in online learning for times series forecasting,” in *UK Workshop on Computational Intelligence*, Springer, 2018, pp. 83–95.

[144] O. Anava, E. Hazan, S. Mannor, and O. Shamir, “Online learning for time series prediction,” in *Conference on learning theory*, PMLR, 2013, pp. 172–184.

[145] C. Liu, S. C. Hoi, P. Zhao, and J. Sun, “Online arima algorithms for time series prediction,” in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[146] R. Candela, P. Michiardi, M. Filippone, and M. A. Zuluaga, “Model monitoring and dynamic model selection in travel time-series forecasting,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2020, pp. 513–529.

[147] M. Aiolfi and A. Timmermann, “Persistence in forecasting performance and conditional combination strategies,” *Journal of Econometrics*, vol. 135, no. 1-2, pp. 31–53, 2006.

[148] G. Brown, J. L. Wyatt, and P. Tiňo, “Managing diversity in regression ensembles,” *Journal of machine learning research*, vol. 6, no. Sep, pp. 1621–1650, 2005.

[149] W. N. Street and Y. Kim, “A streaming ensemble algorithm (sea) for large-scale classification,” in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 377–382.

[150] A. Krogh and J. Vedelsby, “Neural network ensembles, cross validation, and active learning,” in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds., vol. 7, MIT Press, 1995. [Online]. Available: `https://proceedings.neurips.cc/paper/1994/file/b8c37e33defde51cf91e1e03e51657da-Paper.pdf`.

[151] G. Brown, “Ensemble learning.,” *Encyclopedia of Machine Learning*, vol. 312, 2010.

[152] G. Brown, J. Wyatt, R. Harris, and X. Yao, “Diversity creation methods: A survey and categorisation,” *Information fusion*, vol. 6, no. 1, pp. 5–20, 2005.

[153] Y. Zhang, S. Burer, W. Nick Street, K. P. Bennett, and E. Parrado-Hernández, “Ensemble pruning via semi-definite programming.,” *Journal of machine learning research*, vol. 7, no. 7, 2006.

[154] G. Martinez-Munoz, D. Hernández-Lobato, and A. Suárez, “An analysis of ensemble pruning techniques based on ordered aggregation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 245–259, 2008.

[155] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, “Ensemble selection from libraries of models,” in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 18.

[156] Y. Yu, Y.-F. Li, and Z.-H. Zhou, “Diversity regularized machine,” in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[157] A. V. Krikunov and S. V. Kovalchuk, “Dynamic selection of ensemble members in multi-model hydrometeorological ensemble forecasting,” *Procedia Computer Science*, vol. 66, pp. 220–227, 2015.

[158] Z.-H. Zhou, J. Wu, and W. Tang, “Ensembling neural networks: Many could be better than all,” *Artificial intelligence*, vol. 137, no. 1-2, pp. 239–263, 2002.

[159] T. Zhang, “Covering number bounds of certain regularized linear function classes,” *Journal of Machine Learning Research*, vol. 2, no. Mar, pp. 527–550, 2002.

[160] H. Chen, P. Tiňo, and X. Yao, “Predictive ensemble pruning by expectation propagation,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 7, pp. 999–1013, 2009.

[161] I. Partalas, G. Tsoumakas, and I. Vlahavas, “A study on greedy algorithms for ensemble pruning,” *Aristotle University of Thessaloniki, Thessaloniki, Greece*, 2012.

[162] Z. Ma, Q. Dai, and N. Liu, “Several novel evaluation measures for rank-based ensemble pruning with applications to time series prediction,” *Expert systems with applications*, vol. 42, no. 1, pp. 280–292, 2015.

[163] G. Martınez-Munoz and A. Suárez, “Aggregation ordering in bagging,” in *Proc. of the IASTED International Conference on Artificial Intelligence and Applications*, Citeseer, 2004, pp. 258–263.

[164] R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer, “Ensemble diversity measures and their application to thinning,” *Information Fusion*, vol. 6, no. 1, pp. 49–62, 2005.

[165] G. Giacinto, F. Roli, and G. Fumera, "Design of effective multiple classifier systems by clustering of classifiers," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, IEEE, vol. 2, 2000, pp. 160–163.

[166] A. Lazarevic and Z. Obradovic, "Effective pruning of neural network classifier ensembles," in *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, IEEE, vol. 2, 2001, pp. 796–801.

[167] Z.-H. Zhou and W. Tang, "Selective ensemble of decision trees," in *International workshop on rough sets, fuzzy sets, data mining, and granular-soft computing*, Springer, 2003, pp. 476–483.

[168] I. Partalas, G. Tsoumakas, I. Katakis, and I. Vlahavas, "Ensemble pruning using reinforcement learning," in *Hellenic Conference on Artificial Intelligence*, Springer, 2006, pp. 301–310.

[169] J. Zhang, Q. Dai, and C. Yao, "Dep-tsp meta: A multiple criteria dynamic ensemble pruning technique ad-hoc for time series prediction," *International Journal of Machine Learning and Cybernetics*, pp. 1–24, 2021.

[170] H. Yu and G. I. Webb, "Adaptive online extreme learning machine by regulating forgetting factor by concept drift map," *Neurocomputing*, vol. 343, pp. 141–153, 2019.

[171] J. Y. Choi and B. Lee, "Combining lstm network ensemble via adaptive weighting for improved time series forecasting," *Mathematical Problems in Engineering*, vol. 2018, 2018.

[172] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby, "Selective sampling using the query by committee algorithm," *Machine learning*, vol. 28, no. 2-3, pp. 133–168, 1997.

[173] P. Gaillard, G. Stoltz, and T. Van Erven, "A second-order bound with excess losses," in *Conference on Learning Theory*, PMLR, 2014, pp. 176–196.

[174] P. Gaillard and Y. Goude, "Forecasting electricity consumption by aggregating experts; how to design a good set of experts," in *Modeling and stochastic learning for forecasting in high dimensions*, Springer, 2015, pp. 95–115.

[175] O. Wintenberger, "Optimal learning with bernstein online aggregation," *Machine Learning*, vol. 106, no. 1, pp. 119–141, 2017.

[176] M. Devaine, P. Gaillard, Y. Goude, and G. Stoltz, "Forecasting electricity consumption by aggregating specialized experts," *Machine Learning*, vol. 90, no. 2, pp. 231–260, 2013.

[177] P. Gaillard, "Contributions to online robust aggregation: Work on the approximation error and on probabilistic forecasting. applications to forecasting for energy markets.," *HAL*, vol. 2015, 2015.

[178] V. Vovk, "Aggregating strategies," in *COLT '90*, 1990.

[179] M. Herbster and M. K. Warmuth, "Tracking the best expert," *Machine learning*, vol. 32, no. 2, pp. 151–178, 1998.

[180] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*. Cambridge university press, 2006.

[181] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 421–436.

[182] L. Todorovski and S. Džeroski, “Combining classifiers with meta decision trees,” *Machine learning*, vol. 50, no. 3, pp. 223–249, 2003.

[183] J. R. Quinlan *et al.*, “Bagging, boosting, and c4. 5,” in *Aaai/Iaai, vol. 1*, 1996, pp. 725–730.

[184] C. D. Sutton, “Classification and regression trees, bagging, and boosting,” *Handbook of statistics*, vol. 24, pp. 303–329, 2005.

[185] N. C. Oza and S. J. Russell, “Online bagging and boosting,” in *International Workshop on Artificial Intelligence and Statistics*, PMLR, 2001, pp. 229–236.

[186] M. H. D. M. Ribeiro and L. dos Santos Coelho, “Ensemble approach based on bagging, boosting and stacking for short-term prediction in agribusiness time series,” *Applied Soft Computing*, vol. 86, p. 105 837, 2020.

[187] F. Petropoulos, R. J. Hyndman, and C. Bergmeir, “Exploring the sources of uncertainty: Why does bagging for time series forecasting work?” *European Journal of Operational Research*, vol. 268, no. 2, pp. 545–554, 2018.

[188] T. Hastie and R. Tibshirani, “Generalized additive models: Some applications,” *Journal of the American Statistical Association*, vol. 82, no. 398, pp. 371–386, 1987.

[189] Y. Freund, R. Schapire, and N. Abe, “A short introduction to boosting,” *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[190] R. E. Schapire, “The boosting approach to machine learning: An overview,” *Nonlinear estimation and classification*, pp. 149–171, 2003.

[191] P. Bühlmann and B. Yu, “Boosting with the l 2 loss: Regression and classification,” *Journal of the American Statistical Association*, vol. 98, no. 462, pp. 324–339, 2003.

[192] J.-B. Lamy, B. Sekar, G. Guezennec, J. Bouaud, and B. Séroussi, “Explainable artificial intelligence for breast cancer: A visual case-based reasoning approach,” *Artificial intelligence in medicine*, vol. 94, pp. 42–53, 2019.

[193] E. Tjoa and C. Guan, “A survey on explainable artificial intelligence (xai): Toward medical xai,” *IEEE transactions on neural networks and learning systems*, vol. 32, no. 11, pp. 4793–4813, 2020.

[194] É. Zablocki, H. Ben-Younes, P. Pérez, and M. Cord, “Explainability of vision-based autonomous driving systems: Review and challenges,” *arXiv preprint arXiv:2101.05307*, 2021.

[195] M. T. Ribeiro, S. Singh, and C. Guestrin, “" why should i trust you?" explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[196] T. Miller, “Explanation in artificial intelligence: Insights from the social sciences,” *Artificial intelligence*, vol. 267, pp. 1–38, 2019.

[197] B. Kim, R. Khanna, and O. O. Koyejo, “Examples are not enough, learn to criticize! criticism for interpretability,” *Advances in neural information processing systems*, vol. 29, 2016.

[198] F. K. Došilović, M. Brčić, and N. Hlupić, “Explainable artificial intelligence: A survey,” in *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, IEEE, 2018, pp. 0210–0215.

[199] A. Kharal, “Explainable artificial intelligence based fault diagnosis and insight harvesting for steel plates manufacturing,” *arXiv preprint arXiv:2008.04448*, 2020.

[200] L. C. Brito, G. A. Susto, J. N. Brito, and M. A. Duarte, “An explainable artificial intelligence approach for unsupervised fault detection and diagnosis in rotating machinery,” *Mechanical Systems and Signal Processing*, vol. 163, p. 108 105, 2022.

[201] M. Sundararajan and A. Najmi, “The many shapley values for model explanation,” in *International conference on machine learning*, PMLR, 2020, pp. 9269–9278.

[202] A. Inglis, A. Parnell, and C. B. Hurley, “Visualizing variable importance and variable interaction effects in machine learning models,” *Journal of Computational and Graphical Statistics*, pp. 1–13, 2022.

[203] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[204] A. Saadallah and K. Morik, “Active sampling for learning interpretable surrogate machine learning models,” in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2020, pp. 264–272.

[205] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: `http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf` (visited on 02/06/2020).

[206] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML’17, Sydney, NSW, Australia: JMLR.org, Aug. 6, 2017, pp. 3319–3328. (visited on 02/18/2020).

[207] Z. Chen, Y. Bei, and C. Rudin, “Concept whitening for interpretable image recognition,” *Nature Machine Intelligence*, vol. 2, no. 12, pp. 772–782, Dec. 1, 2020, ISSN: 2522-5839. DOI: `10.1038/s42256-020-00265-z`. [Online]. Available: `https://doi.org/10.1038/s42256-020-00265-z`.

[208] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, “Sanity checks for saliency maps,” *arXiv preprint arXiv:1810.03292*, 2018.

[209] O.-M. Camburu, “Explaining deep neural networks,” *arXiv preprint arXiv:2010.01496*, 2020.

[210] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, “Explaining deep neural networks and beyond: A review of methods and applications,” *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, 2021.

[211] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, *et al.*, “Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav),” in *International conference on machine learning*, PMLR, 2018, pp. 2668–2677.

[212] F. Küsters, P. Schichtel, S. Ahmed, and A. Dengel, “Conceptual explanations of neural network prediction for time series,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–6.

[213] N. Frosst and G. Hinton, “Distilling a neural network into a soft decision tree,” *arXiv preprint arXiv:1711.09784*, 2017.

[214] X. Cheng, Z. Rao, Y. Chen, and Q. Zhang, “Explaining knowledge distillation by quantifying the knowledge,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 925–12 935.

[215] C. Olah, A. Mordvintsev, and L. Schubert, “Feature visualization,” *Distill*, vol. 2, no. 11, e7, 2017.

[216] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps,” *arXiv preprint arXiv:1312.6034*, 2013.

[217] R. Assaf and A. Schumann, “Explainable deep neural networks for multivariate time series predictions.,” in *IJCAI*, 2019, pp. 6488–6490.

[218] R. Assaf, I. Giurgiu, F. Bagehorn, and A. Schumann, “Mtex-cnn: Multivariate time series explanations for predictions with convolutional neural networks,” in *2019 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2019, pp. 952–957.

[219] M. Muschalik, F. Fumagalli, B. Hammer, and E. Hüllermeier, “Agnostic explanation of model change based on feature importance,” *KI-Künstliche Intelligenz*, pp. 1–14, 2022.

[220] B. Hammer and E. Hüllermeier, “Interpretable machine learning: On the problem of explaining model change,” in *PROCEEDINGS 31. WORKSHOP COMPUTATIONAL INTELLIGENCE*, vol. 25, 2021, p. 1.

[221] A. Artelt, F. Hinder, V. Vaquet, R. Feldhans, and B. Hammer, “Contrastive explanations for explaining model adaptations,” in *International Work-Conference on Artificial Neural Networks*, Springer, 2021, pp. 101–112.

[222] ——, “Contrasting explanations for understanding and regularizing model adaptations,” *Neural Processing Letters*, pp. 1–25, 2022.

[223] I. Ahmed, G. Jeon, and F. Piccialli, “From artificial intelligence to explainable artificial intelligence in industry 4.0: A survey on what, how, and where,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 8, pp. 5031–5042, 2022.

[224] J. Schmitt, J. Bönig, T. Borggräfe, G. Beitinger, and J. Deuse, “Predictive model-based quality inspection using machine learning and edge cloud computing,” *Advanced engineering informatics*, vol. 45, p. 101 101, 2020.

[225] D. Lieber, B. Konrad, J. Deuse, M. Stolpe, and K. Morik, “Sustainable interlinked manufacturing processes through real-time quality prediction,” in *Leveraging Technology for a Sustainable World*, Springer, 2012, pp. 393–398.

[226] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[227] G. Forestier, F. Petitjean, H. A. Dau, G. I. Webb, and E. Keogh, "Generating synthetic time series to augment sparse datasets," in *2017 IEEE international conference on data mining (ICDM)*, IEEE, 2017, pp. 865–870.

[228] D. Lahat, T. Adali, and C. Jutten, "Multimodal data fusion: An overview of methods, challenges, and prospects," *Proceedings of the IEEE*, vol. 103, no. 9, pp. 1449–1477, 2015.

[229] A. Saadallah, F. Finkeldey, K. Morik, and P. Wiederkehr, "Stability prediction in milling processes using a simulation-based machine learning approach," *Procedia CIRP*, vol. 72, pp. 1493–1498, 2018.

[230] P. M. Grulich, J. Traub, S. Breß, A. Katsifodimos, V. Markl, and T. Rabl, "Generating reproducible out-of-order data streams," in *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '19, Darmstadt, Germany: ACM, 2019, pp. 256–257, ISBN: 978-1-4503-6794-3. DOI: `10.1145/3328905.3332511`. [Online]. Available: `http://doi.acm.org/10.1145/3328905.3332511`.

[231] A. Biland *et al.*, "Calibration and performance of the photon sensor response of FACT the first G-APD Cherenkov telescope," *Journal of Instrumentation*, vol. 9, no. 10, P10012–P10012, 2014, ISSN: 1748-0221. DOI: `10.1088/1748-0221/9/10/P10012`. [Online]. Available: `http://stacks.iop.org/1748-0221/9/i=10/a=P10012`.

[232] Y. Fischer and A. Bauer, "Object-oriented sensor data fusion for wide maritime surveillance," in *2010 International WaterSide Security Conference*, IEEE, 2010, pp. 1–6.

[233] M. Bunse, A. Saadallah, and K. Morik, "Towards active simulation data mining," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) Workshops*, 2019, p. 104.

[234] Fischer, C., "Runtime and accuracy issues in three-dimensional finite element simulation of machining," *International Journal of Machining and Machinability of Materials*, vol. 6, no. 1/2, p. 35, 2009, ISSN: 1748-5711 (P), 1748-572X (E).

[235] M. Tang, C. Yang, J. Yan, and Q. Yue, "Validity and limitation of analytical models for the bending stress of a helical wire in unbonded flexible pipes," *Applied Ocean Research*, vol. 50, pp. 58–68, 2015, ISSN: 0141-1187. DOI: `10.1016/j.apor.2014.12.004`.

[236] G. Meschke, B. Cao, A. Egorov, A. Saadallah, S. Freitag, and K. Morik., "Big data and simulation- a new approach for real-time tbm steering.," *Peila, D.; Viggiani, G.; Celestino, T. (eds.), Tunnels and Underground Cities. Engineering and Innovation Meet Archaeology, Architecture and Art, Proceedings of the WTC 2019 ITA-AITES World Tunnel Congress (WTC 2019), Naples, Italy, Taylor and Francis, London*, pp. 2681–2690, 2019.

[237] B.-T. Cao, S. Freitag, and G. Meschke, "A hybrid rnn-gpod surrogate model for real-time settlement predictions in mechanised tunnelling," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 3, no. 1, p. 5, 2016.

[238] I. Goldenberg and G. I. Webb, "Pca-based drift and shift quantification framework for multidimensional data," *Knowledge and Information Systems*, vol. 62, no. 7, pp. 2835–2854, 2020.

[239] A. Dempster, D. F. Schmidt, and G. I. Webb, "Minirocket: A very fast (almost) deterministic transform for time series classification," in *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021, pp. 248–257.

[240] A. Dempster, F. Petitjean, and G. I. Webb, “Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels,” *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1454–1495, 2020.

[241] X. Wang, A. Wirth, and L. Wang, “Structure-based statistical features and multivariate time series clustering,” in *Seventh IEEE international conference on data mining (ICDM 2007)*, IEEE, 2007, pp. 351–360.

[242] T. Räsänen and M. Kolehmainen, “Feature-based clustering for electricity use time series data,” in *International conference on adaptive and natural computing algorithms*, Springer, 2009, pp. 401–412.

[243] W. He, Z. Wang, and H. Jiang, “Model optimizing and feature selecting for support vector regression in time series forecasting,” *Neurocomputing*, vol. 72, no. 1-3, pp. 600–611, 2008.

[244] S. Du, T. Li, Y. Yang, and S.-J. Horng, “Multivariate time series forecasting via attention-based encoder–decoder framework,” *Neurocomputing*, vol. 388, pp. 269–279, 2020.

[245] L. Munkhdalai, T. Munkhdalai, K. H. Park, T. Amarbayasgalan, E. Batbaatar, H. W. Park, and K. H. Ryu, “An end-to-end adaptive input selection with dynamic weights for forecasting multivariate time series,” *IEEE Access*, vol. 7, pp. 99 099–99 114, 2019.

[246] X. Wang and M. Han, “Improved extreme learning machine for multivariate time series online sequential prediction,” *Engineering Applications of Artificial Intelligence*, vol. 40, pp. 28–36, 2015.

[247] A. Gonzalez-Vidal, F. Jimenez, and A. F. Gomez-Skarmeta, “A methodology for energy multivariate time series forecasting in smart buildings based on feature selection,” *Energy and Buildings*, vol. 196, pp. 71–82, 2019, ISSN: 0378-7788. DOI: `https://doi.org/10.1016/j.enbuild.2019.05.021`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0378778818338775`.

[248] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, “Connecting the dots: Multivariate time series forecasting with graph neural networks,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 753–763.

[249] A. Sagheer and M. Kotb, “Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems,” *Scientific reports*, vol. 9, no. 1, pp. 1–16, 2019.

[250] C. Ding and H. Peng, “Minimum redundancy feature selection from microarray gene expression data,” *Journal of bioinformatics and computational biology*, vol. 3, no. 02, pp. 185–205, 2005.

[251] G. Gulgezen, Z. Cataltepe, and L. Yu, “Stable and accurate feature selection,” in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part I 20*, Springer, 2009, pp. 455–468.

[252] K. Michalak and H. Kwaśnicka, “Correlation-based feature selection strategy in classification problems,” *International Journal of Applied Mathematics and Computer Science*, vol. 16, no. 4, pp. 503–511, 2006.

[253] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *The Journal of Machine Learning Research*, vol. 5, pp. 1205–1224, 2004.

[254] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: A review," *Acm sigkdd explorations newsletter*, vol. 6, no. 1, pp. 90–105, 2004.

[255] G. Jurman, S. Merler, A. Barla, S. Paoli, A. Galea, and C. Furlanello, "Algebraic stability indicators for ranked lists in molecular profiling," *Bioinformatics*, vol. 24, no. 2, pp. 258–264, 2008.

[256] B. Schowe and K. Morik, "Fast-ensembles of minimum redundancy feature selection," *Ensembles in Machine Learning Applications*, pp. 75–95, 2011.

[257] V. Cerqueira, L. Torgo, J. Smailović, and I. Mozetič, "A comparative study of performance estimation methods for time series forecasting," in *IEEE Int. Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2017, pp. 529–538.

[258] D. J. MacKay *et al.*, "Introduction to gaussian processes," *NATO ASI series F computer and systems sciences*, vol. 168, pp. 133–166, 1998.

[259] G. Melki, A. Cano, V. Kecman, and S. Ventura, "Multi-target support vector regression via correlation regressor chains," *Information Sciences*, vol. 415, pp. 53–69, 2017.

[260] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review," *Information Systems*, vol. 53, pp. 16–38, 2015.

[261] A. Sardá-Espinosa, "Comparing time-series clustering algorithms in r using the dtwclust package," *R package vignette*, vol. 12, p. 41, 2017.

[262] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works of Wassily Hoeffding*, Springer, 1994, pp. 409–426.

[263] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[264] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in neural information processing systems*, 1997, pp. 155–161.

[265] J. H. Friedman and W. Stuetzle, "Projection pursuit regression," *Journal of the American statistical Association*, vol. 76, no. 376, pp. 817–823, 1981.

[266] J. H. Friedman *et al.*, "Multivariate adaptive regression splines," *The annals of statistics*, vol. 19, no. 1, pp. 1–67, 1991.

[267] B. Mevik and R. Wehrens, "Introduction to the pls package," *Help Section of The "Pls" Package of R Studio Software*, pp. 1–23, 2015.

[268] T. Koskela, M. Lehtokangas, J. Saarinen, and K. Kaski, "Time series prediction with multilayer perceptron, fir and elman neural networks," in *Proceedings of the world congress on neural networks*, Citeseer, 1996, pp. 491–496.

[269] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, 2015, pp. 802–810.

[270] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*, Springer, 1992, pp. 196–202.

[271] A. Saadallah, M. Jakobs, and K. 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*, N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, and J. A. Lozano, Eds., Cham: Springer International Publishing, 2021, pp. 404–420, ISBN: 978-3-030-86486-6.

[272] L. Birgé and P. Massart, "Gaussian model selection," *Journal of the European Mathematical Society*, vol. 3, no. 3, pp. 203–268, 2001.

[273] R. Godahewa, C. Bergmeir, G. I. Webb, and P. Montero-Manso, "An accurate and fully-automated ensemble model for weekly time series forecasting," *International Journal of Forecasting*, 2022.

[274] V. Assimakopoulos and K. Nikolopoulos, "The theta model: A decomposition approach to forecasting," *International journal of forecasting*, vol. 16, no. 4, pp. 521–530, 2000.

[275] A. M. De Livera, R. J. Hyndman, and R. D. Snyder, "Forecasting time series with complex seasonal patterns using exponential smoothing," *Journal of the American statistical association*, vol. 106, no. 496, pp. 1513–1527, 2011.

[276] P. Gaillard and Y. Goude, *Opera: Online prediction by expert aggregation*, R package version 1.0, 2016. [Online]. Available: `https://CRAN.R-project.org/package=opera`.

[277] N. Ueda and R. Nakano, "Generalization Error of Ensemble Estimators," in *Neural Networks, 1996., IEEE International Conference on*, 1996, pp. 90–95.

[278] T. W. Liao, "Clustering of time series data—a survey," *Pattern recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.

[279] P. Coretto and C. Hennig, "Robust improper maximum likelihood: Tuning, computation, and a comparison with other methods for robust gaussian clustering," *Journal of the American Statistical Association*, vol. 111, no. 516, pp. 1648–1659, 2016.

[280] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[281] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, 3. MIT Press Cambridge, MA, 2006, vol. 2.

[282] R. Wehrens and B.-H. Mevik, "The pls package: Principal component and partial least squares regression in r," 2007.

[283] J.-W. Kim, J.-S. Jang, M.-S. Yang, J.-H. Kang, K.-W. Kim, Y.-J. Cho, and J.-W. Lee, "A study on fault classification of machining center using acceleration data based on 1d cnn algorithm," *Journal of the Korean Society of Manufacturing Process Engineers*, vol. 18, no. 9, pp. 29–35, 2019.

[284] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the 20th international conference on machine learning (icml-03)*, 2003, pp. 928–936.

[285] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006, ISSN: 1532-4435.

[286] S. B. Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition," *Expert systems with applications*, vol. 39, no. 8, pp. 7067–7083, 2012.

[287] S. Zhang, Y. Chen, W. Zhang, and R. Feng, “A novel ensemble deep learning model with dynamic error correction and multi-objective ensemble pruning for time series forecasting,” *Information Sciences*, vol. 544, pp. 427–445, 2021.

[288] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning deep features for discriminative localization,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921–2929.

[289] K. Demertzis, L. Iliadis, and V.-D. Anezakis, “A deep spiking machine-hearing system for the case of invasive fish species,” in *2017 IEEE International Conference on Innovations in Intelligent Systems and Applications (INISTA)*, IEEE, 2017, pp. 23–28.

[290] A. Borovykh, S. Bohte, and C. W. Oosterlee, “Conditional time series forecasting with convolutional neural networks,” *arXiv preprint arXiv:1703.04691*, 2017.

[291] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” *arXiv preprint arXiv:1511.07122*, 2015.

[292] R. Mittelman, “Time-series modeling with undecimated fully convolutional neural networks,” *arXiv preprint arXiv:1508.00317*, 2015.

[293] M. Binkowski, G. Marti, and P. Donnat, “Autoregressive convolutional neural networks for asynchronous time series,” in *International Conference on Machine Learning*, PMLR, 2018, pp. 580–589.

[294] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, “Adaptive mixtures of local experts,” *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[295] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, “Deepar: Probabilistic forecasting with autoregressive recurrent networks,” *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020.

[296] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, “N-beats: Neural basis expansion analysis for interpretable time series forecasting,” *arXiv preprint arXiv:1905.10437*, 2019.

[297] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[298] J. Burkardt, “K-means clustering,” *Virginia Tech, Advanced Research Computing, Interdisciplinary Center for Applied Mathematics*, 2009.

[299] A. Mozaffari and N. L. Azad, “Optimally pruned extreme learning machine with ensemble of regularization techniques and negative correlation penalty applied to automotive engine coldstart hydrocarbon emission identification,” *Neurocomputing*, vol. 131, pp. 143–156, 2014.

[300] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual explanations from deep networks via gradient-based localization,” *International Journal of Computer Vision*, vol. 128, no. 2, pp. 336–359, 2019, ISSN: 1573-1405. DOI: `10.1007/s11263-019-01228-7`. [Online]. Available: `https://doi.org/10.1007/s11263-019-01228-7` (visited on 02/26/2020).

[301] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[302] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 1126–1135.

[303] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, “A simple neural attentive meta-learner,” in *International Conference on Learning Representations*, 2018.

[304] I. Partalas, G. Tsoumakas, and I. Vlahavas, “Pruning an ensemble of classifiers via reinforcement learning,” *Neurocomputing*, vol. 72, no. 7-9, pp. 1900–1909, 2009.

[305] C. Feng and J. Zhang, “Reinforcement learning based dynamic model selection for short-term load forecasting,” in *2019 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, IEEE, 2019, pp. 1–5.

[306] R. S. Sutton and A. G. Barto, *Reinforcement learning*, 1998.

[307] E. S. Page, “Continuous inspection schemes,” *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[308] R. T. Clemen and R. L. Winkler, “Combining economic forecasts,” *Journal of Business & Economic Statistics*, vol. 4, no. 1, pp. 39–46, 1986.

[309] J. Yan and L. Li, “Multi-objective optimization of milling parameters â the trade-offs between energy, production rate and cutting quality,” *Journal of Cleaner Production*, vol. 52, pp. 462–471, 2013, ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2013.02.030.

[310] C. Li, X. Chen, Y. Tang, and L. Li, “Selection of optimum parameters in multi-pass face milling for maximum energy efficiency and minimum production cost,” *Journal of Cleaner Production*, vol. 140, pp. 1805–1818, 2017, ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2016.07.086.

[311] F. Finkeldey, S. Hess, and P. Wiederkehr, “Tool wear-dependent process analysis by means of a statistical online monitoring system,” *Production Engineering*, vol. 11, no. 6, pp. 667–686, Oct. 2017, ISSN: 0944-6524. DOI: 10.1007/s11740-017-0773-0.

[312] M. Salehi, P. Albertelli, M. Goletti, F. Ripamonti, G. Tomasini, and M. Monno, “Indirect model based estimation of cutting force and tool tip vibrational behavior in milling machines by sensor fusion,” *Procedia CIRP*, vol. 33, pp. 239–244, 2015, ISSN: 2212-8271. DOI: 10.1016/j.procir.2015.06.043.

[313] C. Drouillet, J. Karandikar, C. Nath, A.-C. Journeaux, M. El Mansori, and T. Kurfess, “Tool life predictions in milling using spindle power with the neural network technique,” *Journal of Manufacturing Processes*, vol. 22, pp. 161–168, 2016. DOI: 10.1016/j.jmapro.2016.03.010.

[314] F. Klocke, *Manufacturing Processes 1 – Cutting*. Springer, 2011. DOI: 10.1007/978-3-642-11979-8.

[315] M. M. de Aguiar, A. E. Diniz, and R. Pederiva, “Correlating surface roughness, tool wear and tool vibration in the milling process of hardened steel using long slender tools,” *International Journal of Machine Tools and Manufacture*, vol. 68, pp. 1–10, 2013, ISSN: 0890-6955. DOI: 10.1016/j.ijmachtools.2013.01.002.

[316] F. Cus, M. Milfelner, and J. Balic, “An intelligent system for monitoring and optimization of ball-end milling process,” *Journal of Materials Processing Technology*, vol. 175, no. 1-3, pp. 90–97, 2006.

[317] H. Saglam and A. Unuvar, “Tool condition monitoring in milling based on cutting forces by a neural network,” *International Journal of Production Research*, vol. 41, no. 7, pp. 1519–1532, 2003.

[318] P. Huang, J. Li, J. Sun, and J. Zhou, “Vibration analysis in milling titanium alloy based on signal processing of cutting force,” *The International Journal of Advanced Manufacturing Technology*, vol. 64, no. 5-8, pp. 613–621, 2013.

[319] T. Herpel, C. Lauer, R. German, and J. Salzberger, “Multi-sensor data fusion in automotive applications,” in *2008 3rd International Conference on Sensing Technology*, IEEE, 2008, pp. 206–211.

[320] P. Valageas, “Accuracy of analytical models of the large-scale matter distribution,” *Phys. Rev. D*, vol. 88, p. 083 524, 8 Oct. 2013. DOI: `10.1103/PhysRevD.88.083524`.

[321] P. Pohanka, J. Hrabovsk, and M. Fiedler, “Sensors simulation environment for sensor data fusion,” in *14th International Conference on Information Fusion*, IEEE, 2011, pp. 1–8.

[322] B. Chen, R. Jiang, T. Kasetkasem, and P. K. Varshney, “Channel aware decision fusion in wireless sensor networks,” *IEEE Transactions on Signal Processing*, vol. 52, no. 12, pp. 3454–3458, 2004.

[323] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[324] C. Esteban, S. L. Hyland, and G. Rätsch, “Real-valued (medical) time series generation with recurrent conditional gans,” *arXiv preprint arXiv:1706.02633*, 2017.

[325] N. Kaempchen and K. Dietmayer, “Data synchronization strategies for multi-sensor fusion,” in *Proceedings of the IEEE Conference on Intelligent Transportation Systems*, vol. 85, 2003, pp. 1–9.

[326] T. Huck, A. Westenberger, M. Fritzsche, T. Schwarz, and K. Dietmayer, “Precise timestamping and temporal synchronization in multi-sensor fusion,” in *2011 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2011, pp. 242–247.

[327] K. Morik and W. (Rhode, “Technical report for collaborative research center sfb 876 - graduate school,” TU Dortmund University, Tech. Rep., 2019.

[328] K. Yamanishi and J. Takeuchi, “A unifying framework for detecting outliers and change points from non-stationary time series data,” in *Proceedings of the Eighth ACM SIGKDD*, ser. KDD ’02, New York, NY, USA: ACM, 2002.

[329] W. K. Ngui, M. S. Leong, L. M. Hee, and A. M. Abdelrhman, “Wavelet analysis: Mother wavelet selection methods,” in *Advances in Manufacturing and Mechanical Engineering*, vol. 393, Nov. 2013, pp. 953–958.

[330] K. Fukumizu, L. Song, and A. Gretton, “Kernel bayes’ rule: Bayesian inference with positive definite kernels,” *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 3753–3783, 2013.

[331] Y. Chen, M. Welling, and A. Smola, “Super-samples from kernel herding,” *arXiv preprint arXiv:1203.3472*, 2012.

[332] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. Sci. Comput.*, vol. 16, no. 5, pp. 1190–1208, Sep. 1995, ISSN: 1064-8275. DOI: 10.1137/0916069.

[333] S. Matzka and R. Altendorfer, "A comparison of track-to-track fusion algorithms for automotive sensor fusion," in *Multisensor Fusion and Integration for Intelligent Systems*, Springer, 2009, pp. 69–81.

[334] B. Krishnapuram, A. Harternink, L. Carin, and M. A. Figueiredo, "A bayesian approach to joint feature selection and classifier design," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1105–1111, 2004.

[335] I.-S. Oh, J.-S. Lee, and B.-R. Moon, "Hybrid genetic algorithms for feature selection," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 26, no. 11, pp. 1424–1437, 2004.

[336] H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining.* Kluwer Academic Publishers, 1998.

[337] A. Severyn and A. Moschitti, "Automatic feature engineering for answer selection and extraction," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013, pp. 458–467.

[338] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2015, pp. 1–10.

[339] M. Hofmann and R. Klinkenberg, *RapidMiner: Data mining use cases and business analytics applications.* CRC Press, 2013.

[340] S. Nogueira, K. Sechidis, and G. Brown, "On the stability of feature selection algorithms.," *Journal of Machine Learning Research*, vol. 18, pp. 174–1, 2017.

[341] P. K. Varshney, "Multisensor data fusion," *Electronics & Communication Engineering Journal*, vol. 9, no. 6, pp. 245–253, 1997.

[342] M. H. Ko, G. West, S. Venkatesh, and M. Kumar, "Using dynamic time warping for online temporal fusion in multisensor systems," *Information Fusion*, vol. 9, no. 3, pp. 370–388, 2008.

[343] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[344] F. Finkeldey, A. Saadallah, P. Wiederkehr, and K. Morik, "Real-time prediction of process forces in milling operations using synchronized data fusion of simulation and sensor data," *Engineering Applications of Artificial Intelligence*, vol. 94, p. 103 753, 2020.

[345] Z. Wang, B. Zhao, H. Guo, L. Tang, and Y. Peng, "Deep ensemble learning model for short-term load forecasting within active learning framework," *Energies*, vol. 12, no. 20, p. 3809, 2019.

[346] S. Palani, S.-Y. Liong, and P. Tkalich, "An ann application for water quality forecasting," *Marine pollution bulletin*, vol. 56, no. 9, pp. 1586–1597, 2008.

[347] Z. Lv, Y. Han, A. K. Singh, G. Manogaran, and H. Lv, "Trustworthiness in industrial iot systems based on artificial intelligence," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 2, pp. 1496–1504, 2020.

[348] R. S. Shoberg, "Engineering fundamentals of threaded fastener design and analysis. i," *Fastening*, vol. 6, no. 2, pp. 26–29, 2000.

[349] G. Wang, A. Ledwoch, R. M. Hasani, R. Grosu, and A. Brintrup, "A generative neural network model for the quality prediction of work in progress products," *Applied Soft Computing*, vol. 85, p. 105 683, 2019.

[350] N. V. Chawla, "Data mining for imbalanced datasets: An overview," *Data mining and knowledge discovery handbook*, pp. 875–886, 2009.

[351] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[352] L. Li, "Bottleneck detection of complex manufacturing systems using a data-driven method," *International Journal of Production Research*, vol. 47, no. 24, pp. 6929–6940, 2009.

[353] J. Wang, J. Li, J. Arinez, and S. Biller, "Quality bottleneck transitions in flexible manufacturing systems with batch productions," *IIE Transactions*, vol. 45, no. 2, pp. 190–205, 2013.

[354] G. Köksal, İ. Batmaz, and M. C. Testik, "A review of data mining applications for quality improvement in manufacturing industry," *Expert Systems with Applications*, vol. 38, no. 10, pp. 13 448–13 467, 2011, ISSN: 0957-4174. DOI: `https://doi.org/10.1016/j.eswa.2011.04.063`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0957417411005793`.

[355] Y. Li, K. Mohan, H. Sun, and R. Jin, "Ensemble modeling of in situ features for printed electronics manufacturing with in situ process control potential," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 1864–1870, 2017.

[356] S. Stoyanov and C. Bailey, "Machine learning for additive manufacturing of electronics," in *2017 40th international spring seminar on electronics technology (ISSE)*, IEEE, 2017, pp. 1–6.

[357] A. Saadallah, N. Piatkowski, F. Finkeldey, P. Wiederkehr, and K. Morik, "Learning ensembles in the presence of imbalanced classes.," in *ICPRAM*, 2019, pp. 866–873.

[358] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," *Advances in neural information processing systems*, vol. 28, 2015.

[359] T. Gneiting and M. Katzfuss, "Probabilistic forecasting," *Annual Review of Statistics and Its Application*, vol. 1, pp. 125–151, 2014.

[360] L. Torgo and O. Ohashi, "2d-interval predictions for time series," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 787–794.

[361] S. B. Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition," *Expert systems with applications*, vol. 39, no. 8, pp. 7067–7083, 2012.

[362] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, pp. 606–660, 3 2017.

[363] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: `http://archive.ics.uci.edu/ml`.

[364] A. Karatzoglou, A. Smola, K. Hornik, and M. A. Karatzoglou, “Package ‘kernlab’,” *CRAN R Project*, 2019.

[365] D. Milborrow, “Wind power on the grid,” in *Renewable Electricity and the Grid*, Routledge, 2012, pp. 52–75.

[366] P. J. Green and B. W. Silverman, *Nonparametric regression and generalized linear models: a roughness penalty approach.* Crc Press, 1993.

[367] R. C. Team *et al.*, “R: A language and environment for statistical computing,” 2013.

[368] J. Padarian, B. Minasny, A. McBratney, and N. Dalgliesh, “Predicting and mapping the soil available water capacity of australian wheatbelt,” *Geoderma Regional*, vol. 2, pp. 110–118, 2014.

[369] J. Elith and J. Leathwick, “Boosted regression trees for ecological modeling,” *R Documentation. Available online: https://cran. r-project. org/web/packages/dismo/vignettes/brt. pdf (accessed on 12 June 2011)*, 2017.

[370] G. Ridgeway, D. McCaffrey, A. Morral, B. A. Griffin, L. Burgette, M. L. Burgette, M. Ridgeway, and M. Burgette, “Package ‘twang’,” 2015.

[371] P. McCullagh and J. A. Nelder, *Generalized linear models.* Routledge, 2019.

[372] M. Kuss and C. Rasmussen, “Gaussian processes in reinforcement learning,” *Advances in neural information processing systems*, vol. 16, 2003.

[373] I. T. Jolliffe, “A note on the use of principal components in regression,” *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 31, no. 3, pp. 300–303, 1982.

[374] A. Saadallah, H. Mykula, and K. Morik, “Online adaptive multivariate time series forecasting,” in *Joint European conference on machine learning and knowledge discovery in databases*, Springer, 2022.

[375] A. Saadallah, M. Jakobs, and K. Morik, “Explainable online ensemble of deep neural network pruning for time series forecasting,” in *Machine Learning Journal*, Springer International Publishing, 2022.

[376] A. Saadallah, J. Büscher, O. Abdulaaty, T. Panusch, J. Deuse, and K. Morik, “Explainable predictive quality inspection using deep learning in electronics manufacturing,” *Procedia CIRP*, vol. 107, pp. 594–599, 2022.

[377] A. Saadallah, O. Abdulaaty, J. Büscher, T. Panusch, K. Morik, and J. Deuse, “Early quality prediction using deep learning on time series sensor data,” *Procedia CIRP*, vol. 107, pp. 611–616, 2022.

[378] A. Saadallah, F. Finkeldey, J. Buß, K. Morik, P. Wiederkehr, and W. Rhode, “Simulation and sensor data fusion for machine learning application,” *Advanced Engineering Informatics*, vol. 52, p. 101 600, 2022.

[379] A. Saadallah, A. Egorov, B.-T. Cao, S. Freitag, K. Morik, and G. Meschke, “Active learning for accurate settlement prediction using numerical simulations in mechanized tunneling,” *Procedia CIRP*, vol. 81, pp. 1052–1058, 2019.

A Time Series Data Sets and Learning Algorithms

A.1 Data Sets

A.1.1 MTS Data Sets for the Meta-Learning Task in OAMTS

Name	Nr of time series	Nr of variables	Source	Characteristics	Target variable
Stock data	5	82	UCI [1]	Daily Closing Price from 2010 to 2017	Close
NASA Flight data	20	30	NASA [2]	Data collected every second from aviation domain	alt
Air quality data	9	15	UCI [3]	Hourly responses of gas multisensor device	PT08.S1.CO.
Beijing Guangzhou Shanghai	9 17 8	86	UCI [4]	Hourly data of the PM2.5 data	PM_Dongsi PM_City.Station PM_Jingan
Maintenance of Naval Propulsion Plants	12	16	UCI [5]	Data collected from simulator of gas turbines	V55
TLC Trip Record Data (Yellow taxi - 2021-01)	20	16	NYC [6]	New York taxi trip data	total_amount

Table A.1: The summary of datasets used for the meta-learning in OAMTS.

A.1.2 MTS Data Sets for OAMTS Evaluation

Name	Nr of time series	Nr of variables	Source	Characteristics	Target variable
TLC Trip Record Data (Yellow taxi - 2021-07)	20	16	NYC [7]	New York taxi trip data	total_amount
Shenyang Chendgu	19 10	86	UCI [8]	Hourly data of the PM2.5 data	PM_Taiyuanjie PM_Caotangsi
Bike Sharing Dataset	17	16	UCI [9]	Hourly count of rental bikes between 2011 and 2012	cnt

Table A.2: The summary of datasets used for the evaluation of OAMTS.

A.1.3 Time Series Data Sets for DEMSRC, OMS-ROC, OS-PGSM, and OEP-ROC, and OEA-DRL Evaluation

All used datasets, together with a short description, can be found in Table A.3. For the datasets from the UEA & UCR Time Series repositories, we ”converted“ them to time series forecasting datasets by taking the first feature vector X_0 from the respective training set and using that for splitting into training, validation and test parts. We also used a total of 98 datasets from the M4 competition [142]. More precisely, we used columns of the hourly, monthly, weekly and daily tables given by the challenge. The extracted columns were cleaned by skipping the NaN values at the beginning and end of each time series, should they exist.

A.2 Learning Algorithms

We also use several standard regression learning algorithms in all our methods in addition to the forecasting models that we have presented in Chapter 2. These algorithms were applied in an autoregressive fashion using the time series embedding (See Section 2.4.4). The list of learning algorithms is the following:

SVR: Support vector regression with linear, radial basis function, Laplacian, and polynomial kernels. The parameter for cost of constraints violation is set to 1 (default), and the epsilon in the insensitive-loss function is set to 0.1 (default). We used the implementation from the R package kernlab [364];

[1]https://archive.ics.uci.edu/ml/datasets/CNNpred%3A+CNN-based+stock+market+prediction+using+a+diverse+set+of+variables

[2]https://data.world/us-nasa-gov/f70d5ecd-0c18-4033-9568-416c4e14c96c

[3]https://archive.ics.uci.edu/ml/datasets/Air+Quality

[4]https://archive.ics.uci.edu/ml/datasets/PM2.5+Data+of+Five+Chinese+Cities

[5]https://archive.ics.uci.edu/ml/datasets/Condition+Based+Maintenance+of+Naval+Propulsion+Plants

[6]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[7]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[8]https://archive.ics.uci.edu/ml/datasets/PM2.5+Data+of+Five+Chinese+Cities

[9]https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset

Name	Nr. of time series	Source	Characteristics
Amount registered	1	Bike sharing [9]	Hourly, Jan. 1, to Mar. 01, 2011
AbnormalHeartbeat	1	UEA & UCR [362]	3053 measurements (4kHz)
CatsDogs	1		14773 audio samples (16kHz)
Cricket	1		1197 accelerometer readings (184Hz)
EOGHorizontalSignal	1		1250 measurements (1kHz)
EthanolConcentration	1		1 second spectrum measurement
Phoneme	1		1024 samples of audio
Rock	1		2844 samples of spectrum analysis
SNP500	1	UCI [363][111]	Daily closing, 2010 to 2017
DJI	1		
NYSE	1		
RUSSELL	1		
Electricity (Hourly)	11	Monash Forecasting Benchmark [1]	Energy consumption measurements
KDD Cup 2018	13		Forecast air quality indices (AQIs)
Pedestrian Counts	12		Hourly pedestrian counts from Melbourne
Solar (10 minutes)	12		Solar power production
M4 (Daily)	12		Daily time series from M4 dataset
M4 (Weekly)	13		Weekly time series from M4 dataset
Weather	15		Daily weather forecasts

Table A.3: List of datasets used for the experiments of DEMSRC, OMS-ROC, OS-PGSM, OEP-ROC, and OEA-DRL.

MARS: Multivariate adaptive regression splines [266] with different parameters regarding the maximum degree of interaction (Degree), and the maximum number of model terms before pruning (No. terms). The forward stepping threshold is set up to 0.001 (default). We the implementation from the R package earth is used [365].

RF: Random forests [343] with a varying number of trees ranging from 50 to 1000. The number of variables to possibly split at in each node is set to a third of the number of predictor variables [343].

PPR: Projection pursuit regression with different number of terms $(2, 5)$, and two different methods used for smoothing the ridge functions: the Friedman's super smoother [265] or the smoothing spline [366]. We used the implementation from the R package stats [367].

RBR: Rule-based regression based on Quinlan's model tree [183] with a varying number of boosting iterations ranging from 10 to 100. We used the implementation provided in the cubist R package [368];

GBR: Generalized boosted regression [369] with a Gaussian or Laplacian distribution. The number of trees is set in the range of 500 to 1000. The maximum depth of each tree is set to 5 or 10, and the shrinkage parameter applied to each tree in the expansion is set to 0.1 (default). We used the implementation from the R package gbm [370];

GLM: Generalized linear model [371] regression with a Gaussian distribution and a different penalty mixing. When the penalty is set to 1, the algorithm represents LASSO (Least Absolute Shrinkage and Selection Operator) regression, and Ridge regression when it is set to 0. In between 0 and 1, the algorithm is a linear model with elastic net regularisation. These models are implemented in the R package glmnet [133];

GP: Gaussian processes [372] with linear, radial basis function, Laplacian, and polynomial kernels. The tolerance of termination criterion is set to either 0.01 or 0.001. We used the implementation provided in the R package kernlab [364];

PCR: Principal components regression [373] with a default parameter setting provided in the pls R package [267];

PLS: Partial least squares (Geladi and Kowalski, 1986) regression with two different methods: the kernel method, and the SIMPLS method. These are also provided in the pls R package [267];

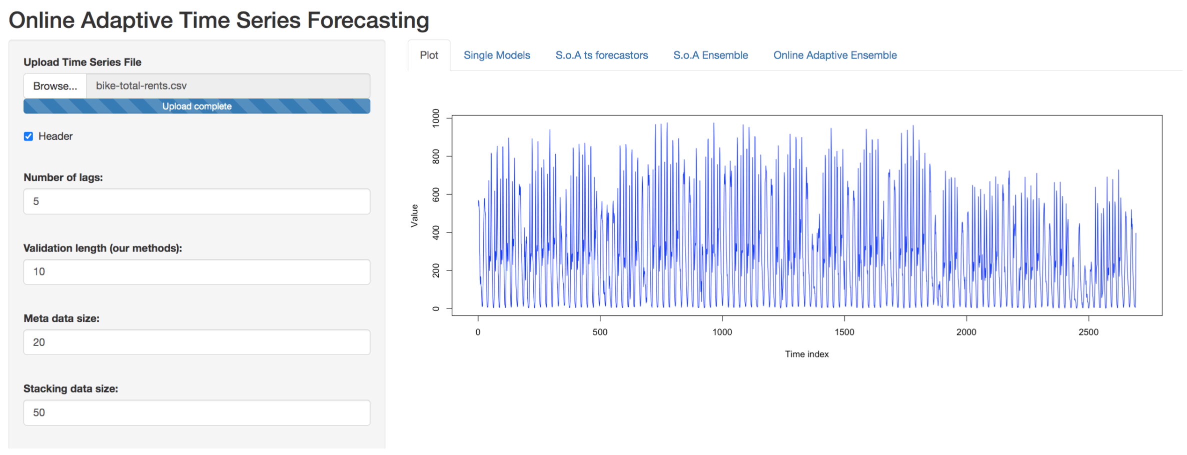
B Online Adaptive Time Series Forecasting-R Shiny Interactive Web App

The developed R-Shiny App is devised to make our methods easier to use and evaluate on new time series datasets. It is also straightforward to use by non-ML domain experts, especially since online time series forecasting is required in a wide range of application domains such as machining production, traffic prediction, and stock market indices forecasting, to name by a few. The App can also be used by ML researchers to reproduce easily our experiments.

Our first plan is to finish the design of the App and publicly host it. In addition, we plan to add an extension to the same App for time series explainability tools including the presented approaches in this thesis.

In this Appendix, we aim at giving an overview of the first version of the App and our research vision for automating time series forecasting in an understandable manner.

Uploading Time Series Data: The data can be stored in ".csv" file.


Online Adaptive Time Series Forecasting

Upload Time Series File

Browse... bike-total-rents.csv

Upload complete

☑ Header

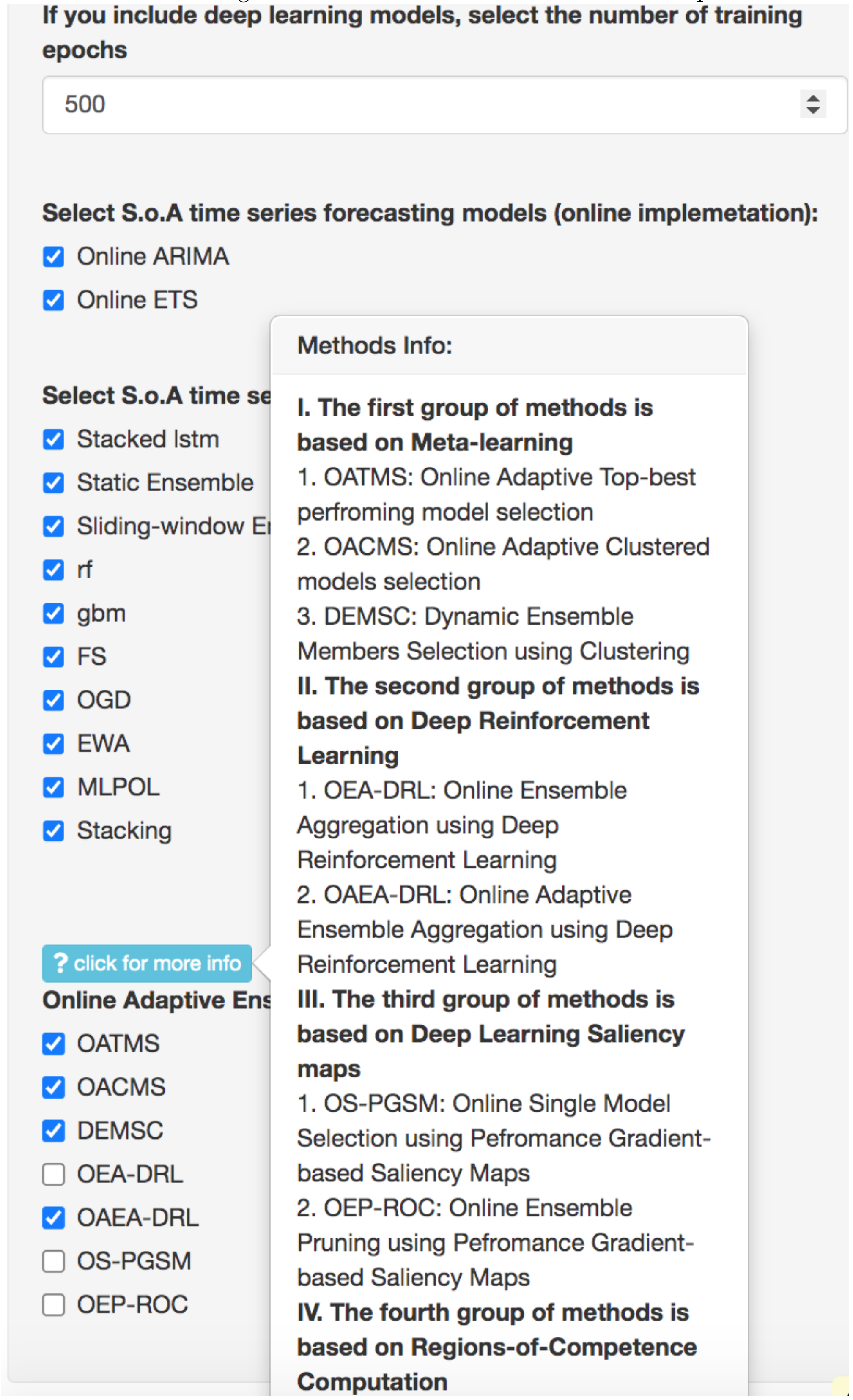
Number of lags: 5

Validation length (our methods): 10

Meta data size: 20

Stacking data size: 50

User Preferences Setting: The user can choose between different hyperparameters and families of ML individual models. The online implementation of ARIMA [61] and ETS [97] is also provided. The implementation of many SoA methods for online ensemble learning for time series forecasting is also included in addition to our developed methods.


If you include deep learning models, select the number of training epochs

500

Select S.o.A time series forecasting models (online implemetation):

- [x] Online ARIMA
- [x] Online ETS

Select S.o.A time se

- [x] Stacked lstm
- [x] Static Ensemble
- [x] Sliding-window E
- [x] rf
- [x] gbm
- [x] FS
- [x] OGD
- [x] EWA
- [x] MLPOL
- [x] Stacking

? click for more info

Online Adaptive Ens

- [x] OATMS
- [x] OACMS
- [x] DEMSC
- [ ] OEA-DRL
- [x] OAEA-DRL
- [ ] OS-PGSM
- [ ] OEP-ROC

Methods Info:

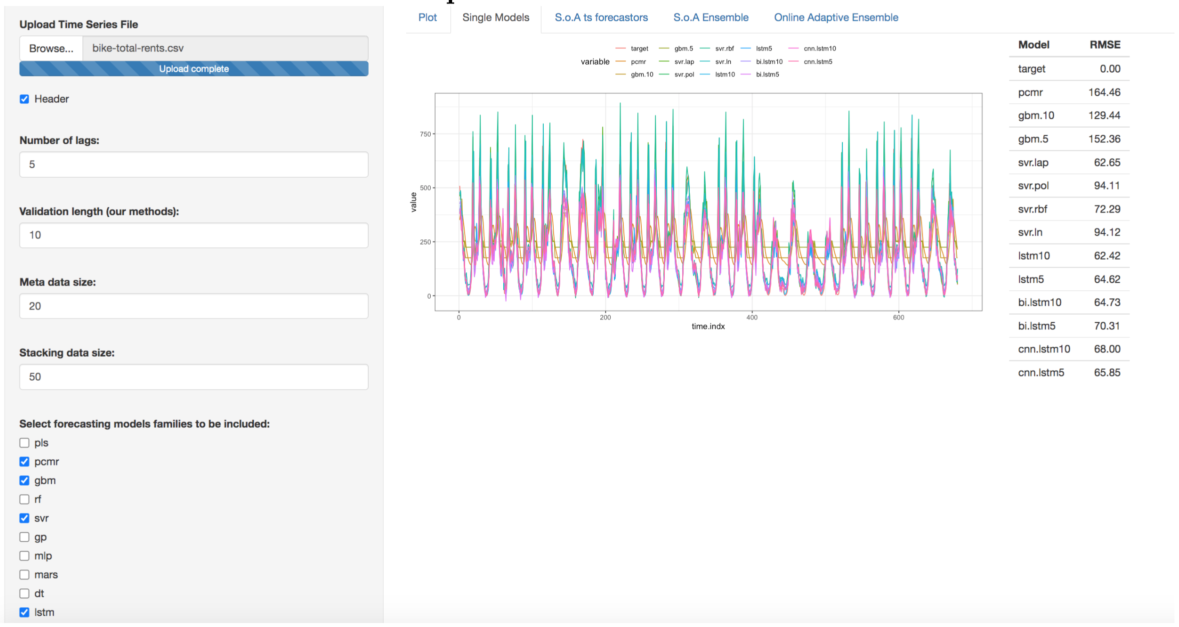
I. The first group of methods is based on Meta-learning
1. OATMS: Online Adaptive Top-best perfroming model selection
2. OACMS: Online Adaptive Clustered models selection
3. DEMSC: Dynamic Ensemble Members Selection using Clustering

II. The second group of methods is based on Deep Reinforcement Learning
1. OEA-DRL: Online Ensemble Aggregation using Deep Reinforcement Learning
2. OAEA-DRL: Online Adaptive Ensemble Aggregation using Deep Reinforcement Learning

III. The third group of methods is based on Deep Learning Saliency maps
1. OS-PGSM: Online Single Model Selection using Pefromance Gradient-based Saliency Maps
2. OEP-ROC: Online Ensemble Pruning using Pefromance Gradient-based Saliency Maps

IV. The fourth group of methods is based on Regions-of-Competence Computation

Evaluation Examples: Individual Models Evaluation


Upload Time Series File

Browse... bike-total-rents.csv

Upload complete

☑ Header

Number of lags: 5

Validation length (our methods): 10

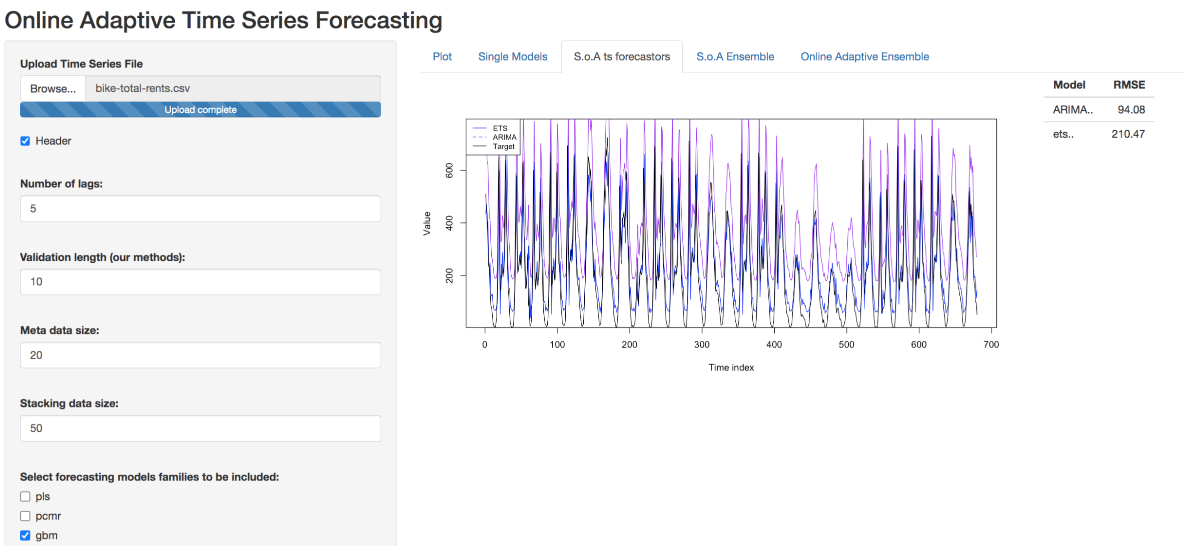
Meta data size: 20

Stacking data size: 50

Select forecasting models families to be included:

- ☐ pls
- ☑ pcmr
- ☑ gbm
- ☐ rf
- ☑ svr
- ☐ gp
- ☐ mlp
- ☐ mars
- ☐ dt
- ☑ lstm

Model	RMSE
target	0.00
pcmr	164.46
gbm.10	129.44
gbm.5	152.36
svr.lap	62.65
svr.pol	94.11
svr.rbf	72.29
svr.ln	94.12
lstm10	62.42
lstm5	64.62
bi.lstm10	64.73
bi.lstm5	70.31
cnn.lstm10	68.00
cnn.lstm5	65.85


Online Adaptive Time Series Forecasting

Upload Time Series File

Browse... bike-total-rents.csv

Upload complete

☑ Header

Number of lags: 5

Validation length (our methods): 10

Meta data size: 20

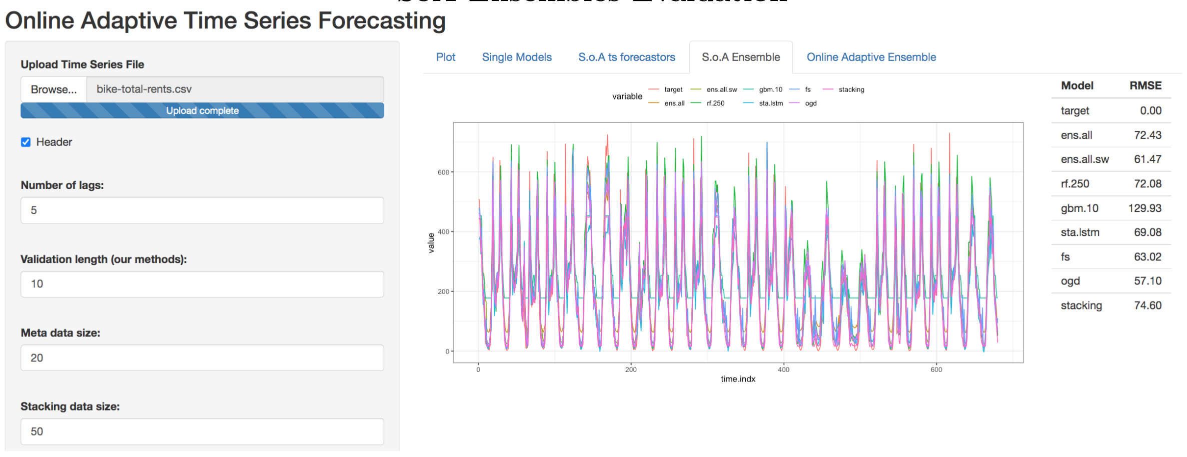
Stacking data size: 50

Select forecasting models families to be included:

- ☐ pls
- ☐ pcmr
- ☑ gbm

Model	RMSE
ARIMA..	94.08
ets..	210.47

SoA Ensembles Evaluation


Online Adaptive Time Series Forecasting

Upload Time Series File

Browse... bike-total-rents.csv

Upload complete

☑ Header

Number of lags: 5

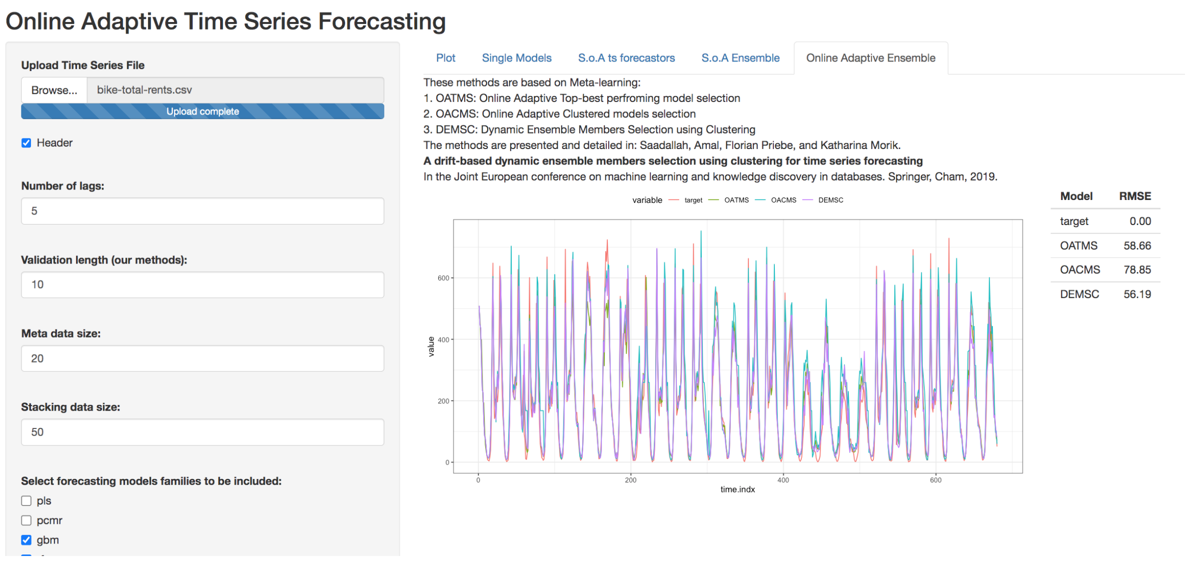
Validation length (our methods): 10

Meta data size: 20

Stacking data size: 50

Model	RMSE
target	0.00
ens.all	72.43
ens.all.sw	61.47
rf.250	72.08
gbm.10	129.93
sta.lstm	69.08
fs	63.02
ogd	57.10
stacking	74.60

Example of DEMSRC and its variants evaluation



Online Adaptive Time Series Forecasting

Upload Time Series File

Browse... bike-total-rents.csv

Upload complete

☑ Header

Number of lags:

5

Validation length (our methods):

10

Meta data size:

20

Stacking data size:

50

Select forecasting models families to be included:

☐ pls

☐ pcmr

☑ gbm

These methods are based on Meta-learning:

1. OATMS: Online Adaptive Top-best perfroming model selection
2. OACMS: Online Adaptive Clustered models selection
3. DEMSC: Dynamic Ensemble Members Selection using Clustering

The methods are presented and detailed in: Saadallah, Amal, Florian Priebe, and Katharina Morik.
A drift-based dynamic ensemble members selection using clustering for time series forecasting
In the Joint European conference on machine learning and knowledge discovery in databases. Springer, Cham, 2019.

Model	RMSE
target	0.00
OATMS	58.66
OACMS	78.85
DEMSC	56.19

C Publications, Joint Work and Collaborations

This chapter gives an overview of publications, and how far material from them has been included in this thesis. It further states the collaborations and contributions of other authors to this work. Whenever "the author" appears, the author of this thesis is meant.

Parts of this thesis have been published in the following articles:

- **A. Saadallah, H. Mykula, and K. Morik, "Online adaptive multivariate time series forecasting," in *Joint European conference on machine learning and knowledge discovery in databases*, Springer, 2022**: Almost all parts of this publication are included in Chapter 3. All the parts have been written by the author. The idea of the online drift-aware time series variables selection is developed by the author to control spatial dependencies in MTS data. The idea of forecasting error monitoring is suggested by Katharina Morik to control temporal dependencies in MTS data. The choice of evaluating multiple dependencies measures as well as several MTS forecasting models and different drift detection methods is made by the author. The idea of introducing meta-learning to automate all the choices is made by the author. The choice and the design of MTS meta-features including the introduction of the landmarking concept for MTS data is made by the author. The organization of all the parts in one algorithm is made by the author. The experiments are conducted by our student assistant Hanna Mykula.

- **A. Saadallah, M. Jakobs, and K. 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*, N. Oliver, F. Pérez-Cruz, S. Kramer, *et al.*, Eds., Cham: Springer International Publishing, 2021, pp. 404–420, ISBN: 978-3-030-86486-6**: Almost all parts of this publication are included in Chapter 5-Section 5.3 and have been slightly adapted for presentation in this thesis. All these parts are written by the author. The idea and the algorithm of the Performance Gradient-based Saliency Maps PGSMs to compute the RoCs of candidate DNNs are developed by the author. The introduction of the drift-detection mechanism to update the RoCs is suggested by the author. The implementation is done by Matthias Jacobs. Comparisons with the recent SoA methods have been conducted by the author.

- **A. Saadallah, M. Jakobs, and K. Morik, "Explainable online ensemble of deep neural network pruning for time series forecasting," in *Machine Learning Journal*, Springer International Publishing, 2022**: Many parts of this publication are included in Chapter 5-Section 5.4. These parts are written by the author. All the rest of the work including the development of the method for online ensemble pruning, the implementation, and the comparisons are carried out by the author and Matthias Jacobs. The theoretical bound for the pruned ensemble size is derived by the author.
- **A. Saadallah, F. Priebe, and K. Morik, "A drift-based dynamic ensemble members selection using clustering for time series forecasting," in *Joint European conference on machine learning and knowledge discovery in databases*, Springer, 2019**: Almost all parts of this publication are included in Chapter 4-Section 4.3 and have been slightly adapted for presentation in this thesis. All these parts are written by the author. The main idea of the drift-aware ensemble pruning, its implementation, the experiments, and the comparisons are done by the author. Florian Priebe helped with preparing the presentation of the results.
- **A. Saadallah, M. Tavakol, and K. Morik, "An actor-critic ensemble aggregation model for time-series forecasting," in *IEEE International Conference on Data Engineering (ICDE)*, 2021**: Some parts of this publication are included in in Chapter 6. The idea of using an actor-critic Reinforcement Learning approach is suggested by Maryam Tavakol. The design of the RL environment to fit the forecasting task is made by the author. Maryam Tavakol and the author collaborated together on the writing. The implementation and evaluation of the method are made by the author.
- **A. Saadallah and K. Morik, "Online ensemble aggregation using deep reinforcement learning for time series forecasting," in *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2021**: Some parts of this publication are included in in Chapter 6. All the parts are written by the author. The idea of the online update of the RL policy, its implementation, and evaluation is achieved by the author.
- **A. Saadallah, J. Büscher, O. Abdulaaty, *et al.*, "Explainable predictive quality inspection using deep learning in electronics manufacturing," *Procedia CIRP*, vol. 107, pp. 594–599, 2022**: Almost all parts of this publication are included in Chapter 8-Section 8.4. The methodology part is written by the author. The idea of opting for different reshaping of the data to enhance explainability aspects is suggested by the author. The data and its description have been provided by Jan Büscher. The data preprocessing is conducted by our student assistant Omar Abdulaaty.

- **A. Saadallah, O. Abdulaaty, J. Büscher, *et al.*, "Early quality prediction using deep learning on time series sensor data," *Procedia CIRP*, vol. 107, pp. 611–616, 2022** Almost all parts of this publication are included in Chapter 8-Section 8.3. All these parts are written by the author. The algorithm for early quality prediction using time series data is developed by the author. The data for the case study and its description have been provided by Jan Büscher. The experiments are conducted by Omar Abdulaaty. Their description and discussion are done by the author.

- **A. Saadallah, F. Finkeldey, J. Buß, *et al.*, "Simulation and sensor data fusion for machine learning application," *Advanced Engineering Informatics*, vol. 52, p. 101 600, 2022**: Many parts of this publication are included in Chapter 7. All these parts are written by the author except for the use case description and the description of the application of synchronization and calibration to this use case are written by Felix Finkeldey. The general methodology behind simulation synchronization and calibration using ML techniques is detailed and written by the author. The automated data fusion framework is designed and implemented by the author. The synchronization and calibration parts for the presented use case are implemented by Felix Finkeldey who also run the simulation and conducted the real-machining experiments. The use case plots are created by Felix Finkeldey. The ML experiments including the automated fusion level selection and online cutting forces prediction are conducted by the author.

In collaboration with other researchers on related topics to this thesis, the following additional articles have been published:

- **F. Finkeldey, A. Saadallah, P. Wiederkehr, *et al.*, "Real-time prediction of process forces in milling operations using synchronized data fusion of simulation and sensor data," *Engineering Applications of Artificial Intelligence*, vol. 94, p. 103 753, 2020:** The Machine Learning part including development, description, implementation and evaluation is done by the author. The use case data acquisition and preparation is carried out by Felix Finkeldey.

- **A. Saadallah, F. Finkeldey, K. Morik, *et al.*, "Stability prediction in milling processes using a simulation-based machine learning approach," *Procedia CIRP*, vol. 72, pp. 1493–1498, 2018:** The Machine Learning part including development, description, implementation, and evaluation is done by the author. The idea of the Active Learning method is proposed and implemented by the author. The use case data acquisition and preparation and the presentation of the results are carried out by Felix Finkeldey.

- **A. Saadallah, L. Moreira-Matias, R. Sousa, *et al.*, "Bright-drift-aware demand predictions for taxi networks," *IEEE Transactions on Knowledge and Data Engineering*, 2018:** The idea and the design of the framework are done by the author and Luis Moreira-Matias. The implementation and evaluation of the drift-aware VAR model as well as the ensemble pruning and aggregation are performed by the author. GPs Data pre-processing for Stockholm and Shanghai data sets are performed by the author. The presentation of the results is done by the author.

- **A. Saadallah, N. Piatkowski, F. Finkeldey, *et al.*, "Learning ensembles in the presence of imbalanced classes.," in *ICPRAM*, 2019, pp. 866–873:** The idea and the design of the framework as well as its implementation and evaluation are done by the author. The derivation of a bound for the sampling rate is proposed by Nico Piatkowski.
- **A. Saadallah, A. Egorov, B.-T. Cao, *et al.*, "Active learning for accurate settlement prediction using numerical simulations in mechanized tunneling," *Procedia CIRP*, vol. 81, pp. 1052–1058, 2019:** The idea and the design and the implementation, as well as the evaluation of the Active Learning Procedure, are done by the author. The writing is also done by the author. Ba Trung Cao contributed with the use case data and its description.
- **M. Bunse, A. Saadallah, and K. Morik, "Towards active simulation data mining," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) Workshops*, 2019, p. 104:** The work is equally split between the author and Mirko Bunse.
- **A. Saadallah and K. Morik, "Active sampling for learning interpretable surrogate machine learning models," in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2020, pp. 264–272:** The design, implementation, and evaluation of the framework are carried out by the author.