# Time series analysis and prediction of electricity load demand with machine learning techniques


DISSERTATION ZUR ERLANGUNG DES DOKTORGRADES DER NATURWISSENSCHAFTEN (DR. RER. NAT.) DER FAKULTÄT FÜR PHYSIK

DER UNIVERSITÄT REGENSBURG

vorgelegt von

Florian Steinborn

aus

Tegernsee

im Jahr 2021

Promotionsgesuch eingereicht am: 27.10.2021
Die Arbeit wurde angeleitet von: Prof. Dr. Elmar W. Lang

Prüfungsausschuss:

| | |
|---|---|
| Vorsitzender: | Prof. Dr. Christian Schüller |
| Erstgutachter: | Prof. Dr. Elmar W. Lang |
| Zweitgutachter: | ARat Dr. Stefan Solbrig |
| weiterer Prüfer: | Prof. Dr. Jascha Repp |

Datum Promotionskolloquium: 22.12.2021

# Abstract

In the context of climate change, a transition towards more renewable energy sources and more local use of energy is inevitable. To be able to plan machine schedules for local energy systems, accurate forecasts of electricity load demands are needed. However, shortly after smart meters have first been installed, there is only little past data available. Nevertheless, a precise forecast is required. The research in this thesis aims to answer the question of how to achieve high forecast accuracy in Short Term Load Forecasting (STLF) settings when not much training data is available. The focus is thus laid on more simple models, leveraging the internal structure of load demand time series data.

For different model types, it is analyzed how the amount of available training data affects the model performance. Furthermore, a novel NMF-based forecasting model is presented. A time series of the aggregated electricity load demand of 350 households, measured in 30-minute intervals, is used for all experiments.

The done research shows that, for the forecasting accuracy of many models, the amount of training data plays a critical role. It is shown that simple models, leveraging the idea of slicing the data to daily pieces, give satisfying forecasting accuracy, even when only little training data is available. The most successful models were simple regression models, applying a Random Forest or a Ridge Regression on features describing the load values of past days. Including information about working days, in contrast to weekends and holidays, can help to further improve the forecast accuracy. The novel NMF-based models gave promising results but are not suited for a situation with only little training data available. Overall, more elaborated model designs needed more data to achieve low forecast errors.

It is shown that for a practical application, in a setting where not much data has been collected yet, simple forecasting models, strongly relying on the daily periodicity in the data, are superior to more elaborated models like Neural Networks. The results suggest that daily slicing of time series is a simple, yet effective, preprocessing step for STLF tasks, and that one should not simply use large Neural Network models just because it is today's flavor of doing so.

# Kurzfassung

Vor dem Hintergrund des Klimawandels ist ein Übergang zu mehr erneuerbaren Energiequellen und einer stärkeren lokalen Energienutzung unumgänglich. Um Kraftwerksfahrpläne für lokale Energiesysteme planen zu können, sind genaue Prognosen des Strombedarfs erforderlich. Kurze Zeit nach der Installation von Smart Metern stehen jedoch nur wenige Verbrauchsdaten zu vergangenen Zeiten zur Verfügung. Dennoch bedarf es einer genauen Vorhersage.

In dieser Arbeit wird der Frage nachgegangen, wie eine hohe Vorhersagegenauigkeit erreicht werden kann, wenn nur wenige Trainingsdaten verfügbar sind. Der Schwerpunkt liegt daher auf einfacheren Modellen, welche die interne Struktur der vorliegenden Zeitreihendaten nutzen.

In diesem Kontext wird für verschiedene Modelltypen analysiert, wie sich die Menge der verfügbaren Trainingsdaten auf die Vorhersagegenauigkeit auswirkt. Zudem wird ein neuartiges NMF-basiertes Vorhersagemodell vorgestellt. Für alle Experimente wurde eine Zeitreihe des aggregierten Strombedarfs von 350 Haushalten verwendet, die in 30-Minuten-Intervallen gemessen wurde.

Die durchgeführten Forschungsarbeiten zeigen, dass für die Vorhersagegenauigkeit vieler Modelle die Trainingsdatenmenge eine entscheidende Rolle spielt. Außerdem wird gezeigt, dass einfache Modelle, die eine Aufteilung der Daten in tägliche Abschnitte nutzen, eine zufriedenstellende Vorhersagegenauigkeit bieten, selbst wenn nur wenige Trainingsdaten verfügbar sind.

Am erfolgreichsten waren einfache Regressionsmodelle, bei denen ein Random Forest oder eine Ridge Regression auf Features angewendet wurde, die die Lastwerte der vergangenen Tage beschreiben. Wurden Informationen über Arbeitstage beziehungsweise Wochenenden und Feiertage mit einbezogen, konnte die Vorhersagegenauigkeit weiter verbessert werden. Die neuartigen NMF-basierten Modelle lieferten vielversprechende Ergebnisse, sind jedoch in einer Situation mit nur wenigen verfügbaren Trainingsdaten nicht geeignet. Insgesamt benötigten aufwändigere Modelldesigns mehr Daten, um niedrige Vorhersagefehler zu erreichen.

Es wird gezeigt, dass für eine praktische Anwendung in einem Umfeld, in dem noch nicht viele Daten gesammelt wurden, einfache Vorhersagemodelle, die sich vor allem auf die tägliche Periodizität der Daten stützen, ausgefeilten Modellen wie Neuronalen Netzen überlegen sind. Die Ergebnisse legen nahe, dass das Zerlegen von Zeitreihen in tägliche Abschnitte ein einfacher, aber effektiver Vorverarbeitungsschritt ist und dass große Neuronale Netze nicht einfach verwendet werden sollten, nur weil dies heutzutage eine gängige Methode ist.

# Contents

# 1. Introduction

> "Begin at the beginning," the King said gravely, "and go on till you come to the end: then stop."
>
> Lewis Carroll, Alice in Wonderland

## 1.1. Impact of climate change

Climate change is advancing and poses a challenging task. As confirmed by the Intergovernmental Panel on Climate Change (IPCC), extreme weather events occur more often due to climate change. At a global warming of 1.5 °C compared to the level before industrialization, in addition to influences on the weather, negative consequences for entire ecosystems are predicted [1]. Scientists even warn that without urgent action, global warming is likely to exceed 2 °C by 2060 and could increase to 5 °C until 2100 [2]. This would bring irreversible changes to the world as we know it, lead to a loss in biodiversity and also pose problems for food production [2].

Climate change is a global problem concerning us all and making it urgently necessary to significantly reduce the emission of greenhouse gases such as carbon dioxide (CO2) and methane (CH4) as a global response. Many measures need to be taken on many fronts, including the reduction of greenhouse gases emitted in the energy sector.

According to the German Federal Environment Agency (Umweltbundesamt), energy-related emissions account for around 85% of German greenhouse gas emissions. About half of these are emitted in power and heat generation [3]. Energy-related emissions are the emissions of greenhouse gases and air pollutants released by the conversion of energy sources into electrical and/or thermal energy. This includes emissions in industrial processes and power plants, but also the emissions caused in households (mainly by heating with fossil fuels) and emissions in the transport sector [4].

In the time from 1990 to 2016, the total electricity consumption in Germany increased by around 21.5 % to 582 billion kWh per year, while in the same period CO2 emissions from power generation could be reduced from 366 to 300 million tons per year [5]. Thus, relative to the electricity used, the emissions already decreased, but in 2016 in Germany, approximately 67.5% of the required power was still generated from fossil fuels and nuclear energy. Furthermore, the environmentally harmful combustion of lignite made up the largest share of all electricity generation variants with 133 billion kWh [5].

From 1990 to 2017 the amount of CO2 equivalents in Germany decreased, but in 2017 the energy industry still accounts for the largest share of energy-related emissions with 313.4 million metric tons of a total 766 million metric tons CO2 equivalents of energy-related emissions. This amount well exceeds the emissions caused by the transport sector (168 million metric tons) [4].

In light of the high greenhouse gas emissions and the Paris Agreement, the European Union reacted by agreeing on a climate and energy framework, committing to cutting its greenhouse gas emissions by at least 40% by 2030, compared to 1990. In December 2020, the EU realized that it needs to increase its ambitions and the European Council endorsed a binding target for emission reduction by 2030 to at least 55% compared to 1990, increasing it from the 40% reduction target [2, 6].

It is clear that a path towards lower greenhouse gas emissions must be followed, and that the transition towards a sustainable energy supply based on renewable energy sources can largely contribute to the overall goal of stopping climate change. But according to the IPCC report in 2011, saving energy alone will not be enough to compensate for the additional burdens resulting from the increase in population and prosperity [7]. Instead, we need to drastically rethink the energy supply of modern societies.

In a tech report by the Umweltbundesamt it is claimed that, for Germany, an electricity supply system based entirely on renewable energies by 2050 is not only ecologically, but also technically feasible [8]. This could enable us to reduce the greenhouse gas emissions from electricity production to virtually zero. But to achieve such a tremendous reduction in greenhouse gas emissions, a transformation of the electricity supply systems is inevitable. To be able to implement a power system based solely on renewable energies by 2050, a lot still must happen. Storage capacities must be expanded, electricity transport infrastructure must be built and the concept of load management must be rolled out [8].

## 1.2. Decentralized supply of energy and its importance

One way of building towards a more sustainable energy system is to decentralize energy supply. In the course of the energy transition, an increasing number of small generating units for renewable energies are being integrated into the power grid. These include photovoltaic, wind power and geothermal systems as well as biomass power plants and combined heat and power plants. In a study on how to integrate renewable energies into the German electricity supply from 2010, the Deutsche Energie-Agentur GmbH calls for an adjustment of the network structure. An increased share of renewable electricity generation will necessitate an adaptation of the required grid infrastructure. The main call is for a significant and fast grid expansion, as an increasing discrepancy between the expansion of renewable energy technologies and the necessary grid infrastructure was detected [9]. However, a grid expansion alone will not be sufficient. With volatile energy sources, additional requirements such as balancing load fluctuations also have to be managed. Thus, the necessary optimization of the energy grid must go hand in

hand with a more flexible power system overall. Smart meters and load management at the transmission and distribution level are required, in particular with regard to a grid-relieving effect of the optimization [9].

Grid extension is costly and difficult to perform, but grid-relieving effects can and should also be obtained by using the energy supplied by the generating units in a more local, decentralized way. However, the increasingly distributed feeding into the power grid leads to difficulties in compliance with permissible voltage values. Whereas large traditional nuclear or coal power plants and large wind parks can deliver electricity with high voltages, most smaller or medium-size units cannot output high-voltage electricity. This is an issue, as transmitting electric energy with lower voltages leads to higher energy losses during the transmission process. As transformators also have to deal with losses, it is not worthwhile to transmit the electric energy of smaller units over longer distances.

Moreover, the large number of producers raises the problem of deployment planning to dimensions that are difficult to control [10, 11, 12]. Additionally, the strong dependency on the weather and associated fluctuations in power generation by solar and wind power plants make it difficult to plan the power provided by such units. But a stable supply network must also be guaranteed when the power provided by central power plants is reduced. To make this possible, a step-by-step transformation away from central market and network structures towards a decentralized structure is inevitable [13]. This way a local supply can be targeted, helping to reduce the long transmission losses in the network and, thus, using the energy provided by smaller units more effectively.

For such a transformation to work, it is necessary to bundle the generation potential of small systems through information technology [14]. A collection of units with smaller potentials can then be viewed as a single unit with a correspondingly higher performance potential, also called virtual power plant (VPP) [15]. In course of the energy transition, VPPs will gradually have to take over the tasks of larger power plants, as they can bundle the initially low potential and control flexibility of individual plants in a targeted manner [16, 17].

The International Energy Agency (IEA) describes a possible transition to a decentralized solution of the electricity grid in three steps [18]:

1. Accommodation: The distributed generation is accommodated into the current market via price signals, while the centralized control of the networks remains in place.

2. Decentralization: The share of distributed generation increases and virtual utilities optimize the services of decentralized providers through the use of common communication systems. However, monitoring and control by local utilities are still required.

3. Dispersal: Distributed power takes over the electricity market. Microgrids and power parks effectively meet their supply with limited recourse to grid-based

electricity. The Distribution operates more like a coordinating agent between separate systems rather than a controller of the system.

Furthermore, to promote decentralization, the electricity sector should not be viewed in isolation. Instead, it is important to pursue a holistic approach and exploit the possibilities of energy storage units and sector coupling. Sector coupling refers to the networking of individual sectors of the energy industry (electricity, heat, gas) and the mobility sector. The term comprises several coupling mechanisms, which are also referred to as "Power-to-X" (Power-to-Chemicals, Power-to-Gas, Power-to-Heat, Power-to-Liquids, and Power-to-Mobility) and combined heat and power plants. The core and connecting element of the sector coupling is the electricity sector [19]. Using all options from sector coupling to demand side management and smart metering makes a decentralized supply of energy possible and can lead to a more grid-relieving use of energy.

## 1.3. Energy control systems and schedule optimization

### 1.3.1. Intelligent schedule optimization

In conventional operation planning, all measured values, properties and parameters of the producers and consumers usually come together at one central collection point. There, the overall system is evaluated and operating schedules are assigned to the individual system components with the main task of balancing energy supply and consumption at all times.

Procedures that collect all planning-relevant data in one place and try to carry out a global optimization are, however, no longer applicable due to the enormous increase in the number of individual units and the many associated constraints in the course of energy transition [20]. The schedule assignment is generally extremely complex and, with conventional linear programming, it can only under certain circumstances be solved with reasonable computing time [21]. In systems with a large number of units, it is entirely infeasible for them all to operate in one centralized structure, as the optimization problem is NP-hard in the number of participants [22]. In addition, the approach of a central control center is difficult to reconcile with the dynamics of the electricity market and the needs of the individual plant operators. The switch to decentralized controls and the use of VPPs is, therefore, a logical step.

The task of balancing energy supply and consumption can be seen as a resource allocation problem, with a number of system components having to provide a certain amount of resources to satisfy a specific demand. A modern way to tackle these types of problems is to solve them locally. This can be done by a multi-agent system, where multiple agents only have limited resources and knowledge and rely on cooperation through communication [22].

Distributed combinatorial optimization problems are characterized by the fact that the decision variables of a problem are controlled by distributed, autonomous processes. Due to dependencies between the individual variable assignments, the decisions of the processes must be coordinated with each other in a goal-oriented way [14]. In our situation of schedule-based planning of decentralized energy power plants, an operating schedule has to be assigned to every single power plant. In a fully distributed coordination paradigm, each unit searches in its own search space, limited by local constraints for a partial solution that meets the global optimization goal (the power demand), taking into account all other plants involved. The resulting operational plan of all schedules thus represents a consensus among all units jointly matching the global target profile as closely as possible [14].

This process of finding the optimal resource allocation is highly complex. For example, to optimally include sector coupling, the optimization has to be able to include possibilities and restrictions of combined heat and power plant (CHP) systems, Power-to-X conversions and storage units in a VPP in a network-friendly way. For this, schedules must be assigned to the individual subsystems. These define which generator should provide which output at what point in time to be able to cover the required demand with the help of the entire VPP.

The optimization has to take into account several parameters: The conditions and restrictions of each individual subsystem (e.g. available power ranges, coupled heat generation in CHP systems or storage levels to be complied with), the network structure of the supply network, prices on the electricity exchange and the expected demand to be covered represent constraints of the optimization to be fulfilled [23]. In addition, the schedules created should be economical as well as useful to the network. It is therefore a multi-criteria optimization for which a solution with regard to several goals, a so-called optimal Pareto front, has to be found.

For the planning of schedule combinations of the units in a VPP, either the number of possible schedules must be directly available for each individual system or a mathematical modeling with which the feasibility of a timetable can be determined has to be provided. Due to the large number of individual systems, the search space of possible schedules is large. Usually, each separate subsystem can vary its electricity generation within its given limits and, thus, offer different schedules as alternatives for deployment planning. However, technical restrictions as well as economic considerations prevent the implementation of a large part of the theoretically conceivable schedules [24].

The conventional approach to schedule optimization consists of describing the conditions and constraints to be met by linear equations and inequalities. In the solution space described by this, a concrete schedule can then be determined with the help of linear programming. Due to the large number of individual systems and the large number of secondary conditions, the search space is complex. But a transformation of central market and network structures towards a decentralized structure is urgently needed for

the successful integration of renewable energy generation [10]. In order to tackle the optimization problem connected with this decentralized structure, innovative optimization and control techniques are required. There are various methods and heuristics for these and similar optimization problems. What all have in common is a balance between exploration and exploitation of the search space. The task of coordinating the single units by assigning them schedules and even calculating the possible schedule flexibilities depending on technical boundary conditions and current operating circumstances [24] is an active field of research, though not the topic of this thesis but merely outlines the context. For all schedules, a forecast of the demand that must be provided by the conglomerate of generators is the basis. And this load forecasting task is what will be contributed to with this thesis.

### 1.3.2. Flexibilities in schedules

If a schedule for each subsystem of the VPP was found by optimization, there is an area in the search space of a unit in which its schedule can be changed, but the boundary conditions of the unit are still met. These freedoms to change schedules are also known as flexibilities. They can be used to adapt the schedules, if necessary. This allows the system to react to possibly occurring changes in the target load profile, e.g. when unexpectedly high peak demands occur and high network loads must be avoided.

For example, in case an unforeseen power shortage occurs in the network, the schedule of a combined heat and power plant can be adjusted such that it feeds power into the grid at the time of the bottleneck and temporarily stores the generated heat in a buffer until it is needed. Therefore, by using the flexibilities in their schedules, small systems can largely contribute to help stabilize the power grid.

To ensure the balance between power generation and demand, a certain amount of control power is permanently kept available to compensate for forecast deviations, load fluctuations or power plant outages [25]. Nevertheless, the demand for control power is increasing due to the rise of volatile generation. In particular, the need for negative control power to compensate for power surpluses has increased significantly in recent years. Therefore, forecasting the expected feed-in and demand as accurately as possible is an important challenge [25].

In contrast to the technical boundary conditions of a system, the actual demanded power is only known at the time of usage. A good forecast of the expected demand is therefore crucial. The better the forecast reflects the actual demand, the more precisely the search space of the schedules can be defined and the better flexibilities can be calculated and offered. In return, this offers larger potential for regularization measures to stabilize the power grid. A precise forecast of the energetic load that is to be provided is, hence, an essential prerequisite for the effective planning of systems and therefore, an important issue in terms of pushing the energy transition forward.

# 2. Motivation and goal

> I solemnly swear that I am up to no good.
>
> J.K. Rowling, Harry Potter and the Prisoner of Azkaban.

Increasing the share of renewable energy in the energy distribution systems is one of the key goals to prevent climate change from doing even more damage. Where fossil fuels emit a lot of greenhouse gases, furthering climate change, renewables provide green and clean energy by using energy sources like solar, wind or geothermal activities. However, although these energy sources are practically inexhaustible, the use of renewable energy sources comes with a huge problem: While traditional power relying on fossil fuels can be controlled proactively, renewables often rely on weather conditions that are hard to predict. Thus, the amount of provided energy fluctuates and makes it challenging to schedule. This issue leads to large problems in energy systems, as appropriate planning of the amounts of energy that must be provided is one of the main tasks of energy utility operators. They must ensure that the demand for energy is covered by generation at all times.

With an increasing share of renewable energy sources in the energy network, forecasting the assumed generation and demand is of high importance and can help to reduce the uncertainties connected with the application of energy sources like wind or solar. However, forecasting energy loads remains a challenging task.

Forecasts can be given for multiple time ranges, starting from very short time ranges, such as a few minutes, to long time load forecasting with horizons of multiple years. All of these have different applications. This thesis provides a contribution to the area of Short Term Load Forecasting (STLF), with horizons of a few hours to a few days, mainly helping to schedule small and medium-sized energy feeders on the low-voltage to mid-voltage level.

## 2.1. The need for accurate load forecasting

Humankind has always had the drive to predict the future and tell what will or will not happen. In the early days, most of these predictions were done in a religious and superstitious context [26]. But with the rise of science in multiple cultures, evermore researchers worked on finding and analyzing patterns and started to collect data. By looking at these patterns and extrapolating them, one could start to give more reliable

explanations of what is likely to happen. Yet despite more complex methods evolving, the predictions given by forecasting models are still a guess of what is likely to happen and are riddled with uncertainty.

As Makridakis puts it in his work about the forecasting state of the art, there is absolute certainty about two things. First, that no one has prophetic powers and, second, that all predictions are uncertain, with the only thing varying among such predictions being the extent of such uncertainty [27].

One of many application cases of forecasting are forecasts for electric load demands. In order to integrate producers into the electricity market, a half-hourly load forecast must be made. In Germany, schedules of larger power units must be reported to respective network operators at 12 o'clock noon the previous day.Reporting the schedules ahead of time is important for the energy distribution taking place in the so-called Day-Ahead Spot Market Trading, operated by the European Power Exchange (EPEX) in Paris. The EPEX provides a market organized as an auction, where the bids of buyers and sellers have to be placed no later than 12 noon on the day before [28]. Therefore, to be able to participate in this trading and to market the energy that can be provided, it is important for utilities to know how much they can trade with. Thus, they need reliable forecasts with high accuracy, giving them a basis for their trades. However, in the field of time series forecasting, energy load measures with measurements in periodicity of less than an hour, as will be used in this thesis, are only considered by a fraction of the published work [29].

Forecasting does not only concern the question of how much energy can be provided and marketed. Generally, forecasting electric loads is one of the most important tools allowing system operators to plan properly, especially with the increasing use of renewable energy sources.

In practice, Load Forecasting can be used for multiple reasons: Hong brings up five areas where utilities can and do make use of Load Forecasting across different departments of system operators [30]. Firstly, the already mentioned aspect of energy purchasing is a huge field for which short term load forecasts are especially important. In transmission and distribution planning, forecasts are needed to be able to estimate how much the future load demand and the number of customers will grow. Based on that, decisions about upgrading and maintaining the system can be made. In daily operations, forecasts are also important in order to estimate at what times maintenance outages of systems can be practical. Hong also mentions the field of Demand Side Management (DSM), where Load Forecasting strongly supports the decisions of load control. Lastly mentioned is the financial planning of utilities, with load forecasts helping to estimate future revenues and, thus, helping to make business decisions.

The importance of forecasting has grown even more with the increasing use of the aforementioned renewable energy sources. Whereas large energy distribution networks previously mainly worked in a top to bottom process, consumers today, have started to

generate electricity for their own use, e.g. by using solar panels, and interact with the energy distribution networks, making the task of energy distribution even more complex and dynamic. On the one hand, this affects the dynamics of providing load, on the other hand, e.g. the increasing usage of electric vehicles also changes the kind of loads on the demand side that must be provided.

To have a chance to approach these challenges, reliable load forecasts are crucial. But before being able to give good forecasts, first, a proper data situation has to be ensured. Therefore, a monitoring system also providing data at local levels of the energy grid is needed. This is why smart meters are of great importance for the future of our energy distribution system. With smart meter data available, a data-driven forecasting approach can be taken. It is possible to analyze past data together with external factors even on a local level and in short time intervals. Having calculated the predicted load demand, system operators can estimate available flexibilities in the system, helping to keep load demand and provision on par.

In the context of the importance of Short Term Load Forecasting on a more local level, this thesis aims to forecast the time series of the aggregated power demand of 350 households with a granularity of 48 values per day for the time horizon of 2 days.

## 2.2. Importance of smart meters

Collecting local-level data with smart meters in high-resolution frequency brings many opportunities and is a basic requirement for installing a smart grid infrastructure, as it gives the opportunity to measure power demands at substations feeders and even on household levels. This is essential for the decentralization of energy distribution.

A local usage of the network is required to make full use of the available power sources. The flows in the resulting smart electrical grid will vary in magnitude and direction continuously. Thus, the smart grid has to use the options of Demand Side Management (DSM) and VPPs to balance supply and demand [10]. Ramchurn et al. name the prediction of demand and supply at various points in the network as one of the main challenges for installing a smart grid, given the variety of demand control mechanisms and energy sources, each with different degrees of uncertainty of their capability [10]. They point out that for an automated decentralization, coordination between entities and the individual properties of all actors must be involved.

The German energy agency (Deutsche Energie-Agentur GmbH) in 2010 stated in their study about the energy network that the flexibilization of the demand side should be motivated through appropriate tariff systems. Additionally, the electric grids should be adapted at the transmission and distribution level in combination with the creation of incentive systems that are close to the market for the construction and use energy storage systems, especially with regard to a grid-relieving effect [9].

This can only be done if sufficient data is available, making smart meter data the raw material on which the decentralized energy supply and the smart grid should be

built upon. For a smart grid to be effective, it is important to collect data at multiple aggregation levels, i.e. at the level of individual households, feeders, substations, etc. Depending on the aggregation level, the observations will differ strongly. Even among households, the profiles will depend on factors such as the existence of solar panels, overnight storage heating or electric vehicle chargers [31]. Thus, even at the household levels, smart meter data is crucial to provide reliable load forecasts.

## 2.3. Forecasting with little training data

A prerequisite for the implementation of intelligent energy supply is that all participating devices and system components are connected. Nevertheless, digitization in the power sector is still not as advanced as it could be. This is partly due to political hurdles and partly due to the relatively low acceptance on the customer side [32]. To fully use the possibilities of decentralized energy distribution, a measuring, control and regulation unit (MSR) needs to be implemented in the individual systems. Technically, the opportunities for measuring energy demands in a highly resolved frequency are given, however, they still have to be implemented and rolled out over more systems.

In local energy supply systems, an early integration into the power grid is desirable. The systems are built, connected and equipped with digital MSR technology. To control a system, a prediction of the power load is necessary, which takes into account the specific situation in the periphery and user behavior. However, shortly after equipping the system with the MSR, there is no or little data available. Since the load demand is, among other factors, dependent on the outside temperature having an annual seasonality, ideally at least one year of consumption data should be available for a data-based forecast. However, this is not the case shortly after meters have been installed or in newly built areas. Nevertheless, a precise forecast is required.

This is the situation that motivates this thesis. Many models proposed for STLF still need to be trained on a lot of past data. But when a new system is being equipped with a MSR, no past data is available. Schedules still need to be set and the expected power demand is still the most important boundary condition. This lack of past data is something that most works do not consider and that numerous forecasting models are not capable to work with, as they usually need a lot of data to be trained on.

There are approaches to estimate energy demands before any data is available [33]. However, these do not solve the problem of situationally relevant forecasts being needed in the smart grid.

In this thesis, a novel NMF-based forecasting model is being presented which is developed for forecasting load profiles at lower aggregation levels, as they occur in the situation of a decentralized energy supply.

Furthermore, an iteration-based framework is presented to analyze how well different forecasting models work based on the provided amount of training data. This not only makes it possible to compare models when little data is available versus when the data

history goes far back, but can even provide a view on how the amount of training data affects each model's forecasting performance.

The goal of this thesis is to perform an analysis of different forecasting models based on the amount of training data, allowing for a reflected decision on which models are eligible for situations without much training data.

The rest of this thesis is structured as follows: Chapter 3 narrows down the task of Load Forecasting and explains associated considerations. In Chapter 4, a survey about relevant literature on time series forecasting in general and Load Forecasting in detail is given. Moreover, a short survey about the use of one of the most utilized methods of this thesis, the Nonnegative Matrix Factorization and other applications of the used data set are given. Over the course of chapter 5, the theory and background of applied methods are presented, before a detailed description and analysis of the used data set is given in chapter 6. In chapter 7, the created forecasting models are explained, before chapter 8 lays out the process of how these models are iteratively compared to each other. The results of the model comparisons are presented in chapter 9 and discussed in chapter 10. The last chapter, chapter 11, finally concludes the performed research. All developed forecasting models were implemented using the Python programming language.

# 3. Load forecasting

> Difficult to see. Always in motion is the future.
>
> Yoda

Load Forecasting is the process of predicting how a time series with measurements of energy load will look like in the future. A time series in this context is a series of numerical data points in time order and indexed in time. Load Forecasting thus attempts to estimate the discrete numerical data points for future time indices.

Load Forecasting is a subfield of time series analysis. In contrast to the analysis of tabular data, where we can assume that observations are independent and identically distributed, the correlation among adjacent points, caused by the dependence on time, must be considered in time series research and can prevent the application of many traditional statistical analytics. From the viewpoint of a regression problem, many Load Forecasting models use past values of the time series together with some information about the future days, such as information about weekends, holidays or the expected weather situation as input, to output the prediction of values for future indices. This can be done for a single point in time or for a horizon ranging from a few minutes up to multiple years.

For years researchers have tried to find a universal forecasting model that can be used for all types of time series. However, it turned out that using models which are designed for the specific case in which they are applied is more successful and that there is no universal solution to Time Series Forecasting (TSF) in general or to Load Forecasting in particular [27, 29].

It is therefore important to analyze and understand the energy consumption of households on smaller aggregation levels and to develop models for load profile forecasting specifically for the situation of decentralized energy supply.

## 3.1. Different time ranges

When predicting load profiles, a distinction is made between four areas of the forecast horizon. Long Term Load Forecasting considers forecasting horizons of more than three years, Medium Term Load Forecasting covers the range of up to three years. Forecasts for a period of less than two weeks are assigned to Short Term Load Forecasting and Very Short Term Load Forecasting describes the forecast over a horizon of less than a day [30].

The different forecast horizons aim at different applications. Very Short Term Load Forecasting is often done with the target of battery control or demand response. Medium and Long Term Forecasts can be used for planning grid maintenance and extensions and for general investment planning [34]. When creating specific schedules for decentralized energy utilities and energy trading on shorter terms, it is usually important to forecast the period over the next one to two days. The research of this thesis can thus be assigned to the field of Short Term Load Forecasting (STLF).

## 3.2. Different aggregation levels

Since large power plants serve a large number of end-users, the individual consumption habits of the users overlap and the use of mean values from historical data provides a good basis for the forecasts to be made. At the local level of smaller aggregations, however, the user behavior of individual consumers plays a larger role and the considered time series are significantly more volatile. Pure averages or simple statistics are not enough to make reliable predictions. Instead, the used forecasting tool must be able to include the characteristics of the individual user behavior into the forecasts.

While the time series of a load profile shows recognizable seasonality and relatively smooth properties at large aggregation levels, these are noisier at aggregation levels of houses or quarters. The statistical properties of the load profile change if different user behavior is shown.

Since the behavior of individual consumers shows complex dynamics, the observed aggregation levels of neighborhoods are not sufficient to statistically average them. Additionally, such time series sometimes can show irregular behavior, caused by external variables such as weather, holidays or vacation times. A part of this behavior also happens without a recognizable system and can be understood as noise.

Data of electric load has one outstanding feature, i.e. a relatively clear daily seasonality. So it is a reasonable approach in forecasting to assume that a future day is likely to be similar to a day in the past, with similar properties, like the weather, season, or day of the week. These approaches work well when the demand profiles are smooth and regular. However, when the demand profiles of consumers are more irregular, the assumption of similarity does not hold as well. Consequently, algorithms that can also recreate those more volatile load profiles with the irregular peaks are needed [34].

Hayes et al. identify four different levels of load aggregation that are common in STLF tasks [35]:

1. Primary high-voltage / mid-voltage substation level with hundreds to thousands of customers
2. Secondary mid-voltage / low-voltage substation level with a few hundreds of customers
3. Low-voltage feeder level with few to tens of customers

4. End-user level of a single customer

The data used in this thesis describes the aggregated load of 350 residential households and can thus be categorized on the level of secondary substations.

Local demand profiles are typically more noisy and volatile, making STLF more difficult than on higher levels of aggregation. Hayes et al. show that at the local level standard STLF models may not be effective, and that simple models created from historical smart meter data can give similar prediction accuracies [35]. Especially the load of individual households can be highly volatile and hard to predict. Furthermore, not forecasting individual households but aggregates can also reduce the computational cost without decreasing the forecasting accuracy, compared to single predictions that are aggregated afterward [36].

When predicting load on different levels of aggregation, the size of the feeder is one of the biggest factors influencing the forecast accuracy, with smaller feeders being more difficult to predict than larger feeders [31, 35].

But forecasting both aggregated loads and individual loads is necessary. One might argue that when an aggregation of households supplied by one feed is forecasted well, predictions of its individual customers become obsolete. But feeders not only supply energy to residential households but also to a diverse mix of other domestic and non-domestic customers, overnight storage heaters, street lighting, etc. and the combination varies from feeder to feeder. This variety will most likely increase with the rise of local renewable generators and storage technologies. So focusing on households helps to better understand one of the most volatile parts of the load of these feeders. In addition, it is mentioned that aggregated load profiles depend strongly on the weather and have clearer signs of seasonality [37]. Thus, looking at individual profiles can help to understand what other factors can affect the load. It is important to understand both, aggregations of load profiles and the individual profiles.

Nevertheless, in the context of new local energy distributions, we will focus on the aggregated data of 350 residential households, as such a forecast would likely be the basis for schedule creations.

# 4. Survey of relevant literature

So many books, so little time.

Frank Zappa

In this chapter, an overview of research areas relevant to this thesis is presented. The context of this work is explained and similarities and differences to other works are discussed. First, an overview over the field of Time Series Forecasting (TSF) in general and Short Term Load Forecasting (STLF) specifically is given. Afterward, the use of Nonnegative Matrix Factorization (NMF) is reviewed. Additionally, it is discussed how forecasting models are usually evaluated when comparing their performances. Finally, an overview of work that has been done on the same data set is given.

## 4.1. Time Series Forecasting

In general, anything that is observed sequentially over time is a time series [38]. In this work when talking about time series, we refer to a series of numerical data points in chronological order, indexed in time with a constant time interval between the indices.

The current research in the field of time series analysis can be subdivided into two main areas: Time Series Forecasting (TSF) and Time Series Classification (TSC). The goal of TSC is to classify entire time series by assigning a label to the series and to group similar time series into clusters. TSF has a different focus. Instead of comparing the entire series $\boldsymbol{y}$ to other time series, TSF focuses more on the individual entries $y_t$ of the series. The goal is to give a prediction that indicates how the entries of the time series will continue in the future. These are two very different tasks, as TSC is a classical classification or clustering problem and TSF is more to be seen as a regression task.

The research field of TSF can further be divided into multiple subcategories, depending on their application. The arguably most prominent of which is the application in the field of Natural Language Processing (NLP) with tasks like machine translation. Another subcategory with increasing interest in the research community is the field of Load Forecasting, to which this thesis can be assigned. Additionally, there have been some mentionable industry applications of TSF lately.

These subcategories are shortly described with a more general focus on how and why TSF is important in these fields. The main part of this section focuses on describing common and important methods of TSF.

### 4.1.1. Research areas in Time Series Forecasting

The field of Natural Language Processing (NLP) is a subfield of computer science, Artificial Intelligence and linguistics. It deals with the problem of how computers can interact with the human language and how they can process and analyze language data [39]. Among other tasks, NLP includes speech recognition, speech segmentation, automatic text summarization and machine translation. As the output of these tasks is usually a sequence of characters and words, they can be seen as forecasting tasks, as the used models have to predict the next item in the sequence after each word, in order to finally receive a complete sentence or paragraph containing the required information. For a long time, the techniques used in NLP were dominated by traditional Machine Learning approaches, including linear models or Support Vector Machines trained on very high dimensional but sparse feature vectors. In recent years, however, the field has had major successes by using Neural Networks for the tasks [40, 41]. Especially Recurrent Neural Networks are used succesfully, e.g. for creating character-level language models [42].

The subcategory of Load Forecasting deals with the task of forecasting electrical or thermal loads over a certain horizon. This can range from very short horizons, like a few minutes or hours, up to long term forecasting of the expected loads of the next year or decade. In today's world, Load Forecasting is of major importance for several departments in power utilities, including the planning department, operations department and also the trading department [30].

Industry applications of TSF can appear in many different areas, as time series can be measured with every sensor that tracks data over time. The applications range from forecasting the number of items that a machine can produce in the next hour to the task of predicting which machine in a certain production line will most likely soon need maintenance and is likely to cause an error. The latter task is also known as predictive maintenance.

### 4.1.2. Models for Time Series Forecasting

Over the years, many different kinds of models have been applied to the Time Series Forecasting task, ranging from classical approaches, like Moving Average and ARIMA models, to more modern Machine Learning models, including those making use of Deep Neural Networks.

This section gives an overview of the most prominent techniques used in Time Series Forecasting before the next section narrows down on the models applied to the research area of load forecasting.

The forecasting of a time series can most often be divided into two parts. First, the structure and underlying patterns of the observed data are extracted into features. In a second step, a forecasting model is chosen and fitted to make future predictions from these features.

One can describe a time series signal as a composition of multiple parts [43, 44]:

- A trend, describing the general movement exhibited by the time series, without taking seasonality and irregularities into account;
- Seasonality, indicating periodic fluctuations and regular patterns;
- Uncertainties, that, for example, can be caused by external influences;
- Noise or Residuals, the remaining part of the time series that is mostly unexplainable;

The analysis of time series usually distinguishes between two types of time series, as is also often done in TSF: Univariate and multivariate time series. A univariate time series is a time series containing only a single observation for each time index, whereas a multivariate time series records multiple observations at each time step. For the analysis of multivariate time series, the interactions between the multiple observations at each time step are also considered [43].

The development of Time Series Forecasting models has come a long way, as can be seen from the results in the Makridakis competitions, a series of open competitions to evaluate and compare the accuracy of forecasting models. What is special about this series of competitions is that the models had to forecast multiple different time series.

In the first competition in 1982, the winning entry was a simple Exponential Smoothing applied to data that was deseasonalized with a classical multiplicative decomposition [27]. 45 years later, the participating models of the fourth competition in 2018 were more complex. A large number of the most accurate methods were a combination of multiple models, including ML and statistical approaches. The winning entry by Smyl et al. was based on a Recurrent Neural Network design, combined with Exponential Smoothing [45]. However, among the more accurate methods, many statistical approaches were still present [46].

Due to these findings, the more traditional statistical approaches are also included in the presentation of forecasting models in this chapter.

#### Exponential Smoothing

Exponential Smoothing is a method that was proposed by Brown, Holt and Winters in the 1950s [47, 48, 49]. The idea behind Exponential Smoothing is to forecast the next data point in a time series as a weighted average of past observations. The assigned weights decay exponentially with increasing time distance to the target point. The more recent the observation, the higher the assigned weight. This approach is relatively simple and can quickly be applied to a wide range of time series, leading to its major importance in industry applications [38].

There are three versions of Exponential Smoothing with increasing complexity. The simplest form of which is the so-called simple Exponential Smoothing, used for time

series without a clear trend or seasonal pattern. The forecast $\hat{y}_t$ for the time index $t$ is computed by a weighted average over the past observations with exponentially decreasing weights

$$\hat{y}_t = \alpha y_{t-1} + \alpha(1-\alpha)y_{t-2} + \alpha(1-\alpha)^2 y_{t-3} + \cdots, \tag{4.1}$$

where the smoothing parameter $0 \leq \alpha \leq 1$ controls how fast the weights decrease. Thus for a larger $\alpha$ the model is more sensitive to more recent observations. The sum ranges over all past observations from $y_1$ to $y_{t-1}$.

Alternatively, the equations of Exponential Smoothing can be written in the so-called component form, which extends to the more complex forms of exponential smoothing. For this, the time series is written as a sum of different components [38]. For simple Exponential Smoothing, only one component, the level $\ell$ is considered. The issue with simple Exponential Smoothing is that it is not suitable for data with a trend or seasonal variations.

This can be tackled by using Holt's linear trend method, or double exponential smoothing [48]. This extension to simple Exponential Smoothing introduces a trend component $b$ and a second smoothing parameter $0 \leq \beta \leq 1$ for this trend, in addition to the already known level $\ell$ and its parameter $\alpha$.

As this is still not able to deal with seasonal data, Holt and Winters proposed the Holt-Winters' seasonal method, also known as triple Exponential Smoothing, to include seasonality into the model [48, 49]. For this method, a third equation is added to the component form of the Exponential Smoothing. This equation introduces a smoothing parameter $0 \leq \gamma \leq 1$ for the smoothing of a seasonal component $s$.

For triple Exponential Smoothing, an additive and a multiplicative method exist. The additive method is suitable if the seasonal variations are roughly constant throughout the time series [38].

In contrast to the additive form, multiplicative triple Exponential Smoothing is preferred when the magnitude of the seasonal variations depends on the level of the series [38].

**ARIMA**

If time series stem from human behavior, they usually have some temporal dependence. This is due to our circadian rhythm [34]. If past observations are a reliable indicator, it makes sense to build models that forecast the future based on these past data points. Box and Jenkins incorporated this idea of using past values as regressors for future value in the arguably most well-known methods in univariate Time Series Forecasting: The group of Autoregressive Integrated Moving Average (ARIMA) models [50].

The approach of ARIMA models is different from the one of Exponential Smoothing models. The components level, trend and season are not modeled separately, but instead, trend and season are removed by differencing the data, to receive a time series that is then treated as stationary (i.e. its statistical properties remain constant over time) [34]. Overall ARIMA melts three processes into one larger model idea. Leveraging Autoregression

(AR), using Moving Averages (MA) and differencing the data before later integrating (I) it back into its original form.

For the Autoregression part, the idea is to use a simple Linear Regression with past values as predictors. Therefore, the prediction for $y_t$ is a linear combination of past observations of the time series itself. Hence, the name Autoregression. For this regression, an order $p$ has to be specified, defining the lookback range, i.e. the number of past observations that are to be included in the regression. An Autoregressive model of order $p$ is usually referred to as an $AR(p)$ model and be written as

$$\hat{y}_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t, \tag{4.2}$$

with weights $\phi_i$, a constant $c$ and $\varepsilon_t$ being white noise.

The idea of a Moving Average is a similar but different one. Rather than using past values of the regression target, a Moving Average model of the order $q$ or $MA(q)$ model uses the past $q$ forecast errors as predictors in a Linear Regression model, leading to a forecast equation of

$$\hat{y}_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}, \tag{4.3}$$

again with a constant $c$ and $\varepsilon_t$ being white noise and $\theta_i$ denoting the weights.

A time series can be called stationary if its statistical properties, like mean or variance, are independent of the time. A time series with a trend or a seasonality is therefore not stationary. One way of making a time series stationary is to remove its trend and seasonalities by differencing i.e. computing the differences between consecutive observations (in the case of trend) or between the observations one season apart (in the case of seasonality). This can help to reduce or even remove trend and seasonality and make a time series (more) stationary by stabilizing the mean. Other transformations, like taking the logarithm of a time series, can additionally help to stabilize the variance of a time series [38].

If we now combine the Autoregression approach with the Moving Average approach and apply it to a differenced time series, we obtain an Autoregressive Integrated Moving Average (ARIMA) model. Here the I stands for Integration, which is the reverse of the differencing process. The regression equation for an ARIMA model can be written as

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t,, \tag{4.4}$$

with $y'_t$ denoting the entries of the differenced time series. Denoting the number of times the original time series $y$ was differenced with $d$, this is called an $ARIMA(p, d, q)$ model.

These orders $p$, $d$, and $q$ have to be determined before fitting the model.

**VAR**

The ARIMA models are useful, but are aimed at univariate time series and do not scale well to multidimensional data. Instead an approach called Vector Autoregression (VAR)

is used in multidimensional time series forecasting. VAR is a generalization of the $AR$ models that were already discussed, i.e. a Linear Regression of past values. For a time series with $k$-dimensional observations $\boldsymbol{y_t}$, we may write the forecasting equation of a VAR model of order $p$ as

$$\hat{\boldsymbol{y}}_t = \boldsymbol{c} + \mathbf{A_1}\boldsymbol{y}_{t-1} + \mathbf{A_2}\boldsymbol{y}_{t-2} + \dots \mathbf{A_p}\boldsymbol{y}_{t-p} + \boldsymbol{\varepsilon}_t, \tag{4.5}$$

with $\boldsymbol{c}$ being a constant $k$-dimensional vector, $\boldsymbol{\varepsilon_t}$ denoting white noise in each of the $k$ dimensions and $\mathbf{A_i}$ being $k \times k$ matrices of weights that are constant over time. VAR models are arguably the most well known model in mutlidimensional TSF, however, with their linear approach they are unable to capture non-linearities in the data [51].

### Machine Learning approaches in general

In recent years, Machine Learning (ML) and Artificial Intelligence (AI) models have received a lot of attention for their contributions in many computer science tasks. As systems relying on hard-coded knowledge had difficulties in many fields, AI systems came to use, having the ability to extract patterns from raw data and thus acquire knowledge. This capability is also often called Machine Learning [52]. One challenge in the AI community is to create models that are successful at tasks that are relatively easy for humans to perform but at the same time are hard to formalize, as we approach these tasks intuitively. This includes recognizing spoken words or faces in images [52].

While most of the statistical models that were looked at so far assume that the data follows some underlying structure that can be described by a mathematical relation, ML methods use fewer assumptions. They simply expect the model to output the same values when receiving the same input. As this does not include any requirements on how the relation between input and output looks, most ML models can deal with non-linearities.

TSF can be phrased as a task of supervised learning. For new unknown states $\boldsymbol{X_t}$ the corresponding label $\boldsymbol{y_t}$ is to be predicted. Model parameters are learned via Machine Learning by iteratively adapting model parameter values until a certain loss function is minimized [53].

This is usually done by splitting the available data into three sets: Training, validation and test. The model is trained on the training set and evaluated on the second split of the data - the validation set. With the received evaluation results, some decisions about the model design can be made before a final evaluation of the model's performance is done on the test set. Minimizing the training loss should enable proper modeling of the data distribution, while at the same time the model's complexity should be kept small to avoid overfitting the training data. When choosing a model, the concept of "Ockham's razor" is to be applied. It states that if multiple explanations are to be considered, the simplest one is usually the right one.

### Support Vector Regression

One commonly used ML method that also comes to use in TSF are Support Vector Machines (SVM). A SVM is originally a tool for classification tasks. It classifies data by finding a boundary separating different clusters, such that the margin between two categories is maximized. However, it may also be used for regression tasks. Support Vector Regression (SVR) follows a similar concept as SVM but instead of trying to separate categories with a large margin, the goal of SVR is to find a function that best approximates the data while ignoring data points that are closer to this function than a given margin $\varepsilon$. It thus learns the target $\boldsymbol{y_t}$ to be a function of inputs $\boldsymbol{X_t}$ up to an error tolerance margin $\varepsilon$. For non-linear solutions, both SVM and SVR first map the data to a higher dimensional space using so-called kernels, before classifying or performing the regression task [54]. However, it has to be noted that the type of kernel and, thus, the type of non-linearity must be chosen in advance. As a regression technique, SVR can naturally be used for load forecasting.

### Neural Network models

Of all developments in the ML and AI research, the use of artificial Neural Networks (NN), especially with Deep Learning (DL) architectures, has caused a sensation in many areas in recent years [55]. Above all in the areas of image classification [56, 57, 58] and image segmentation [59], groundbreaking successes could be achieved with the help of NNs. Also in the area of Natural Language Processing (NLP) the use of Neural Networks led to great advances e.g. in document classification [40] , speech recognition [60] or machine translation [61, 62, 41]. Many researchers have been working on applying this potential to the problems in TSF in general and STLF in particular [63].

The success of most Neural Network models used in TSF relies on two types of architectures: Convolutional Neural Networks and Recurrent Neural Networks.

Recurrent Neural Networks (RNN) are the go-to architecture for sequential data and thus also for time series. The idea behind RNNs is that the neurons at a certain time step get inputs from other neurons at previous time steps [55]. RNNs today are mostly used by leveraging the Long Short Term Memory (LSTM) architecture developed by Hochreiter and Schmidhuber [64]. This special form of RNNs is characterized by the internal structure of the individual neurons, out of which the network is built. These neurons keep an internal state and use a gating mechanism, allowing them to overcome the vanishing gradient problem of basic RNNs. In addition, the internal state helps to capture long-range dependencies in the data.

The second type of architecture that is heavily used is the Convolutional Neural Network (CNN) [56]. Two of the main ideas of CNNs are using local connections and weight sharing [55]. In its most basic form, one can imagine the weights of a CNN being a learned filter sliding over the data. At each position, the filter stays the same, and a discrete convolution with the data is computed, indicating how well the filter represents

the data and allowing the network to recognize local patterns.

Originally, CNNs were mainly used for image data, with most developments in image classification being driven by the Imagenet data set [65]. The introduction of CNNs in 2012 by Krizhevsky et al. beat other models on this data set by a large margin and soon became the state-of-the-art [56]. The models were further improved with deeper architectures and ideas like residual connections, however, the main idea of convolutions stayed the same [58].

The idea of CNNs was later adopted to time series data by using 1D-Convolutions and moving filters across the time axis of the data. Networks using this approach are also referred to as Temporal Convolution Networks (TCN). For multivariate time series, these filters can either reach across multiple input dimensions or use separate filters for each input. The convolutional approaches proved to be very successful not only on image data but also for sequence modeling and even outperformed LSTM architectures across a range of tasks while demonstrating longer effective memory [66].

For TSF tasks, it must be ensured that no information leakage from future to past can happen. This problem is addressed by the idea of causal convolutions. Causal convolutions can replace the normal convolution operation such that only the past data is used for forecasting and the filters have no access to the future samples [67].

In the past few years, as NNs also gained more attention in the TSF field, many different models have been proposed, some of which achieved great forecasting accuracy.

The winning entry to the M4 competition in 2018 [46] was a hybrid model combining Exponential Smoothing (ES) and an LSTM based RNN by Smyl et al. called ES-RNN [45]. Their approach applied Holt-Winters' triple Exponential Smoothing onto the saved hidden states that were created by a stack of dilated LSTM networks with residual connections.

As the ES-RNN still used a statistical approach as part of the model, Makridakis et al. concluded that pure ML models are not as accurate as statistical ones [46]. However, in 2019 Oreshkin et al. proposed Neural Basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS) [68]. Their architecture is a pure Deep Learning method for TSF, beating the score of the ES-RNN at the M4 competition and thus proving that a pure ML model can outperform statistical models and their combinations with ML models in univariate TSF tasks [68].

In 2018 Lai et al. proposed a new Deep Learning (DL) architecture achieving state-of-the-art level for multivariate TSF: Long- and Short-term Time-series network (LSTNet) [69]. LSTNet uses both CNNs and RNNs. The CNN part is used to extract short-term local dependency patterns among variables and the RNNs help discover long-term patterns in the time series. In addition, they leverage an autoregressive approach in their model design. This results in an approach combining both linear and non-linear models for a robust prediction [69]. Their model was, as far as known, the first to be specifically designed for multivariate TSF with state-of-the-art (SOTA) results on multiple benchmark data sets.

Also designed for multivariate time series is the temporal pattern attention [51]. Shih et al. use a set of filters to extract time-invariant temporal patterns similar to a frequency domain transformation and apply an attention mechanism on the retrieved information, also leading to state-of-the-art results and, according to the authors, being superior in a few shortcomings of LSTNet, e.g. that the model does not have to be manually be tuned to match data periodicity. The temporal pattern attention allows the model to learn periodic patterns itself and also adapt to non-periodic data [51]. Their idea of feature-wise attention, for the model to learn interdependencies among multiple variables across all previous times and not only within the same time step, leads to state-of-the-art results.

The models ES-RNN, N-BEATS and the temporal pattern attention were also applied to stock market data and were able to showcase exceptional results in this domain and could be established as state-of-the-art for univariate time series [67].

### 4.1.3. Models used in Load Forecasting

After having looked at a general selection of models used for TSF problems, we now go into detail for the Short Term Load Forecasting (STLF) setting. In this setting, given some past load and some external factors like temperature or information about time and weekday, a forecast for the short-term future load is to be created.

In a review of forecasting methods for buildings, Amasyali et al. describe that there is no universal solution for STLF problems, but that the choice of method should be adapted to the respective forecast situation [29]. This results from the fact that the term Short Term Load Forecasting only refers to the requested time range of the forecast, but not to the kind of load data that is to be predicted. This can range from the power usage of an individual household to the power demand of entire countries, or refer to the volatile load profiles of renewable energy sources like solar and wind power plants.

There are several different approaches to address the problem of STLF. They can generally be classified into two main categories: Physical methods and data-driven methods.

For larger buildings, heat and electricity requirements are often calculated using physical methods relying on simulations and thermodynamic rules based on data like the building structure, heating, ventilation and air conditioning. However, this is hardly feasible for residences. There is indeed the procedure of collecting user or household data through surveys, based on which, for example, through clustering or fuzzy logic, suitable load profiles can be assigned [70, 33]. This is helpful to get a general idea of the range in which the power demand is likely to be. The main problem with this approach, however, is the data situation. On the one hand, collecting user data is extremely time-consuming and, on the other hand, it can hardly be guaranteed that the data is always up-to-date. This approach is thus not too helpful when forecasts for every day are required.

In contrast to physical methods, data-driven methods solely rely on historical and current data of the system that is to be forecast. Nowadays, with the increasing use of smart meters and other monitoring devices, such data can easily be made available,

encouraging researchers to use data-driven methods for their Load Forecasting tasks and making it a common way for providing load forecasts.

The methods used for Load Forecasting for decentralized producers or even single households differ from centralized ones. While the scheduling of larger power plants is largely based on statistical methods [25], some of the traditional forecasting methods prove to be ineffective on smaller aggregation levels [35, 71]. ML approaches in particular, though have led to good results on all aggregation levels.

Hayes et al. examine four different levels of demand aggregation present in the field of STLF (also see section 3.2) [35]. They found that, generally, the accuracy of forecasting models decreases remarkably when moving towards lower levels of load aggregation [35]. This coincides with the findings of Zufferey et al., who also mention the difficulties of forecasting individual households and increased accuracy for commercial and industrial loads [36]. It has also been shown to be more effective to predict already spatially aggregated time series instead of summing up individual forecasts for each household, as it reduces the computational cost without decreasing the forecasting accuracy [36]. Haben et al. also report the size of the feeder to be one of the biggest determining factors of their models' accuracy, with smaller feeders being more difficult to predict than larger feeders [31].

Many of the publications on STLF relate to the electricity demand of larger commercial buildings [43], use data from higher aggregation levels that are initially not considered in the course of a decentralized energy supply [72, 73], or even work with the data from entire countries or large cities [74, 75, 76]. Though some papers tackle the challenge of forecasting the power demand of individual households [77, 71], relatively few publications focus on aggregations of secondary substations [78, 31], as is done in this thesis.

#### Linear Regression

One of the more simple choices for a forecasting model is to do a Multiple Linear Regression on given inputs. Linear Regression has been mentioned in review papers multiple times and can be seen as a fairly popular method for STLF [79, 80].

#### Exponential Smoothing and ARIMA models

The main idea behind Linear Regression models is that the inputs $\boldsymbol{X_t}$ and the target variable $y_t$ are independent. However, this assumption usually does not hold for time series or any sequential data, and time series methods such as Exponential Smoothing and ARIMA models can be better suited for the data. Even though a careful parameter selection should be done when choosing the model [50] these model types have remained common baseline models [73].

Hagan and Behr applied ARIMA to Load Forecasting [81] as early as 1987. They reported this approach to be well suited, but to have problems when there was a

sudden change in the weather. As load profiles naturally show daily, weekly and yearly seasonality, Taylor used a multiplicative triple Exponential Smoothing for modeling weekly and daily seasonalities and adding an $AR(1)$ model for further improvement [82].

An extensive analysis of traditional methods on the low-voltage level was done by Haben et al. in 2019 [31]. They found that accurate forecasts can already be obtained by relatively simple methods. Even a simple average of the previous four weeks performed well for their data of multiple low-voltage feeders. Their best-performing methods among the tested statistical ones were those based on Autoregression and one based on Exponential Smoothing, namely Holt-Winters-Taylor Exponential Smoothing, an extension of triple Exponential Smoothing.

#### Machine Learning approaches in general

Machine Learning (ML) approaches are a good choice when it comes to extracting patterns from raw data and dealing with non-linearities [52]. The downside, however, is that the parameters of the models are often less easy to interpret than those of statistical models or Linear Regression approaches. But when it comes to their ability to deal with energy load data, ML models have proven themselves to be successful, ranging from high-voltage systems [74, 75, 76] to individual household level [71, 77].

#### Similar days approach

One approach leveraging the undeniable daily periodicity of load profiles is the approach to look for similar days in the past data. This has been done in multiple ways but is always based on this general idea.

Madal et al. use neural networks to recognize the similarity of a power load profile to past days and to learn similarity structures of the data [83]. Based on the similarity in the load profile and the associated temperature curve, a forecast is created by using subsequent values known from the past. They forecast the load by using information of the days that are similar to the weather condition of the forecasted day. Their forecasting strategy is a composite of two steps: First selecting similar days and second assuming that the load of the target day is similar to the load of the days following the days selected in step 1 [83].

The similarities of days do not have to be learned by Neural Networks, but can also be evaluated by using the k-Nearest-Neighbours (kNN) algorithm [74, 84].

A similar approach, grouping the targeted day with meteorologically similar days in data history and using their average load as prediction, was compared to ARIMA models by Cao et al. [85]. They demonstrated that, on ordinary days, the ARIMA models performed better, while the model based on the similar days approach showed better results on unordinary days.

A more complex hybrid model developed by Zheng et al. in 2017 uses the popular k-means algorithm with weighted feature importances to merge similar days into one

cluster and combining the similar days approach with Empirical Mode Decomposition and LSTM networks [72].

#### Tree-based models

Models based on decision trees [86] are not used very often in STLF, but are regularly successfully applied to tabular data [87, 88]. Nevertheless, there are a few applications, mostly of the Random Forest (RF) algorithm in the field of STLF.

Moon et al. use a RF algorithm to classify their data in the preprocessing before feeding it to a NN [89]. A direct use of RF Regression for Load Forecasting is applied by Dudek [90]. In this work, it could be shown that the RF forecaster yields a forecast accuracy similar to the one by the NN architecture, to which it was compared and it outperformed a single Classification and Regression Tree (CART), ARIMA and Exponential Smoothing models. Similar results concerning the accuracy of RF and NN models have also been found in other work [91, 92]. An application of gradient boosted trees to Load Forecasting on factory level is performed by Walther et al. [93].

According to these reports, it seems favorable to build forecasting models using tree-based algorithms instead of basic Neural Network models, as they have fewer parameters.

#### Support Vector Machine and Support Vector Regression

As already mentioned, a common technique used in TSF is the Support Vector Regression (SVR). It thus has also been applied regularly to load problems. In 2004 Chen et al. proposed a SVR model as winning entry to a Mid Term Load Forecasting competition [94]. Since then, it has also been used for STLF tasks regularly [95, 96, 97, 98], but the usage of SVR in STLF seems to have decreased while still being favored as a benchmark model in many publications.

#### NN models

In the last few years, Neural Network models have grown to become arguably the most popular technique in STLF because of their ability to model non-linearity [99].

In the early days, NN approaches still struggled with the issue that the models were over-parameterized and their forecasting results were not satisfying [63]. Some of the criticism also was focused on the issue that Neural Networks tended to overfit the training data [97]. With the development of LSTM [64] and CNN [100, 56] architectures, NN models gained more attention and grew to give better results. But simple NN models with shallow and fully connected architectures are also still used [101, 36].

Models with multiple hidden layers, so-called Deep Neural Networks (DNN), increase the models' capability of feature abstraction, making them more efficient in learning complex non-linear patterns [102]. In 2016 Mocanu et al. used a DNN model trained with Restricted Boltzmann Machines for Load Forecasting that showed superior performance to the benchmark SVR [103].

Promising results were also reported by Marino et al. by using a DL architecture based on a standard LSTM and an LSTM with sequence to sequence modeling on similar data [104]. Following the work of Mocanu and Marino in 2016, in 2017 Kong et al. also proposed an LSTM model and did an in-depth analysis of its performance on multiple single households before combining these forecasts to predict a low aggregation level [71]. Also in 2017 Ryu et al. proposed a model using DNNs, showing superior results to Holt-Winters Exponential Smoothing, ARIMA and a shallow NN when applied to industrial customers' electricity consumption data [105]. Shi et al. used a pooling-based deep RNN with LSTM neurons to predict the power demand of single households [77]. LSTM networks have not exclusively been applied to single households, but have also been successfully used to forecast aggregation levels of a few hundred households [106] and the demand of a wider metropolitan area [73]. A more detailed review on the usage of RNNs for sequence learning is given by Lipton et al. [107].

Instead of using Recurrent Neural Networks, in 2018 Kuo and Huang applied a Convolutional Neural Network model using a typical combination of convolutional layers and pooling layers to forecast hourly data three days into the future while receiving the past seven days as input [91]. An extensive analysis of shallow 1-dimensional CNNs was done by Lang et al. in 2019 [78]. They showed that good forecast quality can already be achieved with very basic CNN architectures, applying their models not only to higher aggregation levels but also to data on a low-voltage feeder level.

A combined approach using both CNNs and LSTM in parallel as feature extractors and later on combining these features for the forecast was proposed by Shao et al. in 2020. They report that their hybrid model is able to achieve state-of-the-art performance, beating both, LSTM and CNN models.

With the use of residual connections, Chen et al. transferred a concept very successful in other Artificial Intelligence applications to the field of STLF [108]. They propose a model using a ResNet based architecture and a two-stage ensemble strategy for forecasting loads of the next 24 hours.

The latest development in the field of NN applications in Load Forecasting tasks is the use of Neural Basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS) by Oreshkin et al. After proposing the model in 2019 and beating all other models on the data set of the M4 competition [68], the model was applied to Mid Term Load Forecasting in 2020, achieving promising results [109]. As far as known, the N-BEATS model has not yet been applied to Short Term Load Forecasting.

#### Hybrid models

Today, almost all proposed models in TSF are some kind of combination of different approaches, with one of the most prominent example being the ES-RNN model [45]. Such combinations of approaches have dominated the rankings of the M4 competition. Especially models combining traditional time series techniques with ML methods performed well [27]. Despite the huge success of hybrid models it has to be mentioned that

a combination of models alone does not guarantee good forecasting results. The most important element is a careful selection of appropriate models and their weights [27]. Makridakis et al. explain the advantage of hybrid models in the M4 competition with the design of the competition itself, containing multiple different time series and thus reducing the amount of background information that can be applied to each forecasting problem [27].

In the field of energy Load Forecasting hybrid forecasting techniques were also found to be dominant over single methods [110].

### 4.1.4. Preprocessings and features used in models

One aspect of model design is to choose the features on which to operate on and how these features may be extracted. Many models can perform poorly if they operate on bad features and many simple algorithms can excel, given a set of wisely chosen features. Most approaches do not use full end-to-end models operating on raw data but apply some kind of preprocessing and choose some representation for the data beforehand.

The choice of data representations has an enormous impact on the performance of ML models. Therefore, after mentioning the most common and promising model architectures, this section is designated to review some of the different features that were created and used to give load forecasts. The preprocessing can mostly be assigned to one of two categories: Automatic feature extraction or manual feature creation and selection.

Preprocessing can help to decompose a signal that can be understood as a combination of regular patterns, uncertainty (external influences) and noise [44]. Methods used for automatic feature extraction and data transformations include but are not limited to wavelet decompositions, application of the Empirical Mode Decomposition (EMD) and the use of autoencoders. Especially in forecasting tasks for renewable energy forecasting, wavelet decompositions and EMD are widely used methods [44]. As the original data often shows a variety of irregularities with non-linear and non-stationary features, these preprocessings can help to decompose the signal into multiple components that each behave better in terms of variance and outliers.

In load forecasting, the use of the EMD is seen regularly. For example, Qiu et al. propose a forecasting method that combines the idea of EMD with deep learning [111]. Here, the original time series is broken down into several additive modes, so-called Intrinsic Mode Functions (IMF), using the EMD. Then, a separate NN is trained for each IMF, with which the individual future IMF can be predicted. Finally, the predicted IMFs are combined into a weighted sum to predict the total signal. Zheng et al. similarly also decompose the raw signal into IMFs, but forecast each IMF with an LSTM network, while additionally using a XGBoost algorithm for determining feature importances and applying a similar days approach [72].

Kim and Cho use a different approach. They propose a forecasting model for energy demand that is based on an autoencoder, projecting the data onto an appropriate state for each given data situation and using a predictor to forecast the energy demand based

on that defined state [112].

Instead of applying an algorithm to detect features or decompose data automatically, one can also choose a set of features manually. This has the advantage that the extracted features often have an interpretable meaning and can be understood by domain experts.

An extensive set of features drawn from smart meter data has been presented by Hopf et al. in 2016 [113]. From one week of smart meter data, they extract 88 features describing statistical properties of the data of individual households. The features include several consumption patterns (e.g. on weekdays, in the morning, at night, maximal consumption, etc.), ratios (e.g. mean/max, night/day, weekday/weekend, etc.), statistical measures (e.g. mean, quartiles, etc.) and temporal information (e.g. time where the consumption was above a certain threshold). Even though their paper aimed at household classification, the use of these features is not limited to classification, but also gives a good general description of a week of smart meter time series.

A detailed process of analyzing features and then step-by-step adding and changing features used as input for a General Linear Model for forecasting is presented by Hong [30].

Calculating features from the data, whether automatically or manually, is an important process and can substantially improve or decrease forecasting quality. However, before deciding on a feature extraction technique usually a general analysis of the present data is done. In some cases, the data shows some structure that can be leveraged when deciding on how features are to be drawn. One prototypical example of this is to leverage seasonalities that are present in the data. Especially with time series influenced by human behavior, one often finds a temporal connection to the human circadian rhythm [34]. The power demand used in this thesis falls into this category, as it showcases a strong daily periodicity (also see section 6.1).

Leveraging this daily seasonality is commonly done when working with time series data, especially with power demand data. In most cases, this is done such that e.g. the load of the previous day or several days ago at the same time is used as a feature and used for the prediction of the target time. In this work, such features will be referred to as shift features or lag features, as a lagged or shifted time series is used when creating the features.

An example for the use of shift features is the work Chen et al. who used shifts of 1-7 days and 1,2,3,4,12,16,20 and 24 weeks as well as the past 24 hours as some of the inputs to their DL model for STLF [108]. Wei et al. not only used full days as shifts, but additionally used features like the load of 1h and 2h ago on the previous day, and the load of 1h and 2h ago two days before the target day [114]. Bouktif et al. also use lags as features for their model. However, instead of choosing lags by hand, they apply a genetic algorithm to find the appropriate number of lags for their time series model. This should help to find appropriate time lags to eliminate redundant features and improve the forecast accuracy [73].

Some authors go even further and divide the original load time series into slices. This can be useful, as the model may then operate on these slices which can simplify the forecasting problem, especially when the time series exhibits non-stationarity, heteroscedasticity, trend and multiple seasonal cycles [90]. Dudek uses this approach to divide the data into daily cycles [90]. This can be seen in connection with the similar days approach (see section 4.1.3). Here, the chosen similarity measure operates on daily slices and thus also leverages the daily seasonality present in the data. Following a similar idea, Zheng et al. apply a k-means algorithm to cluster similar days [72].

## 4.2. Nonnegative Matrix Factorization

Nonnegative Matrix Factorization (NMF) is a matrix factorization technique with the goal of factorizing a non-negative matrix $\mathbf{V}$ into two matrices $\mathbf{W}$ and $\mathbf{H}$ that are also non-negative. NMF was first introduced by Paatero and Tapper in 1994 [115]. However, the main attention was drawn to NMF after the seminal paper by Lee and Seung published in Nature in 1999 [116]. The authors showed that NMF has the ability to learn parts of objects and extract meaningful representations from data.

Since then, NMF has become a widely-used tool for the analysis of non-negative data in various fields of research. It has found applications in fields like image processing [117, 118], Natural Language Processing [119, 120], computational biology [121, 122] and in industrial projects [123, 124].

One of the main properties of NMF is that, when applied on high-dimensional data, it can extract meaningful and, due to non-negativity, easily interpretable features [116]. Lately, NMF has also found use in time series related projects. Weiderer et al. applied NMF on sensory data from an industrial process and were able to extract meaningful components related to the physics happening in the sensored process [124]. An application on speech data was proposed by Tseng et al. [125]. They used sparse NMF to extract features from a noisy single observation with the target of enhancing the speech quality in that observation.

In the context of energy-related data, NMF has, among others, been applied for the task of energy disaggregation i.e. to extract device-level energy consumption information from an aggregated signal without having to monitor every single device [126, 127]. Kim and Giannakis apply a combination of low-rank factorization and sparse matrix factorization similar to NMF on a data matrix of load time series of multiple sites [128]. They use this method to impute missing values in the data matrix and to forecast data by adding a column to their matrix. Mei et al. propose a NMF method that takes side information into account and uses this to predict values of interest for new, previously unseen rows and columns in their data matrix, to estimate and predict electricity loads of temporal aggregates [129].

Seemingly closer to the work in this thesis, but conceptionally different is the application of NMF by Wei et al. [114]. After choosing multiple features including lags, statistical measures and information about external factors they apply NMF as a dimensionality

reduction technique to reduce the dimension of the input variables before feeding it to their regression model, whereas in this work NMF will be used at the stage of feature extraction.

## 4.3. Comparison of forecasting models

One crucial aspect when talking about different forecasting models is the comparison of their accuracy. The ways models are compared to each other can be described by three main criteria:

1. The number of time series used in the comparison,
2. the procedure of how the models are applied to each time series and
3. the measure(s) with which the models' performances are compared to each other.

The first thing that differs among the different papers is the number of time series to which models are applied. Many papers test models on a single time series or in a single application case. That is reasonable if models are designed to work in a specific usecase, as is often the case. Other works compare models on multiple time series, as is mostly done when general models for TSF are to be compared. One example of these comparisons are the M competitions by Makridakis et al. [27].

The second way to distinguish different comparisons is by the procedure of how the models are applied to the time series. One common procedure is to simply divide the available time series data into two (or three) parts. Namely the training (, the validation) and the test set. The models are trained on the training set (with some parameters chosen based on the performance on the validation set) and finally evaluated on the test set. It is important that the test set remains unseen by the models until the performance is to be evaluated.

However, the way the time series is split up varies. Some papers simply divide the data in temporal ordering (e.g. 80% training, 20% test) and base the models' comparison on a single error measure calculated on the test set performance [114, 112]. This often-used approach has advantages and disadvantages. On the one hand, this way of testing the models is easy to implement and also only outputs one measure or set of measures by which the models have to be compared. On the other hand, this can also be seen as a downside, as comparing models only on one single time range might not be sufficient for a detailed comparison. The second disadvantage is that the test data is not always representative of the entire time series, as it can be affected by seasonal effects. Whereas a random selection of test samples is applicable in other AI fields using data with independent samples, the time ordering of time series data makes this difficult. Therefore, if e.g. one year of data (January to December) is available and the last 20% of the data are used for testing, then the models' performance is only measured in winter months. Meaning that the comparison of the models based on the performance in winter

might not accord with how they perform in summer. However, such a split-up of the data can be reasonable if enough data is present for each of the sets to be representative of all seasons. This is the way it is done with the three used data sets by Shao et al. [130] or with the 70/30 split performed by Bouktif et al. using data from January 2008 to December 2016 [73].

In the case of having training and test data, each covering more than one year, one can argue that they are representative of the data. Nevertheless, if the data spans long time ranges for time series data with a trend or other changing statistical properties, the model trained on the training set could be biased towards this earlier part [27].

A different approach is chosen by some authors: They compare their models on multiple time ranges of different seasons [108, 72, 90]. E.g. the comparison applied by Chen et al. use two starting dates, one for determining hyperparameters and the other one for a final training of the model [108]. But the evaluation of the model is done on one single test set of two years of data. In their evaluation, they analyze the prediction accuracy in summer and winter separately.

In some cases, however, no discussion of the used test approach is done and a split of the available data based on a percentage [114, 112] or some temporal information (e.g. use the last available 30 days for testing) [77] is done without a comment on why the data is split up this way. In both cases, depending on the size of the data set training and test data can differ in their statistical properties, as is often the case when time series are affected by seasonal influences. When the entire data set is chosen from one season only, though, a percentage-based splitting can be reasonable, as is e.g. done by Kong et al. who only use data from August 2013 [71].

Some papers compare their models' performance on days in different seasons or have enough data to test their models on an entire year to avoid the above-mentioned pitfalls. Zheng et al. perform multiple experiments, one of which includes an evaluation of days among all four seasons and one using an entire year as the test range [72]. Dudek et al. also chose their test set to include days from January and July, in order to evaluate their models in more than one season [90].

Going even further and not only evaluating on different seasons but looking at the performance throughout the year, Kuo and Huang divide the data into overlapping slices of data in a rolling fashion [91]. They divide their twelve-month data set into ten slices with three months of data each. The first one includes January to March, the last one October to December. Then, for each of these slices, the first two months are used as training data and the evaluation of each model is done on the third month. Afterward, the performances for each slice are collected and averaged to get a final performance score. Similarly, Hong evaluated his models on a 4-year rolling basis of the nine years of available data, always using the first three years for training and the fourth year for testing [30].

Such a rolling procedure should be commended, as it allows the evaluation of the models' performance over the course of the data set. It is also closely related to a technique called time series cross-validation [38]. For the latter, the training data always starts at

the first data point in the data set and is continuously extended. The length of test data stays constant. This procedure is applied by Lahouar and Slama with increasing lengths of training data and one week of test data [92].

The third category determining the way models are compared is the used measures with which the models' performance on the test set(s) are evaluated. To determine the quality of a forecast, different measurements including the speed of calculation or interpretability have been used. However, the most commonly used way is to use an error measure for determining the forecast accuracy [131].

Many measures to determine the accuracy of a model are known. Among others, the $R^2$ value and information criteria like Akaike's Information Criterion (AIC) and Bayesian Information Criterion (BIC) can be used for this [38]. While the BIC is especially famous among statisticians - with the ability to select the true underlying model when given enough data - in reality, a true underlying model rarely exists [38].

Instead, in practice often used measures in STLF are Mean Absolute Error (MAE), Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) among the error metrics based on the absolute error calculation. Furthermore, Mean Absolute Percentage Error (MAPE) is a commonly used percentage error, calculated based on the forecast error relative to the value of the target time series at each time step. MAPE is used often, especially when comparing the performance of forecasting models across multiple time series [27]. A detailed survey of different forecast error measures is done by Shcherbakov et al. [131].

## 4.4. Applications of the Irish smart meter trials data set

The Irish smart meter trials data set [132] used in this thesis has been used before by several publications with different goals.

Hopf et al. work with the data of individual households with the goal of classifying these time series and deducing household properties [113]. Jacob et al. used both the data of single households and aggregated data [34]. The goal of their work was to create forecasts with one and 24 hours forecast horizons, with peak forecasting as their main focus. The forecasting of individual households with an LSTM-based Deep Neural Network was performed by Shi et al. [77]. Another mentionable publication using this data set was the work by Hayes et al. in 2015, focusing on local level loads and analyzing the effect of load aggregation [35]. They show that at the local level, standard STLF models may not be effective but simple models can already give good accuracy. The work by Lang et al. in 2019 applying 1D-CNNs on the data of multiple aggregation levels for STLF tasks showed that good forecast quality can already be achieved with basic CNN model architectures [78].

These publications show some of the multiple scenarios in which the data from the Irish smart meter trials can be used and provide the motivation to also make use of the data set in this thesis.

## 4.5. Conclusion

From what we have seen, a few conclusions can be drawn about what kind of research has been done in the field of STLF and which questions remain open.

First of all, we saw that in TSF, when working with multiple time series, hybrid models combining different techniques have proven to be successful. It was discussed that the suitability of a model highly depends on the target data set and, more often than not, a model is particularly useful in a specific data set context, but not in general [27, 110].

Even though combination-based models have proven themselves at competitions with multiple time series [46], in this thesis the goal is to find a reliable model for one specific data situation, i.e. for forecasting the aggregated power demand of 350 households with a 30-minute sample rate and for the horizon of up to 48 hours. The literature suggests that, in such a specific context, one single model can work reasonably well and it does not have to be a combination of many different approaches. The model designs in this thesis are, thus, focused on one main approach and compared with other model ideas instead of merging all approaches into one model.

Among the used ML regression models, often more traditional methods like Linear Regression, SVR or NNs with shallow, fully-connected architectures are found. In more recent publications, many authors use Neural Networks based on LSTM cells or with a convolution structure, sometimes even in a hybrid combination of both. On the one hand, authors report good forecasting quality of their NN models; on the other hand, NN-based models have repeatedly been reported to cause problems by overfitting and struggle with the "curse of dimensionality", describing the potential increase in model complexity when adding more dimensions to the input of the NN, leading to bad forecasts [97, 110].

Even though these issues seem to appear less with LSTM-based NNs and CNN models, models with a Neural Network architecture generally need a lot of data for training, as often thousands of parameters must be fitted. This can be a problem. Firstly, if the time series the model is applied to is not very long, it is questionable if the parameters can be fitted at all, and secondly, if the time series is longer, the question arises if all parameters have been fitted on recent relevant data, as the dynamics of the series may have changed and the model might be biased by the earlier time series part [27].

This thesis focuses on the situation of forecasting in a scenario where no vast amount of past data is available and, thus, NN-based models are not the first choice due to these reasons. However, for a fair analysis, a modern NN model will be implemented as a benchmark.

There is evidence in literature that simpler models without many layers or complex hybrid approaches already seem to yield relatively good forecasts, particularly on lower aggregation levels [35, 78]. The motivation thus was to find a simple reliable model structure that can potentially work with little amounts of data.

For this, tree-based models like the Random Forest (RF) are considered. These models are very successfully used on tabular data sets [87, 88].

As far as known, RF models have very rarely been mentioned for STLF tasks in the literature. The most extensive analysis on RF for STLF was done by Dudek et al. in 2015, reporting that their RF outperformed many simpler models while giving results as good as the ones of a NN, despite being easier to train and tune [90]. Comparable accuracy of RF and NN models has also been found by Lahouar and Slama in 2015 [92]. Similar results were reported by Kuo & Huang in 2018, who only used RF as a benchmark [91]. This benchmark was beaten by their NN approach but still performed noticeably well, given the relative simplicity of the model. What is especially noticeable about these results is that the RF algorithm has significantly fewer parameters to fit than NN models and, thus, is more likely to work well in situations with less training data. As the use of Random Forests seems to be a promising endeavor, they will play a major role in the STLF models used in this thesis.

The preprocessing of the data should always depend on the data situation that is to be analyzed if the model is only to be used in one specific situation. However, literature shows that utilizing lag features is a commonly used procedure that seems to work well [73, 108, 114, 90]. But as lag features do not always capture the entire data situation, some additional features inspired by the ones described by Hopf et al. will be used to supplement the lag features as input to the forecast models [113].

Furthermore, to fully exploit the daily cycles of smart meter data, the data is cut into daily slices, as it has already been done by Dudek et al. [90]. This offers the advantage of being able to treat each daily slice as one high-dimensional sample that is to be forecast. As forecasting an entire day is common in STLF, this reorganization of the data matches the goal. In addition, working with daily slices instead of a continuous time series makes it easier to work with dimensionality reduction techniques.

Similar to the work of Wei et al., a NMF will be applied to reduce the dimensionality of the data [114]. The crucial difference is that in the work of Wei et al., first a set of features including lagged values and information about weather variables was collected, and only afterward this set of features was used as input of the NMF for a dimensionality reduction. A regression model then received the lower dimension output of the NMF as input for the forecasts.

In this thesis, the Nonnegative Matrix Factorization is also used to reduce the dimension of data, but in a conceptionally different way. Instead of first computing features and then combining them into a lower dimension, the NMF is applied directly on the daily slices. These daily slices already capture the daily periodicity, reducing the complexity of data in each slice. The NMF then reduces each day to a few coefficients connected with interpretable inner-day modes. Applying the NMF follows the idea of reducing the dimensionality of the input data while using a relatively simple algorithm that does not need as much data as e.g. an autoencoder. At the same time, interpretable features can be received, on which regression models like a Random Forest can operate.

As far as known, such a NMF-RF hybrid model is unheard of and has not been used in STLF before.

Finally, the current state-of-the-art model in Time Series Forecasting is N-BEATS [68], making it a reasonable choice as a benchmark model for the forecasting task at hand. Additionally, as far as known, it has not been used in STLF yet, making this another valuable contribution of this thesis.

Many papers compare their used models with a single error measure. This can be done in multiple ways, ranging from only one test set to a more sophisticated evaluation with multiple test ranges across the seasons, up to applying time series cross-validation. But, as far as known, no extensive analysis looking at the effect of training data amounts on the models' performance for STLF has been published yet.

In this work, an approach based on time series cross-validation is used for the comparison of the proposed models to benchmark and baseline models. This is similar to how Lahouar and Slama compared their models [92] or to the model comparisons of Kuo and Huang or Hong, done in a rolling fashion [91, 30]. But instead of simply averaging the values of all evaluation steps or only looking at a few test examples, the focus is on the single evaluation steps.

The forecasting models proposed in this thesis are targeted at a situation with little data available. However, as time goes by, more data becomes available with smart metering. Thus, it is important to see, if, after a certain time, some models that might perform poorly on little data have increased in performance and can outperform models that delivered good results with only little data available.

Such an analysis is important in practice, as it provides information on how long a model should be used and at what point of data availability it might not be suited anymore.

As to the availability of data, the Irish smart meter trials are a great resource, as they not only provide the load demands of many individual households that can be aggregated as required, but they also provide data of over more than one year, which is a length perfectly suited for the goals of this thesis [132].

# 5. Used Methods

> The theory of probabilities is basically just common sense reduced to calculus.
>
> Pierre-Simon Laplace

Based on of a prototypical data set of the aggregated power usage of 350 residential households, a new forecasting technique is being developed for STLF by combining methods of matrix decomposition and simple regression techniques. The focus is set on making a forecast possible even if only small amounts of past data are available. For this, an elaborate analysis of the used data is performed before describing the developed model architectures. For data analysis and in the model architecture, multiple methods come to use that are described in this chapter. They range from matrix decompositions like PCA or NMF, to regression techniques like Random Forest and Neural Networks.

Classical time series analysis techniques are used to calculate autocorrelations and partial autocorrelations. Additionally, the Fourier Transform is presented via which seasonality in the data can be recognized. In order to extract features in different time scales, the data is transferred to a latency space with appropriate dimensions (see figure 5.1). The one-dimensional time series is converted into a high-dimensional data matrix, which is analyzed using various matrix decompositions. Due to the non-negativity of the consumption data, in addition to Principal Component Analysis (PCA), the application of Nonnegative Matrix Factorization (NMF) is also possible. The features generated with the different methods are finally used as input for a regression, with which the target time series can be predicted. Classical Linear Regression, as well as approaches such as Random Forest, are options that can be used for this. Several models are used as baseline, including NNs for which the basic theory will also be laid out.

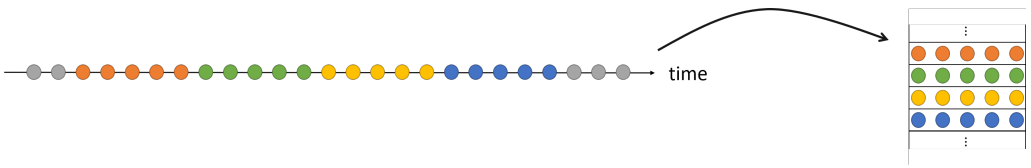



Figure 5.1.: Idea of the latent space of a time series. The time series is sliced into pieces of a given length, which are then stacked into a matrix. Here, each color represents one slice of the time series that is put into one row of the data matrix.

## 5.1. Classical time series methods

The data used in this thesis falls into the category of time series data. Before we look at the multiple methods applied to the present data, we need to understand what a time series is.

Generally, a time series is anything that can be observed sequentially over time [38]. In the context of this thesis, a time series is defined as a series of numerical data points in chronological order, observed in regular time intervals and indexed in time. The number of ordered points in a time series is also called the length of the time series.

The data points at each time step $t$ can be one- or multi-dimensional. As the used data consists of measurements of physical data, the data is restricted to be real-valued. If the data is one-dimensional, the time series is also referred to as univariate time series. If the observed data points are vectors, we call the series multivariate.

A univariate time series $\boldsymbol{x}$ with length $n$ is denoted as $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$, where $x_t \in \mathbb{R}$ $\forall t = 1, \ldots n$. And a multivariate time series $\mathbf{X}$ with dimension $m$ and length $n$ is denoted as $\mathbf{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$, where $\boldsymbol{x}_t \in \mathbb{R}^m$ $\forall t = 1, \ldots n$. A multivariate time series $\mathbf{X}$ of dimension $m$ can be understood as $m$ different univariate time series $\boldsymbol{x^1}, \boldsymbol{x^2}, \ldots \boldsymbol{x^m}$ with the same indices $t = 1, \ldots, n$.

For all time series, constant time intervals between the time indices are expected. Two time intervals are considered in this work: The first one, the time interval of the raw data, i.e. 30 minutes, is indexed with $t$ and carries information about the date and the time. E.g. $y_t = y_{2020-12-15\ 15:00}$ stands for the measurement at time $t$ = 2020-12-15 15:00. The second sample interval is one day, only carrying information about the date. To make the notation easier to distinguish, an index $i$ is used for daily samples. E.g. $\boldsymbol{v_i} = \boldsymbol{v_{2020-08-05}}$ describes a vector $\boldsymbol{v_i}$ with values sampled at $i$ = 2020-08-05.

The most important data type in this thesis are time series with measurements of energy load at each index $t$. These are to be forecasted.

What makes the forecasting and, generally, the analysis of time series data, particularly interesting, are time-related correlations in the data. Because of the sequential nature of the data, the observations cannot be assumed to be independent of one another or to be identically distributed. The correlations among adjacent points are ubiquitous and prevent the application of many conventional statistical methods. The systematic approach to answer questions posed by such time correlations can be referred to as the definition for time series analysis [133].

In time series forecasting, the forecast is usually made with respect to the index for which a value is requested. Thus, following a common Machine Learning notation and treating time series forecasting as a regression problem, the target load time series is denoted as $\boldsymbol{y}$. For such a forecast, external features are often used. If these features are specific for each time step, inputs $\boldsymbol{X_t}$ for each target $y_t$ are used. A second notation states that the target time series $\boldsymbol{y}$ can be predicted by a model based on the input time series $\mathbf{X}$.

The input time series is a multidimensional time series of external data, such as weather information. If it also includes more complex features that are created specifically for a certain model, the created feature matrix will be referred to as $\mathbf{X}^{\mathbf{F}}$.

Before a forecasting model for time series data is chosen, a detailed analysis of the present time series data is necessary. One common procedure is to decompose the time series into three components: trend, seasonality and residuals [134]. The trend describes a general movement, exhibited by the time series during the observation period, without taking into account irregularities or seasonality. The second component, the seasonality, captures periodic behavior of the time series. The last component, the residuals, captures the remaining part of the time series variations not explained by trend or seasonality. This analysis is often combined with a visual inspection of the plotted time series.

A second approach to analyzing time series is to look at the autocorrelation of a time series. The methods for the latter are described in the following subsections.

### 5.1.1. ACF and PACF

One important objective of time series analysis is to develop models that plausibly describe the data [133]. Tools often used for the analysis of time series are the autocorrelation function (ACF) and partial autocorrelation function (PACF), both based on the concept of autocovariance. When applying these tools, a time series $\{x_t\}$ is usually interpreted as a set of observations of an unknown distribution. A time series model is a specification of a sequence of random variables $\{X_t\}$ of which $\{x_t\}$ is postulated to be a realization [134].

The autocovariance function of a time series $\{x_t\}$ can then be defined as

$$\gamma(r,s) = \mathrm{Cov}(X_r, X_s) = \mathrm{E}\left[(X_r - \mu_r))(X_s - \mu_s)\right], \tag{5.1}$$

for indices $r$ and $s$ where $\mu_t = \mathrm{E}(X_t)$ is the mean function of $\{X_t\}$ [134]. It measures the linear dependence between two points $r$ and $s$ on the same series observed at different times. Note that a nonlinear dependence is not captured by the autocovariance function [133].

Assuming the time series to be stationary, i.e. that its statistical properties are constant over time, the autocovariance of two time indices can be reformulated to the autocovariance function of $\{x_t\}$ at the lag $h$ as

$$\gamma(h) = \mathrm{Cov}(X_{t+h}, X_t) \tag{5.2}$$

and, with this, define the autocorrelation function (ACF) of $\{x_t\}$ at lag $h$ as

$$\rho(h) = \frac{\gamma(h)}{\gamma(0)}. \tag{5.3}$$

Having measured a time series $\boldsymbol{x} = \{x_t\}$ with $n$ observations $x_1, x_2, \ldots, x_n$ these definitions can by applied by using the sample mean of the time series

$$\overline{x} = \frac{1}{n}\sum_{t=1}^{n} x_t \tag{5.4}$$

as the replacement for the mean function. This leads to the definition of the sample autocovariance function

$$\hat{\gamma}(h) = \frac{1}{n} \sum_{t=1}^{n-|h|} (x_{t+|h|} - \overline{x})(x_t - \overline{x}), \qquad -n < h < n \tag{5.5}$$

and the sample autocorrelation function

$$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)} \tag{5.6}$$

for the time series $\boldsymbol{x} = \{x_t\}$ [134]. Though technically the ACF and the sample ACF are two different constructs (one operating on sequences of random variables and the other operating on measured data), the term ACF will be used for both, as they follow the same concept in its definitions.

The ACF can be used as a tool to analyze a measured time series $\boldsymbol{x}$ on linear dependence between data points that are lagged by $h$. This is often done visually by plotting the ACF over the lag.

While the ACF is a great tool to evaluate the autocorrelation of a time series, it has one downside: for autoregressive models the ACF only declines slowly, thus assigning large correlation values to lags that are not the direct reason of the correlation but are merely correlated because of the autoregressive structure of the data. To overcome this, the partial autocorrelation function (PACF) is of great use. The PACF $\alpha(\cdot)$ is usually defined for Autoregressive Moving Average (ARMA) processes. Having an observed time series, it may be defined as

$$\hat{\alpha}(0) = 1 \tag{5.7}$$

and

$$\hat{\alpha}(h) = \hat{\phi}_{hh}, \qquad h \geq 1, \tag{5.8}$$

where $\hat{\phi}_{hh}$ is the last component of

$$\hat{\boldsymbol{\phi}}_{\boldsymbol{h}} = \hat{\boldsymbol{\Gamma}}_{\boldsymbol{h}}^{-\mathbf{1}} \hat{\boldsymbol{\gamma}}_{\boldsymbol{h}}, \tag{5.9}$$

with

$$\hat{\boldsymbol{\Gamma}}_{\boldsymbol{h}} = [\hat{\gamma}(i-j)]_{i,j=1}^{h} \tag{5.10}$$

and

$$\hat{\boldsymbol{\gamma}}_{\boldsymbol{h}} = [\hat{\gamma}(1), \hat{\gamma}(2), \ldots, \hat{\gamma}(h)]\,. \tag{5.11}$$

It can be shown, that the PACF of an $AR(p)$ process is zero for lags $\geq p$ [134].

In practice, ACF and PACF can be computed numerically and plotted, offering a good insight into linear dependencies between time steps in the given time series. An example for an $AR(2)$ process is given in figure 5.2.

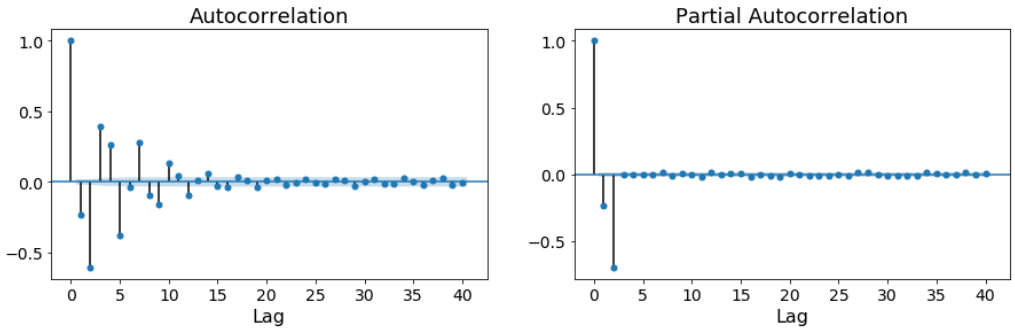



Figure 5.2.: autocorrelation function (ACF) and partial autocorrelation function (PACF) of the $AR(2)$ process $y_t = 1 + 0.4y_{t-1} + 0.7y_{t-2}$. After a lag of 2, the PACF values are no longer significantly different from zero.

### 5.1.2. Covariances for multivariate time series

After looking at univariate time series, we also take a look at multivariate ones. A time series of dimension $m$ is a sequence of random variables $\{\boldsymbol{X_t}\}$ of which a realization is denoted as $\{\boldsymbol{x_t}\}$. For multiple dimensions, in addition to the autocovariance of each dimension, the covariance among the individual components can be calculated, i.e. of $\{X_{t,i}\}$ and $\{X_{t,j}\}$, $i \neq j$. With $\mu_{t,i} = E(X_{t,i})$ the vector mean function of $\{\boldsymbol{X_t}\}$ is defined as

$$\boldsymbol{\mu_t} = E\left(\boldsymbol{X_t}\right) = \begin{pmatrix} \mu_{t,1} \\ \vdots \\ \mu_{t,m} \end{pmatrix}. \tag{5.12}$$

As an extension of the covariance in univariate time series $\gamma(\cdot)$, the covariance matrix of a stationary time series $\{\boldsymbol{X_t}\}$ for lag $h$ is defined as

$$\boldsymbol{\Gamma}(h) = E\left[\left(\boldsymbol{X_{t+h}} - \boldsymbol{\mu_{t+h}}\right)\left(\boldsymbol{X_t} - \boldsymbol{\mu_t}\right)^T\right]. \tag{5.13}$$

Denoting

$$\gamma_{ij}(h) = \mathrm{Cov}(X_{t+h,i}, X_{t,j}), \tag{5.14}$$

the covariance matrix for lag $h$ can be written as

$$\boldsymbol{\Gamma}(h) = \begin{pmatrix} \gamma_{11}(h) & \cdots & \gamma_{1m}(h) \\ \vdots & \ddots & \vdots \\ \gamma_{m1}(h) & \cdots & \gamma_{mm}(h) \end{pmatrix} \tag{5.15}$$

[134].

Having measured a time series $\mathbf{x} = \{\boldsymbol{x_t}\}$ with $n$ observations $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$ the theoretical covariance matrix can be estimated by calculating the sample covariance. Using the sample mean

$$\overline{\boldsymbol{x}} = \frac{1}{n}\sum_{t=1}^{n} \boldsymbol{x_t} \tag{5.16}$$

this estimation of the covariance matrix $\mathbf{\Gamma}(h)$ is given by

$$\hat{\mathbf{\Gamma}}(h) = \frac{1}{n}\sum_{t=1}^{n-h}(\boldsymbol{x_{t+h}} - \overline{\boldsymbol{x}})(\boldsymbol{x_t} - \overline{\boldsymbol{x}})^T \tag{5.17}$$

for $0 \leq h \leq n-1$ and $\hat{\mathbf{\Gamma}}(h) = \hat{\mathbf{\Gamma}}^T(-h)$ for $-n+1 \leq h < 0$ [134].

The sample cross-correlations of $\mathbf{x} = \{\boldsymbol{x_t}\}$ can then be calculated as

$$\hat{\rho}_{ij}(h) = \frac{\hat{\gamma}_{ij}(h)}{\sqrt{\hat{\gamma}_{ii}(0)\hat{\gamma}_{jj}(0)}}, \tag{5.18}$$

where $\hat{\gamma}_{ij}(h)$ is the $(i,j)$th component of $\hat{\mathbf{\Gamma}}(h)$ [134].

The cross-correlations provide a helpful tool to capture linear dependencies in multidimensional time series and can help to find linearly relevant lags $h$.

### 5.1.3. Fourier Transform

The former techniques for analyzing time series all looked at the relations of data points at different time steps and lags. Apart from these analyses in the time domain, a second approach to time series analysis exists, commonly referred to as the frequency domain approach or spectral analysis. In the frequency domain, the time series is not viewed in terms of lags but investigates which cycles and frequencies are important for describing the series [133].

Fourier showed that sine and cosine functions of increasing frequency provide an orthogonal basis for the Hilbert space of functions. From this, the Fourier analysis was born, expressing the function at hand through such periodic functions. By calculating coefficients in this basis of periodic functions, one can express the function in the frequency domain. The Fourier Transform has revolutionized computational mathematics and the Fast Fourier Transform (FFT) algorithm is arguably one of the most significant computational algorithms to date [135].

The idea behind Fourier Series and Fourier Transforms is that any function $f$ can be written in terms of periodic sine and cosine functions, as they build an orthogonal basis of the Hilbert space. A simpler notation emerges when combining both sine and cosine by using a complex exponential. Any function $f \in L_2([-L;L])$ restricted to the range from $-L$ to $L$ can be written as a sum

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{i\frac{k\pi}{L}x} \tag{5.19}$$

of exponentials with frequencies $\omega_k = \frac{k\pi}{L}$, where

$$c_k = \frac{1}{2\pi}\langle f, \psi_k\rangle = \frac{1}{2L}\int_{-L}^{L} f(\xi)\psi_k(\xi)d\xi \tag{5.20}$$

is the inner product of $f$ and the basis function $\psi_k(\xi) = e^{-i\omega_k\xi}$ with frequency $\omega_k$.

The Fourier Transform now extends this idea of the Fourier Series for a function limited to the range of $-L$ to $L$ to the limit $L \to \infty$, yielding

$$f(x) = \int_{-\infty}^{\infty} \frac{1}{2\pi} \underbrace{\int_{-\infty}^{\infty} f(\xi) e^{-i\omega\xi} d\xi}_{\hat{f}(\omega)} e^{i\omega x} d\omega. \tag{5.21}$$

We can thus denote the Fourier Transform of $f$ as

$$\hat{f}(\omega) = \mathcal{F}(f)(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx \tag{5.22}$$

and the inverse Fourier Transform as

$$f(x) = \mathcal{F}^{-1}(\hat{f})(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega x} d\omega. \tag{5.23}$$

Both integrals converge if $f, \hat{f} \in L^1(-\infty, \infty)$ belong to the space of Lebesque integrable functions [135].

The idea of this transform of analytical functions can be transferred to discrete measured data. For the Discrete Fourier Transform (DFT), different definitions are possible. Here the definition follows the one of the used numerical implementation, i.e. the implementation by the numerical Python library `numpy`. For a time series $\boldsymbol{f} = \{f_0, f_1, \dots f_{n-1}\}$ of length $n$, the DFT $\hat{\boldsymbol{f}} = \{\hat{f}_0, \hat{f}_1, \dots \hat{f}_{n-1}\}$ is calculated via

$$\hat{f}_k = \sum_{j=0}^{n-1} f_j \cdot e^{-i\frac{2\pi jk}{n}} \qquad k = 0, 1, \dots n-1 \tag{5.24}$$

and the inverse DFT as

$$f_k = \frac{1}{n} \sum_{j=0}^{n-1} \hat{f}_j \cdot e^{i\frac{2\pi jk}{n}} \qquad k = 0, 1, \dots n-1 \tag{5.25}$$

differing from the forward transform by the sign in the exponential and the normalization factor of $\frac{1}{n}$ [136].

Numerically, this transform is computed by using the Fast Fourier Transform (FFT) algorithm developed by Cooley and Tukey in 1965 [137].

Having a series of length $n$ and a sample rate of $dt$, the DFT yields the coefficients $\hat{f}_k$ for frequencies that are multiples of the fundamental frequency $df = \frac{1}{dt \cdot N}$. As not all frequencies will be present in the data, the complex $\hat{f}_k$ will have different magnitudes. $\hat{f}_k$ shows a higher magnitude if the frequency $\frac{k}{dt \cdot N}$ plays a more important role in the Fourier series of $\boldsymbol{f}$. Thus, looking at the magnitudes of the Fourier coefficient gives information about the prevalent frequencies in the frequency domain and, thus, information about which frequencies and cycles are prevalent in the time series.

A common way to analyze a time series on these frequencies is to transform it into the frequency domain and plot the coefficient magnitudes over the corresponding frequencies.

## 5.2. Principal Component Analysis

Principal Component Analysis (PCA) is an algorithm to project multidimensional data into a linear subspace spanned by orthogonal components, while at the same time keeping maximum variance of the data.

In many data sets the data is not evenly distributed over all dimensions but lies close to a manifold of lower dimension. This motivates the idea of reducing the dimensionality of the data. It can be shown that when a linear projection of $n$-dimensional data into a $d$-dimensional subspace is done, maximum variance is kept if this subspace is spanned by the eigenvectors corresponding to the $d$ largest eigenvalues of the covariance matrix of the data set [54].

PCA is commonly used for dimensionality reduction, data compression and data visualization [54]. For example, in practice, it can be helpful to visually inspect the projection of a data set onto the first two or three principal components, as they are easy to display, while at the same time presenting most of the variance in the data.

## 5.3. Nonnegative Matrix Factorization

NMF is a matrix factorization technique that belongs to a family of unsupervised machine learning techniques that also includes Principal Component Analysis (PCA), Independent Component Analysis (ICA) or Vector Quantization (VQ). All of these techniques generally have the goal of finding two matrices $\mathbf{W}$ and $\mathbf{H}$, such that a data matrix $\mathbf{V}$ can be factored into $\mathbf{V} \approx \mathbf{W} \cdot \mathbf{H}$. The difference among these methods are the constraints that each of these factorizations underlies. These constraints lead to different representational properties of the obtained matrix factors.

Where PCA expects the rows of $\mathbf{H}$ to be orthogonal and, therefore, uncorrelated, leading to a more distributed representation that uses cancellations to generate variability [138], ICA will make sure that the columns of $\mathbf{H}$ are statistically independent. VQ expects each row of $\mathbf{W}$ to be a sparse vector carrying a single entry of 1 and all others being zero, leading to a winner-takes-all clustering [139].

The constraints of NMF enforce a different kind of restriction. NMF not only expects the data matrix $\mathbf{V}$ to be already non-negative but also enforces non-negativity on the entries of $\mathbf{W}$ as well as on the entries of $\mathbf{H}$. This is a restriction to the data it can be used on: It must be ensured that the samples in $\mathbf{V}$ do not carry negative values. However, the constraints also yield some interesting and desirable properties, such as a parts-based representation of internal data structures and the ability to extract interpretable features from the data [116]. The learned basis vectors in $\mathbf{H}$ are used in combinations that are distributed but also sparse and, therefore, lead to interpretable reconstructions of the original data [140].

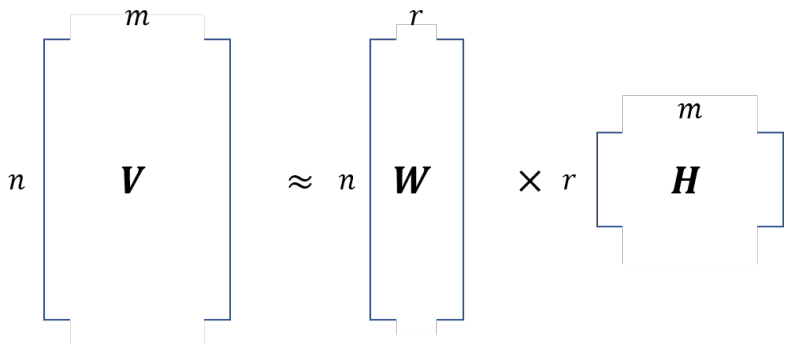



Figure 5.3.: Matrix description of the Nonnegative Matrix Factorization. The matrix $\mathbf{V} \in \mathbb{R}_+^{n \times m}$ is decomposed into two matrices $\mathbf{W} \in \mathbb{R}_+^{n \times r}$ and $\mathbf{H} \in \mathbb{R}_+^{r \times m}$

### 5.3.1. Formulae, non-negativity and matrix dimensions

Consider a non-negative data matrix $\mathbf{V} \in \mathbb{R}_+^{n \times m}$ of $n$ samples having $m$ features. NMF then finds a decomposition into two matrices $\mathbf{W} \in \mathbb{R}_+^{n \times r}$ and $\mathbf{H} \in \mathbb{R}_+^{r \times m}$ such that

$$\mathbf{V} \approx \mathbf{W} \cdot \mathbf{H}, \tag{5.26}$$

i.e. the NMF solves the problem

$$\min_{\mathbf{W} \geq 0, \mathbf{H} \geq 0} d(\mathbf{V}, \mathbf{WH}) \tag{5.27}$$

with $d : \mathbb{R}^{n \times m} \times \mathbb{R}^{n \times m} \to \mathbb{R}_+$ being some suitable distance function. The dimension $r < \min(n, m)$ is a hyperparameter describing the number of features being extracted by the decomposition.

Equation 5.26 can also be noted for each sample $\boldsymbol{v_i} \in \mathbb{R}_+^m$ as

$$\boldsymbol{v_i} \approx \boldsymbol{w_i} \cdot \mathbf{H} = \sum_{j=1}^{r} w_{i,j} \cdot \boldsymbol{h_j} \tag{5.28}$$

with $\boldsymbol{h_j}$ describing the feature rows of $\mathbf{H}$. We see that each sample $\boldsymbol{v_i}$ is approximated by a linear combination of the components or modes stored in the rows of $\mathbf{H}$, weighted by the coefficients stored in $\mathbf{W}$. The matrix $\mathbf{H}$ can therefore be regarded as containing a basis, optimized for the weighted linear approximation of $\mathbf{V}$. As usually $r \ll n$, an accurate approximation is only possible if the basis in $\mathbf{H}$ encodes structure that is latent in the data samples [141].

Unlike with the PCA or ICA, all entries of $\mathbf{H}$ and $\mathbf{W}$ are non-negative. The linear combination of each sample is therefore obtained in an additive way, without any subtraction happening. This leads to a weighted additive superposition of the components that is intuitive and interpretable [116]. It allows the NMF to produce a parts-based representation of the data matrix $\mathbf{V}$, with the parts being stored in the components matrix $\mathbf{H}$ [142].

In other words, given the constraint of all matrix entries being non-negative, the NMF yields a parts-based decomposition of $\mathbf{V}$ that follows the intuition of combining parts to a whole. As the human brain is expected to also think in a parts-based sparse way, the NMF algorithm outputs a factorization that, by design, is relatively intuitive and easy to understand [140, 143].

Lee and Seung showed that when NMF was applied to a set of images of faces, the algorithm managed to extract common modes present in all presented faces. These modes were found in the matrix $\mathbf{H}$ and resembled parts of faces, like parts of eyes, the mouth, or edges around the head [116].

The factorization described in equation 5.26 needs a hyperparameter to be set. With $n$ samples and $m$ features, the data matrix $\mathbf{V} \in \mathbb{R}_+^{n \times m}$ is factorized into $\mathbf{W} \in \mathbb{R}_+^{n \times r}$ and $\mathbf{H} \in \mathbb{R}_+^{r \times m}$. To make this possible, the inner dimension or rank of the factorization $r \leq \min(n, m)$ needs to be predetermined.

This is an important choice, as it defines the number of components that are to be extracted from $\mathbf{V}$ and stored in $\mathbf{H}$. A smaller $r$ lets the NMF find fewer components, meaning that the reconstruction is done by taking a sum over fewer parts, making it easier to interpret, but also less exact. The reconstruction error $d(\mathbf{V}, \mathbf{WH})$ can be expected to be higher than with a larger $r$. Setting $r$ to a higher value increases the number of components that are to be found in the data. The reconstruction is then to be understood as a weighted sum over more and hence smaller parts, decreasing the reconstruction error $d(\mathbf{V}, \mathbf{WH})$, but also making the reconstruction more difficult to interpret as a higher number of coefficients have to be looked at.

Generally, the inner dimension $r$ of the factorization is chosen such that

$$(n + m) \cdot r < n \cdot m. \tag{5.29}$$

The product $\mathbf{W} \cdot \mathbf{H}$ can then be treated as a compressed form of the data matrix $\mathbf{V}$ [116].

When choosing $r$ for the factorization, equation 5.29 provides a suitable range of possible values. A choice still has to be made. On the one hand, the dimensionality should be reduced as much as possible to make the data easier to handle and interpret. On the other hand, as much of the information as possible should be contained when compressing the data. This trade-off can be tackled by looking at how much of the original variance of the data $\mathbf{V}$ is contained in the reconstruction $\mathbf{W} \cdot \mathbf{H}$.

The same idea is common when performing a dimensionality reduction with PCA or Singular Value Decomposition (SVD). For this, it is computed how much of the original variance is kept, when projecting the data into the subspace spanned by the first $r$ singular vectors with the largest singular values. It is common to choose $r$ such that 90%, 95% or 99% of the original variance is kept under the projection. Alternatively, it is a common, though somewhat subjective, procedure to plot the contained variance over the number of components and look for a point at which the slope of this graph goes from "steep" to "flat" (often called an elbow), and only keep the components contributing to the steep slope [144].

When working with NMF rather than PCA, the found components are generally not orthogonal and the contained variance cannot be calculated by simply projecting the data in the subspace. However, what is possible is to calculate the contained variances by using the explained variance score. The explained variance measures the proportion to which a representation accounts for the variance of original data [145, 146]. For a data set $\mathbf{X}$ and its estimation $\hat{\mathbf{X}}$ the explained variance can be calculated as

$$explained\_variance(\mathbf{X}, \hat{\mathbf{X}}) = 1 - \frac{Var(\mathbf{X} - \hat{\mathbf{X}})}{Var(\mathbf{X})} \tag{5.30}$$

where $Var$ denotes the variance.

Using this metric, one may then again choose the inner dimension $r$ of the NMF so that 90%, 95% or 99% of variance of the data $\mathbf{V}$ is kept in the reconstruction $\mathbf{WH}$. Additionally, with NMF it is also possible to search for an elbow when plotting the reconstruction errors $d(\mathbf{V}, \mathbf{WH})$ over the internal dimension $r$.

### 5.3.2. Optimization

#### Loss functions

As equation 5.26 describes an approximation problem, one needs to choose how the quality of the approximation should be measured. There are several choices for this loss function on which the factorization is optimized. The most widely used function to measure the distance between $\mathbf{V}$ and $\mathbf{WH}$ is the well known Euclidean norm, which, in the context of being applied to matrices instead of vectors, is also called Frobenius norm

$$\|\mathbf{V} - \mathbf{WH}\|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} |V_{ij} - (WH)_{ij}|^2 \tag{5.31}$$

[147].

Apart from the Frobenius norm, the Kullback-Leibler (KL) divergence, given by

$$D_{KL}(\mathbf{V} \| \mathbf{WH}) = \sum_{i=1}^{n} \sum_{j=1}^{m} V_{ij} \log \frac{V_{ij}}{(WH)_{ij}} \tag{5.32}$$

often comes to use [148].

In the seminal paper of Lee and Seung the facial recognition data sets were decomposed utilizing a variant of KL divergence

$$L = \sum_{i=1}^{n} \sum_{j=1}^{m} \left( V_{ij} \log (WH)_{ij} - (WH)_{ij} \right) \tag{5.33}$$

as the loss [116]. They interpret NMF as a method for constructing a probabilistic model of image generation, where Poisson noise is added to the reconstruction $\mathbf{WH}$ pixel-wise.

Févotte et al. mention the opportunity to solve the approximation problem with a more general view by using the $\beta$-divergence as a loss function, of which both KL divergence and the Frobenius norm are special cases (with $\beta = 2,\ 1$ respectively) [148].

### Regularization

The non-negativity constraints impose some sparseness on the factorization, but in some cases, this might not be sufficient. If a certain distribution in **W** and **H** is desired, additional regularization terms can be added to the loss function. The two most prominent examples are $L2$ regularization, to enforce more distributed results, or $L1$ regularization, to receive additional sparseness. The regularization can be added either on both factor matrices or a single one [147].

### Optimization techniques and strategies

After choosing a loss function on which the approximation problem is to be optimized, an optimization technique needs to be chosen. The two most common choices are coordinate descent, which can only be used for the Frobenius norm, and multiplicative updates, which are applicable for every $\beta$-divergence [149, 141, 148]. For coordinate descent, a popular implementation uses the alternating least squares (ALS) algorithm. In its two-step iteration, first **H** is kept fixed and **W** is optimized. Next, the roles are switched, the newly found **W** is fixed, and **H** is optimized [149]. In each step, the loss function can decrease or stay the same, but must not increase. This iteration process is repeated until convergence. When using multiplicative updates, factors for both **H** and **W** are calculated in every iteration step. The two matrices are then multiplied with these factors. With the Euclidean distance as loss function, the updates are

$$H_{ij} \leftarrow H_{ij} \cdot \frac{(W^T V)_{ij}}{(W^T W H)_{ij}} \qquad W_{ij} \leftarrow W_{ij} \cdot \frac{(V H^T)_{ij}}{(W H H^T)_{ij}}. \tag{5.34}$$

It can be shown that the multiplicative update procedure is guaranteed to decrease the loss function [141]. For a detailed comparison of the most common NMF algorithms, see Zhou et al. [150].

Even though the decomposition $\mathbf{V} \approx \mathbf{WH}$ is linear, it needs to be pointed out that the non-negativity constraints lead to a non-linear and therefore non-convex problem [116]. Thus, different solvers may find different local minima even when being used on the same loss function and with the same initial conditions [151].

More recently, Le Roux et al. proposed a different approach, Deep NMF, in which they reformulate the NMF problem based on an Neural Network architecture and train it via a variant of backpropagation [152].

### Initialization

Without constraining the factorization problem of the NMF, the resulting matrices are not unique. Having a factorization $\mathbf{V} \approx \mathbf{WH}$, the factors can be scaled by an non-negative invertible matrix **S**, such that $\mathbf{V} \approx \mathbf{WSS}^{-1}\mathbf{H}$. The matrix **S** can be seen as a scaling and permutation of the components in **H**. In terms of interpretability, a scaling of the components, let alone a permutation, is not desirable. How to deal with this indeterminacy,

in general, remains an open question [123, 153, 154], however, there are ways to create a unique solution to the approximation problem. These approaches set a prior to the components and coefficients when initializing the factor matrices before running the optimization algorithm. Setting a solid initialization plays a key role in the performance of the NMF and the quality of its output.

The arguably most prominent initialization technique is a data-driven approach developed by Boutsidis and Gallopoulos, the so-called Non-negative Double Singular Value Decomposition (NNDSVD) [151]. Instead of random non-negative initializations, they propose a strategy based on SVD. This approach, in its basic form, is not based on any randomization and, therefore, converges to the same solution for the same algorithm on the same data. In conclusion, the scaling and order of the components in $\mathbf{H}$ are fixed.

NNDSVD is based on two SVD processes. The first one approximates the initial data matrix $\mathbf{V}$ and the second one is applied to positive sections of the partial SVD factors, exploiting an algebraic property of unit rank matrices. NNDSVD obtains initial columns and rows for $\mathbf{W}$ and $\mathbf{H}$ from the leading singular vectors of the positive section of each one of the first $r$ singular factors of $\mathbf{V}$.

Boutsidis and Gallopoulos point out that this initialization is likely to contain several zeros and, therefore, yield a more sparse factorization [151]. This can be desirable in some cases but is generally a downside when using NMF with multiplicative updates [141]. If a matrix entry is initialized with zero, this number will stay in place, as it cannot be changed by multiplying with any number. This reduces the capacity of the used approximation model. To address this problem, NNDSVD can be modified such that initial zeros are replaced with the average of $\mathbf{V}$, or with a random value close to this average. This modification, on the one hand, solves the problem of zeros staying put under multiplicative updates and, on the other hand, leads to a more dense representation. Hence, it should be used if sparsity is not necessarily desired [151].

However, it should also be noted that multiplicative updates can be seen as a special case of additive updates, by using a smart choice of the corresponding learning rate. Zeros generally do not pose a problem to additive updates [150].

The initializations for the matrix factors $\mathbf{W}$ and $\mathbf{H}$ are derived as follows: For any real-valued matrix $\mathbf{X}$ a Singular Value Decomposition is guaranteed to exist and to be unique up to sign. Using the notation of the economical SVD and assuming that $n > m$ we may decompose $\mathbf{X} \in \mathbb{R}^{n \times m}$ such that

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}, \tag{5.35}$$

where $\mathbf{U} \in \mathbb{R}^{n \times n}, \mathbf{\Sigma} \in \mathbb{R}^{n \times m}$ and $\mathbf{V} \in \mathbb{R}^{m \times m}$. $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. Their columns represent hierarchically arranged left and right singular vectors respectively. $\mathbf{\Sigma}$ is a diagonal matrix stacked atop of zeros, with the diagonal containing the singular values $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_m \geq 0$ of $X$ in decreasing order [135].

The decomposition 5.35 can also be rewritten as a sum over rank-1 matrices

$$\mathbf{X} = \sum_{j=1}^{m} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^T \tag{5.36}$$

with decreasing importance as the $\sigma_j$ decrease in magnitude. Truncating this sum after $r < m$ summands leads to a rank-$r$ approximation of the original matrix $\mathbf{X} \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^{\mathbf{T}}$. The Eckart-Young-Theorem gives

$$\underset{\tilde{\mathbf{X}} \text{ s.t. } rank(\tilde{\mathbf{X}})=r}{\operatorname{argmin}} \left\|\mathbf{X} - \tilde{\mathbf{X}}\right\|_F = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^{\mathbf{T}}, \tag{5.37}$$

stating that in terms of the Frobenius norm, the truncated SVD is the best rank-$r$ approximation of $\mathbf{X}$ [155]. One can therefore read the rank-$r$ approximation as a sum over rank-1 matrices.

$$\mathbf{X} \approx \sum_{j=1}^{r} \sigma_j \mathbf{C}^{(\mathrm{j})} \tag{5.38}$$

with $\mathbf{C}^{(\mathrm{j})} = \boldsymbol{u}_j \boldsymbol{v}_j^T$.

The NNDSVD initialization now uses a modification of this expansion: Let us assume that $\mathbf{X}$ is non-negative. One may generally write any given vector or matrix $\mathbf{A}$ as being a sum of its positive and negative sections, denoting these as $\mathbf{A}_+$ and $\mathbf{A}_-$ respectively. For any entry $a$ of $\mathbf{A}$ the entries of these sections $a_+$ and $a_-$ can be defined as

$$\begin{aligned} a_+ &= \begin{cases} a & \text{if } a \geq 0 \\ 0 & \text{otherwise} \end{cases} \\ a_- &= \begin{cases} -a & \text{if } a \leq 0 \\ 0 & \text{otherwise} \end{cases} . \end{aligned} \tag{5.39}$$

Thus $\mathbf{A}$ can be written as $\mathbf{A} = \mathbf{A}_+ - \mathbf{A}_-$, and if $\mathbf{A} \geq 0$ then $\mathbf{A}_- = 0$. Using this notation, one can rewrite the summands of 5.38 as

$$\begin{aligned} \mathbf{C}^{(\mathrm{j})} = \boldsymbol{u}_j \boldsymbol{v}_j^T &= \left(\boldsymbol{u}_{j+} - \boldsymbol{u}_{j-}\right)\left(\boldsymbol{v}_{j+} - \boldsymbol{v}_{j-}\right)^T \\ &= \left(\boldsymbol{u}_{j+}\boldsymbol{v}_{j+}^T + \boldsymbol{u}_{j-}\boldsymbol{v}_{j-}^T\right) - \left(\boldsymbol{u}_{j+}\boldsymbol{v}_{j-}^T + \boldsymbol{u}_{j-}\boldsymbol{v}_{j+}^T\right). \end{aligned} \tag{5.40}$$

Using the positive sections of the unit rank matrices $\mathbf{C}_+^{(\mathrm{j})} = \boldsymbol{u}_{j+}\boldsymbol{v}_{j+}^T + \boldsymbol{u}_{j-}\boldsymbol{v}_{j-}^T$, it can be shown that $rank(\mathbf{C}_+^{(\mathrm{j})}) \leq 2$ and hence $\mathbf{C}_+^{(\mathrm{j})}$ has no more than two singular vectors and two corresponding singular values, which will be denoted with $\mu_{j+}$ and $\mu_{j-}$ [151].

Now, taking a look at the normalized positive and negative sections $\hat{\boldsymbol{u}}_{j\pm} = \frac{\boldsymbol{u}_{j\pm}}{\|\boldsymbol{u}_{j\pm}\|}$ and $\hat{\boldsymbol{v}}_{j\pm} = \frac{\boldsymbol{v}_{j\pm}}{\|\boldsymbol{v}_{j\pm}\|}$ of $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$, one finds these singular values to be $\mu_{j\pm} = \|\boldsymbol{u}_{j\pm}\|\|\boldsymbol{v}_{j\pm}\|$. One may therefore write the unordered SVD decomposition of $\mathbf{C}_+^{(\mathrm{j})}$ as

$$\mathbf{C}_+^{(\mathrm{j})} = \mu_{j+}\hat{\boldsymbol{u}}_{j+}\hat{\boldsymbol{v}}_{j+}^T + \mu_{j-}\hat{\boldsymbol{u}}_{j-}\hat{\boldsymbol{v}}_{j-}^T. \tag{5.41}$$

As $\mathbf{C}_+^{(\mathbf{j})}$ is non-negative, the Perron-Frobenius theory states that its singular vectors are also non-negative [156].

One may now take the dominant singular triplet of $\mathbf{C}_+^{(\mathbf{j})}$ that is $\left(\mu_{j+}, \hat{\boldsymbol{u}}_{j+}, \hat{\boldsymbol{v}}_{j+}\right)$ if $\mu_{j+} = \max\left(\|\boldsymbol{u}_{j+}\|\|\boldsymbol{v}_{j+}\|, \|\boldsymbol{u}_{j-}\|\|\boldsymbol{v}_{j-}\|\right)$ and $\left(\mu_{j-}, \hat{\boldsymbol{u}}_{j-}, \hat{\boldsymbol{v}}_{j-}\right)$ otherwise and use it to initialize the $j$-th column of $\mathbf{W}$ as

$$\boldsymbol{w}_j = \sqrt{\mu_{j\pm}\sigma_j}\hat{\mathbf{u}}_{\mathbf{j}\pm} \tag{5.42}$$

and the $j$-th row of $\mathbf{H}$ as

$$\boldsymbol{h}_j^T = \sqrt{\mu_{j\pm}\sigma_j}\hat{\boldsymbol{v}}_{j\pm}^T, \tag{5.43}$$

with $\sigma_j$ being the singular value corresponding to $\mathbf{C}^{(\mathrm{j})}$ as in 5.38. Thus, when initializing $\mathbf{W}$ and $\mathbf{H}$ for a NMF with the inner dimension $r$ via NNDSVD, the dominant singular triplets $(\mu_{k\pm}, \hat{\boldsymbol{u}}_{k\pm}, \hat{\boldsymbol{v}}_{k\pm})$ of the $r$ leading positive sections $\mathbf{C}_+^{(\mathbf{k})}$ are used.

In contrast to this data-driven approach, a knowledge-based initialization can be promising in cases where background knowledge about the data samples is present. For example, Weiderer et al. use a knowledge-based approach to initialize the matrix factors in an NMF applied to the temperature time series of a metal casting process. In their work, an initialization based on solutions of the underlying physical thermodynamic heat transfer equation is used [124].

### 5.3.3. NMF applied to days of power load

In the application case of this thesis, the data matrix $\mathbf{V}$ consists of daily samples of power load. One of the incentives to use daily data was to reduce the dimensionality in which each day is presented. As the power usage data by nature is non-negative, applying Nonnegative Matrix Factorization came as a natural choice, as it does not impose any additional constraints, like uncorrelatedness or independence.

Using the NNDSVD initialization, and coordinate descent as an optimization technique for the Frobenius norm as loss function gives fairly sparse and interpretable results. Especially the non-negative coefficients allow seeing how much a certain component is present on a certain day. As we will see in section 6.4.4, the sparsity of the results also accounts for great interpretability of the received components or eigendays.

Having each daily sample as a row of the data matrix $\mathbf{V}$, the decomposition $\mathbf{V} \approx \mathbf{W} \cdot \mathbf{H}$ can be interpreted as follows: The data matrix $\mathbf{V} \in \mathbb{R}_+^{n\times m}$ consists of $n$ sampled days that are time series of measurements at $m$ discrete timestamps. The NMF decomposes $\mathbf{V}$ into $\mathbf{W}$ and $\mathbf{H}$, with $\mathbf{H} \in \mathbb{R}_+^{r\times m}$ storing $r$ components $\boldsymbol{h}_j$ in its rows that can be read as a kind of eigendays, each being a time series of length $m$. Accordingly, each row $\boldsymbol{w}_i$ of $\mathbf{W} \in \mathbb{R}_+^{n\times r}$ stores the $r$ coefficients that describe how much each eigenday is present in the original sample $\boldsymbol{v}_i$. Therefore, each sample $\boldsymbol{v}_i$ has a representation of

$$\boldsymbol{v}_i = \sum_{j=1}^{r} w_{ij} \cdot \boldsymbol{h}_j. \tag{5.44}$$

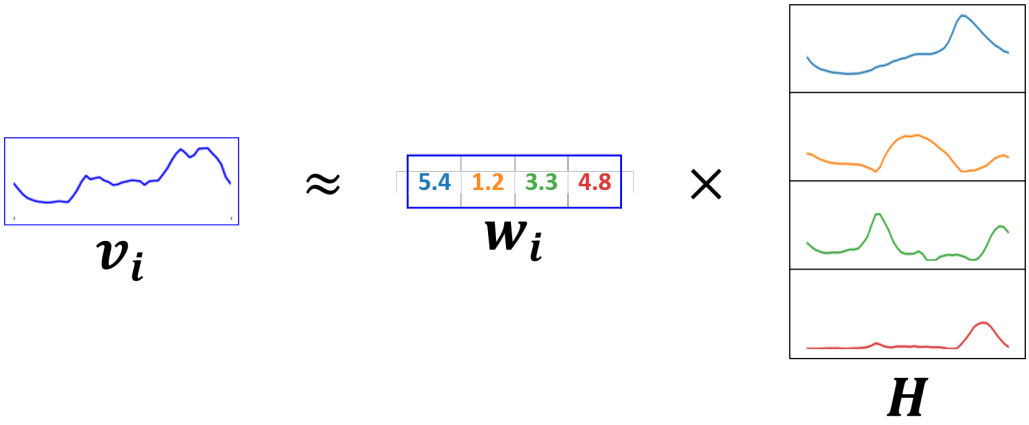



Figure 5.4.: How to understand the NMF decomposition of a single day. Each day $v_i$ is approximated by a weighted sum over $r$ components, stored in the matrix $\mathbf{H}$. The weights $w_i$ are individual for each day $i$, whereas the components are the same for all days.

This interpretation is also visualized in figure 5.4.

The decomposition could also be described such that the components $\mathbf{H}$ provide a summary of the timestamps across the samples, and the coefficients $\mathbf{W}$ provide a summary of the samples across the timestamps.

Of course, as NMF is only an approximation, the exactness of equation 5.44 highly depends on the choice of $r$ and the structure of the data in $\mathbf{V}$. The superposition of components $\boldsymbol{h}_j$ should therefore not be treated as an exact physical model describing the underlying power usage processes, but as a way to analyze the most prominent patterns created by the underlying processes.

## 5.4. Clustering

Clustering describes the task of identifying groups, or clusters, of data points in a multidimensional space. The goal is to partition the data set into $K$ clusters. Depending on the algorithm, the number $K$ can be given or determined in the process. One may think of a cluster as a group of data points that belong together, i.e. the inter-point distances among points of the same cluster are small, compared with the distances to points that do not belong to the same cluster [54].

One of the simplest and most common clustering techniques is the $K$-means algorithm, sometimes also referred to as Lloyd's algorithm. $K$-means finds an assignment of data points $\boldsymbol{x}_n$ to a predefined number of $K$ clusters, as well as a centroid vector $\boldsymbol{\mu}_k$ for each cluster, describing the mean of the samples in the cluster $k$, giving the algorithm its name. Note that the vectors $\boldsymbol{\mu}_k$ do not necessarily have to be elements of the data set.

The algorithm's objective is to minimize

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} \delta_k(n) \|\boldsymbol{x_n} - \boldsymbol{\mu_k}\|^2 \tag{5.45}$$

where $\delta_k(n)$ is 1 if $\boldsymbol{x_n}$ belongs to cluster $k$ and 0 otherwise. This is done in an iterative procedure. After an initialization of $\boldsymbol{\mu_k}$ a loop is performed, updating the assignments $\delta_k(n)$ and locations of centroids in turns, changing one while the other is kept constant. When determining the assignments, each $\boldsymbol{x_n}$ is simply assinged to the closest centroid $\boldsymbol{\mu_k}$. The optimization of the centroids with fixed $\delta_k(n)$ sets $\boldsymbol{\mu_k}$ to the mean of all data points $\boldsymbol{x_n}$ assigned to cluster $k$. This loop ends if the centroids stop moving or if a maximum number of iterations is exceeded [54, 157].

At this point, it should be mentioned that the Nonnegative Matrix Factorization can also be used as a clustering technique. The interpretation mostly used in this thesis focuses on $\mathbf{W}$, reading it as a feature matrix and describing each sample in terms of the modes in $\mathbf{H}$. In contrast, one may also lay the interpretation focus on $\mathbf{H}$ and interpret the NMF results by looking at the highest coefficients in each row of $\mathbf{W}$. One may assign each sample to a single mode of $\mathbf{H}$ that best fits the data. The data can thereby be divided into $r$ clusters, each corresponding to one mode that best fits the samples assigned to it, similar to how it is done with the Vector Quantization algorithm.

## 5.5. Regression models

The goal of regression is to predict the value of a continuous target variable $y_i$ or vector $\boldsymbol{y_i}$, given the value of an input vector $\boldsymbol{X_i}$. A regression model is trained on a training set of pairs $(\boldsymbol{X_i}, \boldsymbol{y_i})$ such that it is able to predict the value of $\boldsymbol{y_j}$ given an unseen input $\boldsymbol{X_j}$. This can be done in multiple ways with many different types of models, each with a different concept of how the outputs are created given the input vectors. Two simpler approaches, Ridge Regression and tree-based models, and more complex Neural Network architectures are presented in this section.

### 5.5.1. Ridge Regression

Linear Regression is a parametric approach that models the output with a function linear in its adjustable parameters. If the target variable is multidimensional, it is also referred to as multiple Linear Regression. Even though Linear Regression models, in practice, struggle with dealing with high dimensional input spaces, they are easy to interpret and have favorable analytical properties [54].

In its most basic form, a Linear Regression model is not only linear in its parameters but also in the input variables. In this case, the equation of a Linear Regression model can be written as

$$\boldsymbol{y_i} = \boldsymbol{X_i}\boldsymbol{\Omega} + \boldsymbol{\epsilon} \tag{5.46}$$

with $y_i$ denoting the dependent target variable vector, the explanatory input variables being written as $X_i$, the models parameter matrix denoted by $\mathbf{\Omega}$ and having an error term $\boldsymbol{\epsilon}$.

Given a training set, the model determines its parameters $\mathbf{\Omega}$ by the ordinary least squares method [54, 158]. Writing the training set as $(\mathbf{X}, \mathbf{Y})$ the estimated parameters are the result of

$$\hat{\mathbf{\Omega}} = \arg\min_{\mathbf{\Omega}} \|\mathbf{Y} - \mathbf{X}\mathbf{\Omega}\|_2^2. \tag{5.47}$$

Linear Regression models fitted with the ordinary least squares methods struggle when input variables are highly correlated, as is often the case in Machine Learning scenarios. In this case, it is useful to penalize large coefficients by introducing a regularization term into the optimization problem. Regularization limits the model complexity and allows for training on limited size data sets without severe overfitting [54].

One option of regularization is to add an *L*2-Regularization term to the cost function of the model, shrinking the parameter magnitudes. When adding such a term to a Linear Regression, the resulting model is also referred to as a Ridge Regression model, with its coefficients estimated as the result of

$$\hat{\mathbf{\Omega}} = \arg\min_{\mathbf{\Omega}} \left( \|\mathbf{Y} - \mathbf{X}\mathbf{\Omega}\|_2^2 + \lambda \|\mathbf{\Omega}\|_2^2 \right). \tag{5.48}$$

The regularization coefficient $\lambda$ controls the amount of parameter shrinking and the relative importance of the data dependent error $\|\mathbf{Y} - \mathbf{X}\mathbf{\Omega}\|_2^2$ and the regularization term $\|\mathbf{\Omega}\|_2^2$. A higher value for $\lambda$ encourages the weights to decay towards zero unless the data supports stronger values, making the model more robust to colinearity [54, 158].

### 5.5.2. Tree-based regression models

A different type of regression models are tree-based models. These models are built out of one or multiple decision trees, each splitting the data into multiple subsets by comparing feature values to learned thresholds. The value a tree outputs for a given sample depends on the subset this sample is assigned to. In the following part CART models, as well as bagging and boosting as ways to build ensembles of trees, are discussed. Finally, it is explained how bagging is used to build a Random Forest for regression.

In contrast to distance-based models, decision trees have the advantage that they are invariant to the scaling of features. The splits are found regardless of the feature magnitudes, making it unnecessary to perform a preprocessing step like scaling or normalizing the data. Decision trees are simple to understand and interpret and can easily be visualized. In addition, the cost of using a tree for predictions is computationally cheap, as the cost is logarithmic in the number of data points that are used to train the tree [159].

#### CART

The Classification and Regression Tree (CART), developed by Breiman in 1984, is the most basic tree structure and can either be used for classification or regression tasks, both

using the same basic principle [86]. A CART is built from nodes and edges. A node has two outgoing edges, leading to two more nodes, called the children. Each of these nodes is then either split further or represents a terminal node, a so-called leaf.

At the non-terminal nodes a split of the input data is performed, such that the samples are divided into two sub-sets. This split is done in an optimal way, maximizing the information gain based on a predefined split criterion (e.g. the gini impurity or the least-squares criterion) [86, 160]. For the use-case of regression, where the target is a continuous value, a common split criterion is minimizing the Mean Squared Error (MSE), the Poisson deviance or the Mean Absolute Error (MAE) [159].

The successive splitting at each node divides the provided data into disjunct chunks at each leaf. The decision if an additional split is performed at a node or if this node is to be used as a leaf depends on hyperparameters of the CART. These include a maximum depth, a minimum sample size $\min_{samples}$ in each leaf, and the maximal impurity $\max_Q$ allowed in each leaf [159].

After a CART model has seen all training data and learned how to optimally split it into the leaves, new data can be shown to the model. Based on the learned split rules, each sample is assigned to one of the leaves. For a classification task, the sample is then classified by majority voting, i.e. after a sample has been assigned to a leaf, it is classified as the class that the majority of training samples in its leaf has. For a regression task, the mean over the target values of all training samples in the leaf is output as the prediction for this sample.

Let $\boldsymbol{X}_i \in \mathbb{R}^m$, $i = 1, \ldots, n$ be input vectors with $m$ features and corresponding $l$-dimensional target values $\boldsymbol{y}_i \in \mathbb{R}^l$, $i = 1, \ldots, n$. We denote the data at node $\omega$ as $D_\omega$ with $|D_\omega|$ samples. A possible split $\theta_\omega = (j, t)$ consists of a feature $j \in \{1, \ldots, m\}$ and a threshold value $t \in \mathbb{R}$ resulting in a partition of $D_\omega$ into $D_\omega^{left}(\theta_\omega)$ and $D_\omega^{right}(\theta_\omega)$ such that

$$\begin{aligned} D_\omega^{left}(\theta_\omega) &= \{(\boldsymbol{X}, \boldsymbol{y}) \in D_\omega | x_j \leq t\}, \\ D_\omega^{right}(\theta_\omega) &= \{(\boldsymbol{X}, \boldsymbol{y}) \in D_\omega | x_j > t\}. \end{aligned} \tag{5.49}$$

The quality of a split $\theta_\omega$ is evaluated by using an impurity or loss function $Q$. For classification problems, common choices for $Q$ are the gini impurity, the entropy or the number of misclassifications [159, 160]. In the regression case, the MSE

$$Q(D_\omega) = \frac{1}{|D_\omega|} \sum_{\boldsymbol{y} \in D_\omega} (\boldsymbol{y} - \bar{\boldsymbol{y}}_\omega)^2 \tag{5.50}$$

with

$$\bar{\boldsymbol{y}}_\omega = \frac{1}{|D_\omega|} \sum_{\boldsymbol{y} \in D_\omega} \boldsymbol{y} \tag{5.51}$$

is used.

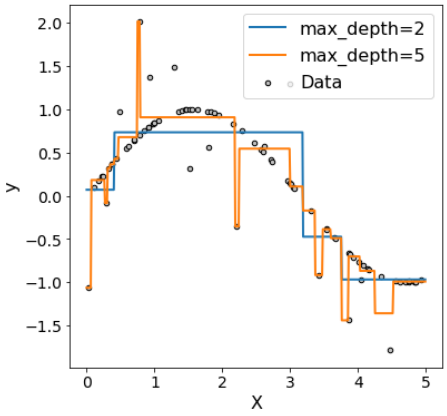



Figure 5.5.: Fitting a noisy sine curve using regression trees of two different depths (example taken from [159]). The regression tree with a maximal depth of 5 also fits some outliers caused by noise.

With the chosen $Q$, a split $\theta_\omega$ is evaluated by computing the rating

$$R(D_\omega, \theta_\omega) = \frac{|D_\omega^{left}(\theta_\omega)|}{|D_\omega|} Q(D_\omega^{left}(\theta_\omega)) + \frac{|D_\omega^{right}(\theta_\omega)|}{|D_\omega|} Q(D_\omega^{right}(\theta_\omega)) \tag{5.52}$$

and an optimal split

$$\theta_\omega^* = \text{argmin}_{\theta_\omega} R(D_\omega, \theta_\omega) \tag{5.53}$$

is chosen and performed. This splitting procedure is then repeated at the children of node $\omega$ with the subsets $D_\omega^{left}(\theta_\omega^*)$ and $D_\omega^{right}(\theta_\omega^*)$ until the maximum depth of the tree is reached, $|D_\omega| < \min_{samples}$ or $Q(D_\omega) \leq \max_Q$ [159].

Restricting the depth is necessary to prevent overfitting. Figure 5.5 shows a noisy sine curve that is fitted with two regression trees. Once with a maximum depth of 2 and once with a maximum tree depth of 5. It can be seen that larger depth lets the model also fit outliers caused by noise.

Comparing a CART model to a Linear Regression, one main difference is recognizable: Whereas a Linear Regression can extrapolate its function into unseen regions, a CART only has a finite number of responses that can be output, leading to a locally constant output function. This leads to the fact that a CART model may not extrapolate beyond the range of values seen in the training data.

### Bagging and boosting

Decision trees are built in a greedy manner, i.e. splits of higher levels are not reviewed and each split is done in only a locally optimal way. Hence, an individual tree is prone to variations in the data and tends to overfit. The variance among multiple trees can thus be large. For this reason, decision trees also belong to the category of so-called weak learners. Nevertheless, combining multiple weak learners can be a useful technique, and

ensembles of decision trees have proven themselves to be robust models with a lot of success in Machine Learning competitions [161, 87, 88].

For decision trees, two main ways of combining single trees in an ensemble of trees are commonly used: Bagging and Boosting.

Bagging, introduced by Breiman in 1996, describes the process of building multiple independent predictors and averaging their outputs [162]. When applying bagging to decision trees, the building process of these trees is usually combined with bootstrapping the training data and, thus, leads to a training of trees with different initial conditions. Each tree is shown only a fraction of the training data. A fraction of samples is drawn randomly with replacement and the tree is trained on this bootstrap of the data. Additionally, a bootstrap of features is drawn, meaning that only a randomly sampled number of features of the input data is presented to each tree. On inference, the results of all trees are collected and a final single output is given. For classification, the target class is determined by majority voting, whereas for regression the outputs of all trees are averaged.

Bagging and bootstrapping are a strong combination, making the trees de-correlated and, thus, decreasing the variance and making the entire method more robust to outliers and noise [161].

Boosting is a second way of building ensembles of weak learners, often used with decision trees. In contrast to the bagging process, the trees in the ensemble are not built independently, but successively; with each tree being trained on the residuals of the previous ensemble and then being added to this ensemble. The final model is an additive combination of these iteratively built decision trees [87]. This concept of building the trees is also visualized in figure 5.6.

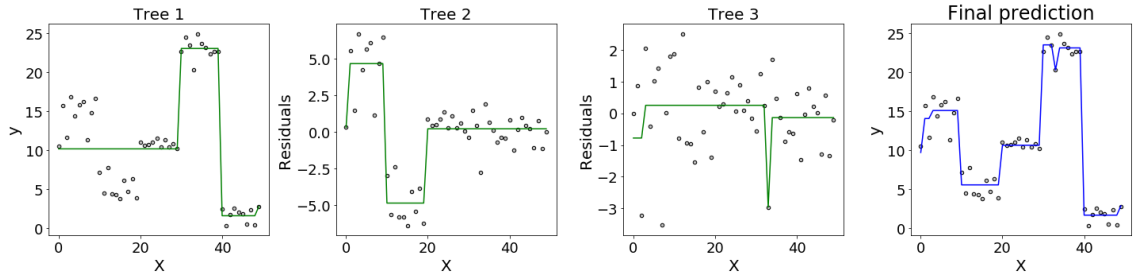



Figure 5.6.: Concept of boosted trees: The first tree is fitted on the original data. All successive trees are fitted on the residuals of the ensemble of previously built trees.

Two of the most popular used boosting algorithms are AdaBoost and Gradient Boosting [163, 164].

#### Random Forest

A Random Forest (RF), introduced by Breiman in 2001, is an ensemble model for both regression and classification problems, leveraging bagging and bootstrapping for CARTs with one extension: Bootstrapping not only happens at the beginning, when showing the data to each tree but a feature bootstrapping is performed at each node [161]. Thus, each tree in the ensemble is built on randomly sampled data drawn with replacement. Each time a node of the tree chooses an optimal split, some of the potentially available features are censored and the split can only be calculated on the remaining features.

The two most important hyperparameters for the RF algorithm are the number of trees to be grown and the number of features shown to each node [165]. Adding more trees does not lead to overfitting, but only towards a limiting value of the generalization error. This error is estimated by an out-of-bag error, i.e. by calculating the error for training points not contained in the bootstrap training sets [161]. However, adding many trees increases the computation time and a tradeoff must be made. For the number of features to be considered for each split, Hastie et al. suggest using one third of the available features when using the RF for regression tasks [166].

It is worth noting that while a single CART model describes a piecewise constant function, an ensemble of CART models can give a smoother result.

### 5.5.3. Neural Networks

The rise of Neural Networks (NN) in recent decades has led to a new spring in AI research. The challenge to AI had been to solve tasks easily performed by people but hard to describe formally and, thus, challenging to solve for machines [52]. New model architectures and the increase of available computing power led to a breakthrough in solving these tasks [52].

Especially in the past few years, the development of complex Deep Neural Network (DNN) models led to astonishing results and breakthrough technology in many fields. Among others, state-of-the-art performances in visual object recognition [56, 57, 58], image segmentation [59], speech recognition [60] and machine translation [61, 62, 41] could be achieved by using DNN models and their ability to discover intricate structure in high-dimensional data [55].

The strength of NN models in general and Deep Learning (DL) models particularly lies in the stacking of non-linear submodules, so-called layers, that are transforming the given data into a more abstract level. By stacking many of these layers, very complex functions can be learned. In fact, it can be proven that with a Fully Connected Network, any Riemann integrable function can be approximated arbitrarily accurately [55, 167].

This section of the thesis focuses on one of the most commonly used Neural Network archetypes, i.e. Fully Connected Networks, before explaining the architecture of N-BEATS, a NN model explicitly designed for Time Series Forecasting.

### Fully Connected Networks

As with all NN models, Fully Connected Networks are built of successive layers. The first one is called the input layer and the last one is referred to as the output layer. All layers in between are just referred to as hidden layers, as they are not directly exposed to the outside of the model.

A Fully Connected Network or Multilayer Perceptron (MLP) is comprised of many layers that essentially have the same structure. Each layer is built from multiple units, called neurons. Every single neuron in the network receives the results $\boldsymbol{x}$ of all neurons from the previous layer as input (the input layer takes the data itself as input), computes a weighted sum of these results, and adds a bias $b$. This biased weighted sum is then passed through a non-linear activation function $\sigma$ and passed on to the units in the next layer. The last computations happen in the neurons of the output layer. Their results are then presented to the user as the final output and can be interpreted depending on the task for which the network is used. The weights $\boldsymbol{\omega}$ used for the summation are individual to each neuron, whereas the used activation function is the same for all units. Hence, in each neuron a computation

$$y = \sigma\left(\boldsymbol{\omega}^T \boldsymbol{x} + b\right) \tag{5.54}$$

is performed.

Typical activation functions $\sigma : \mathbb{R} \to \mathbb{R}$ are the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$, the hyperbolic tangent $\sigma(x) = \tanh(x)$ and the Rectified Linear Unit (ReLU) $\sigma(x) = \max(x, 0)$, with most newer architectures relying on the use of ReLU and its variants.

The power of Neural Networks lies in the fact that the weights $\boldsymbol{\omega}$ and biases $b$ of all neurons are adjustable and trainable. The training of these weights and, thus, the training of the entire network is based on the idea of backpropagating errors [168]. Depending on the values of the weights and biases (the network parameters), the output of the network will be more or less close to the desired output. This difference is determined by an error function. The gradient of this error function is calculated with respect to all weights and biases in the neurons. To minimize the output error, the weights and biases are adjusted a small step in the opposite direction of this gradient. With these new values for the network parameters, one may again calculate the output and a new gradient. This procedure is repeated until a minimum of the error function is reached and the network gives a desirable output [167].

The landscape of error functions can be highly complex. To be able to minimize these, advanced variants of the traditional back-propagation algorithm are used in practice, including but not limited to RMSprop and Adam [169, 170].

### N-BEATS

After looking at the most basic NN architecture, we now want to take a look at Neural Basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS). The

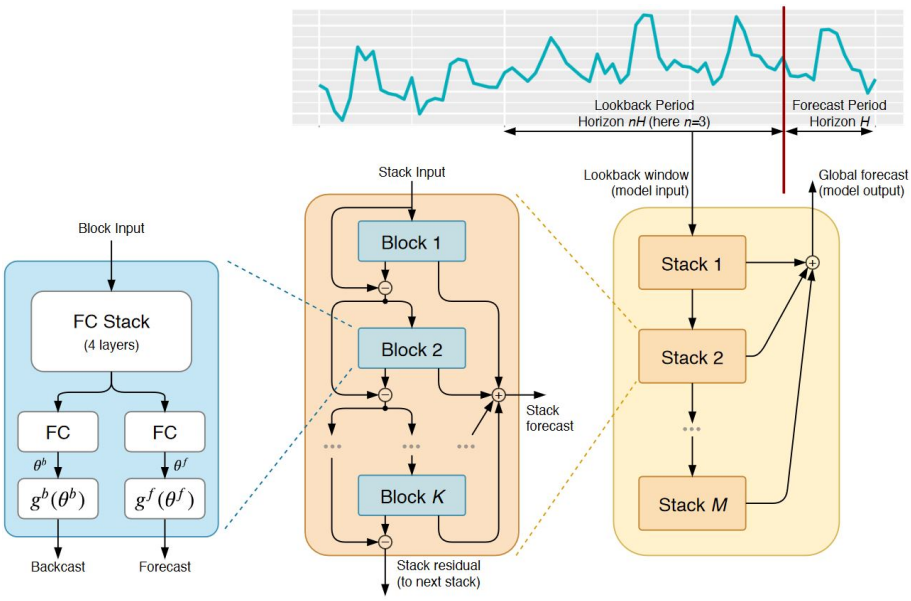



Figure 5.7.: N-BEATS architecture. Image taken from original paper [68]. The model is built from stacks of blocks. These basic building blocks each output a backcast and a forecast. In each stack, the blocks are connected via a subtractive residual connection. The output of each stack is the sum of its blocks' forecasts. The final model output is a sum of all stack outputs.

N-BEATS model, developed by Oreshkin et al. in 2019, got a lot of attention by beating all models participating in the M4 competition in forecast accuracy [68]. This success can be seen as a breakthrough in Time Series Forecasting, as N-BEATS has a purely DL-based architecture, thus proving that a pure ML model can outperform statistical models and their combinations with ML models in univariate TSF tasks.

In principle, the N-BEATS architecture is built out of multiple blocks, connected in a doubly residual topology. The architecture is shown in figure 5.7, taken from the original paper [68]. The basic building block of N-BEATS is a multi-layer Fully Connected Network with ReLU activation functions with one input and two different outputs: A forecast over the given horizon and a backcast, which is the block's estimate of its given input.

Multiple blocks are organized in stacks using a doubly residual stacking, with the first residual branch running over the forecasts of all blocks in a stack and the second branch running over all backcasts. The first residual branch aggregates the forecasts of all blocks in all stacks to receive a final forecast $\hat{y}$ to be output by the model. Meanwhile, the second residual branch operating on the backcasts has a different task: The backcast of a block $l-1$ is subtracted from the signal before using this difference as the input signal for the next block $l$. This way the blocks can remove the portion of the original signal that they can approximate well and leave the following blocks with an input that

has already been thinned out. Hence, the downstream blocks can focus on the residuals that were not explained by the previous blocks [68].

In other words, N-BEATS uses backcasts to allow for some kind of boosting among the individual blocks while only using DL principles.

## 5.6. Error measures for evaluation

After having trained multiple models, their performance is compared by measuring their forecast accuracy. This can be done with multiple error measures, which are presented in this section.

Note that in this thesis, the terms error function and error measure are used differently. The error function, or cost function, refers to the term that is minimized when training a model. It does not necessarily have to be easy to read or interpret but should bring desirable mathematical properties, depending on the kind of model that is fitted. The term error measure, on the other hand, is not used for model training, but only used for evaluation reasons. It can be used to monitor the progress in training but does not affect the training process itself. Error measures are usually easy to read and interpret. E.g. for a classification task, a typical error function minimized in training would be the KL divergence, while the model's performance can be measured by using the percentage of correctly classified samples as an error measure.

The used error functions depend on the type of model that is to be fitted. When evaluating and comparing the performance of different forecasting models, multiple error measures can come to use, independent of the model by which the prediction was made. With each measure, the prediction $\hat{\boldsymbol{y}}$ with entries $\hat{y}_t$ is compared to the target series $\boldsymbol{y}$ with entries $y_t$, both having a length of $n$.

The most prominent error measure to evaluate the deviation of the prediction $\hat{\boldsymbol{y}}$ from the target $\boldsymbol{y}$ is the Mean Squared Error (MSE)

$$\mathrm{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n} \sum_t (\hat{y}_t - y_t)^2 . \tag{5.55}$$

One interpretation issue with MSE is that squaring the differences between $\hat{y}_t$ and $y_t$ also squares the unit. E.g. if the target $\boldsymbol{y}$ carries a unit of kWh, then the MSE is given in $(\mathrm{kWh})^2$. Therefore, the Root Mean Squared Error (RMSE)

$$\mathrm{RMSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sqrt{\frac{1}{n} \sum_t (\hat{y}_t - y_t)^2} \tag{5.56}$$

is a prominent alternative, as taking the root leads to RMSE carrying the same unit as the original time series. The Mean Absolute Error (MAE) is similarly popular, due to its simplicity. It can be calculated as

$$\mathrm{MAE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n} \sum_t |\hat{y}_t - y_t| . \tag{5.57}$$

The main drawback with all three presented error measures is that their values depend on the scale of the time series, making it unfeasible for comparing model performance across series with different scales. Additionally, all these absolute error measures are strongly affected by outliers [131].

One way of calculating a scale invariant error between forecast and target is the Mean Absolute Percentage Error (MAPE)

$$\mathrm{MAPE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n} \sum_t \left| \frac{\hat{y}_t - y_t}{y_t} \right| \cdot 100\%. \tag{5.58}$$

MAPE has the advantage that its values can be compared across multiple time series, however, its calculation also comes with some shortcomings. Mathematically, a division by zero is not possible, however, in practice, the number of instances where $y_t = 0$ can be neglected and, thus, a simple filtering for these cases solves this problem. A bigger issue is that the error calculated with MAPE is unsymmetrical, i.e. the error contribution of a time step $t$ is different, depending on whether $\hat{y}_t$ is bigger or smaller than $y_t$ [131].

One potential drawback of all presented error measures comes with peaky forecasts, which are not unusual in practice. The problem is that these can be penalized twice. First, at the instance at which an actual peak is missed, the second time at the instance where the peak is mistakenly forecasted [34].

In this thesis, however, mainly the conventional error measures RMSE and MAPE are used, for they are reasonably simple and transparent and commonly used in electric load forecasting.

Additionally, one more measure is computed that is interesting to monitor when forecasting models are used in a scheduling task, as it is the intention of the models in this thesis: the Sum Of Differences (SOD)

$$\mathrm{SOD}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_t (\hat{y}_t - y_t), \tag{5.59}$$

giving information about the general tendency of a prediction and whether the model over- or underestimates the target time series.

# 6. Used Data

> If we have data, let's look at data. If all we have are opinions, let's go with mine.
>
> Jim Barksdale

This chapter describes the data used in this thesis. This includes the main data source from the Irish Smart Meter Trials [132], corresponding weather data, and additional information like the dates of holidays. First, each of these data sources is described separately and the preprocessings used are laid out. Afterward, the entire data situation is analyzed, including a look at how external data affects the smart meter data.

## 6.1. ISSDA smart meter data

The main data set used in this thesis is the electricity consumption data of the Irish Commission for Energy Regulation (Irish CER) Smart Metering Project, which is distributed by the Irish Social Science Data Archive (ISSDA) [132]. In this customer behavior trial, over 5000 Irish households and businesses had an electricity smart meter installed in their homes/premises. These smart meters were monitored by the Irish CER, the regulator for the electricity and natural gas sectors in Ireland. The Irish CER works within the framework of national and EU energy policy, which aim to create a single European electricity market that best meets the needs of Europe's energy consumers [171].

In this section, first, the raw data set is described, before the general data cleansing, the slicing of the time series to days, and the handling of the time zone are presented.

### 6.1.1. The data set

The electrical power demand was measured at 30-minute intervals over a period of 1.5 years, lasting from summer 2009 to the end of 2010. Each measurement describes the power demand of a single household in the preceding half hour. In addition to this data about the energy consumption, two surveys, before trials and afterward, were performed for the residential households. This way, additional data was collected, including information about the external circumstances and personal situation of the residents, the number of inhabitants and number of children living in each household, how the inhabitants perceive their energy usage and heating behavior, and information

about the existence and amount of usage of different electrical devices. Different surveys were done for the participating businesses.

The Smart Metering Project was initiated to assess the performance of Smart Meters and whether they have an impact on the energy usage of consumers. It was performed by a collaboration of Irish participants, including network operators and energy suppliers. The main goal of this research project was to help to establish information on how smart metering shapes energy usage behaviors, in order to analyze the cost-benefit analysis for a national rollout. For this, different interventions and incentives for different energy usage were performed for groups of households [171, 132].

Even though this was the original purpose of the project, the customer behavior trial allows using the huge amount of collected data in many ways. The detailed smart metering data was anonymized and published together with the results of the survey. As no personal or confidential information of the participants is contained in the data set, it is ideal for facilitating further research. However, to not give away information implicitly, no information is given about where in Ireland the participants' houses/premises are located. This is something that must be kept in mind when considering additional information, such as weather data.

### 6.1.2. General data cleansing

For this thesis, only a part of the Irish CER Smart Metering Project data set is needed. The focus is laid on electricity usage that is relevant for the case of distributed generation. Therefore, to simulate the usage of a neighborhood or a small city district, the usage of 350 residential households is aggregated. In the Smart Meter Project, the participating households were assigned to groups, each getting a special tariff. To not be affected by such interventions, only households that were part of the control group in the Smart Metering Project are used. As no information about the geographical closeness of the households is available, the residential households having the 350 lowest IDs of all residential control group households in the data set are chosen. The used IDs can be found in the appendix A.4.

As data of these residential households is complete and does not carry any $NaNs$, no missing values have to be taken care of. After summing the power demand of these 350 households, the aggregated power usage was collected to receive the single time series that is plotted in figure 6.1.

### 6.1.3. Slicing time series to days

To compare and later predict entire days at once, it makes sense to not work with a single time series but instead look at it as a collection of days. The time series is divided into daily slices each containing 48 measurements for times 00:00, 00:30, 01:00, ... 23:30. Afterward, these slices are stacked into a matrix. This process is also illustrated in figure 6.2.

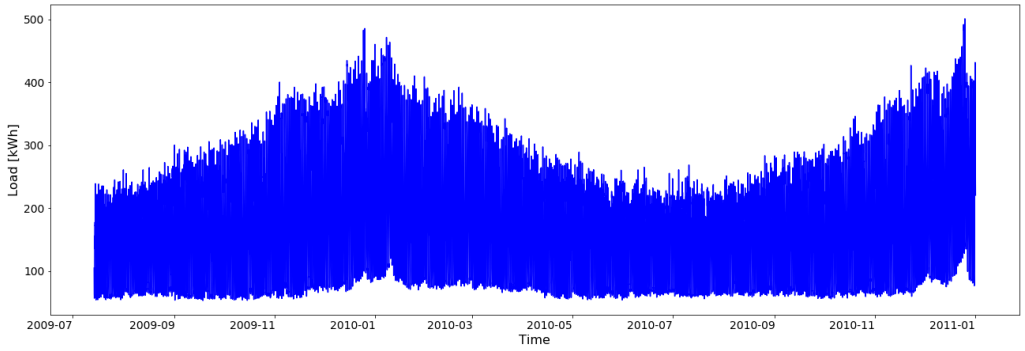



Figure 6.1.: Aggregated load demand of 350 residential households as a single time series from July 2009 until December 2010.

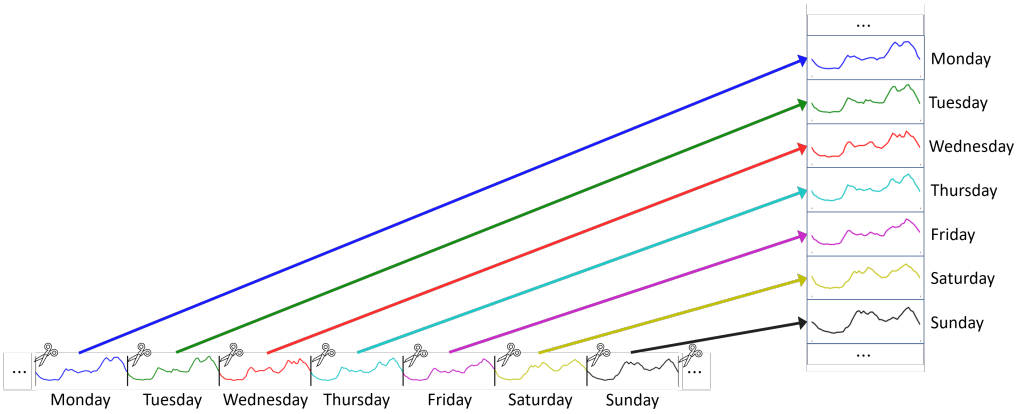



Figure 6.2.: Slicing time series data into daily pieces and stacking them into a matrix.

From now on, such a stacked matrix will be referred to as the data matrix **V**. In **V**, each row $v_i$ describes the aggregated energy demand of 350 residential households on a certain 24 hour day $i$ with a time series of 48 load values measured in 30-minute intervals.

### 6.1.4. Handling the timezone

Before being able to slice the time series to a matrix of daily slices, the local timezone in which the data was recorded, i.e. the timezone of "Europe/Dublin", needs to be handled. The issue with this timezone is that it uses Daylight Saving Time (DST). This means that from October to March, when it is standard time, Ireland is in the timezone of Coordinated Universal Time (UTC). Whereas from March to October, when it is DST, Ireland is in the timezone of UTC + 01:00h. This needs to be accounted for when handling the data.

The question of how to handle timezones is a relevant and often discussed topic in web development, as internet services have to be accessible from all over the globe and therefore from multiple timezones and the services must be able to correctly process the given information at all times. This is especially difficult when it comes to services like calenders that also handle timestamps of future events. Thus, in the development of web services, correctly handling the timezone is a well-known problem. However, in the literature about energy forecasting and time series-related Machine Learning in general, where DST is something that must be accounted for, this topic, as far as known, has not been handled yet.

A detailed discussion of the advantages and disadvantages of storing timestamps in UTC in a web development context is done by Skeet in his blogpost [172].

Generally, there are two options when working with time series whose timestamps carry a timezone that cannot be neglected. One can either convert the entire time series to UTC time or use the currently active local time and drop the timezone information. Both options have their advantages and disadvantages and will be discussed in the following part.

#### UTC conversion

The major advantage of converting the time series to UTC is that one no longer has to worry about DST and all days have 24 hours, as is to be expected. However, depending on whether DST is active or not, the events happening at the same UTC time must be interpreted differently in terms of whether they happen early or late, as consumers do not plan their day according to UTC, but to their local time. This means that after converting to UTC, values at the same UTC time on different days are not necessarily easy to compare. If, for example, a person always gets up at 7am and starts their day, this usually causes an increased energy demand compared to the nighttime. On a winter day, this 7am behavior will be displayed at 7am UTC. But on a summer day, where DST is active, this behavior is at 7am UTC + 01:00h. When converting this to UTC the increasing energy demand will appear at 6am UTC, making it difficult to compare the two days'

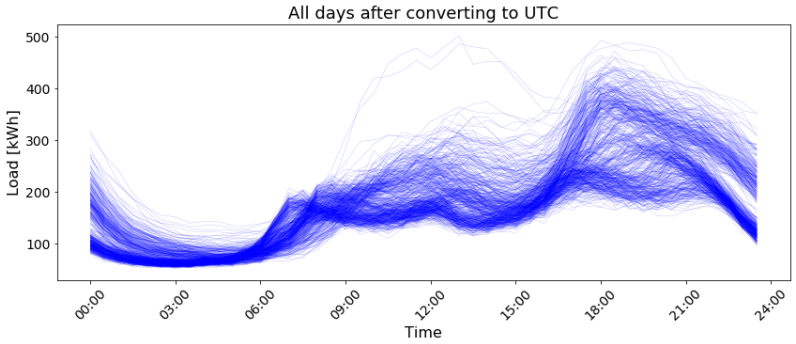



Figure 6.3.: All days of the data set, with a conversion to UTC before slicing the data. Two major morning peaks are present because of the Daylight Saving Time.

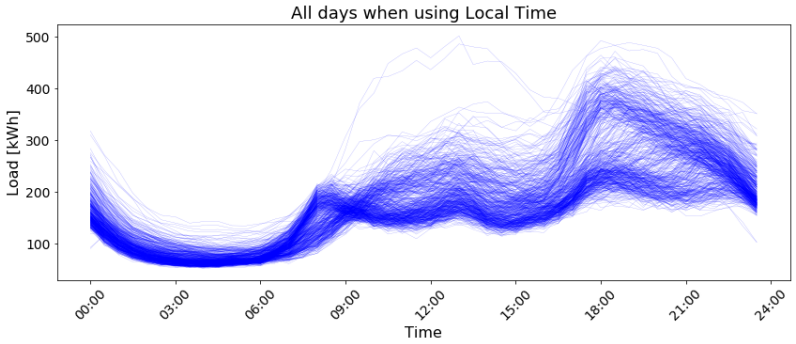



Figure 6.4.: All days of the data set, with the local time kept before slicing the data. Only one major morning peak is present despite the Daylight Saving Time.

behavior. In figure 6.3, each day of the UTC converted data is plotted as a single line. As a result of the UTC conversion, the plot shows two major morning peaks. The later one is caused by the morning peak in standard time, the earlier one by the morning peak with active DST during the summertime.

### Local time

Instead of converting the entire time series to UTC, the second option to handle the timezone is to drop the timezone information of the index timestamps and only keep the time information about the currently active local time. In consequence, on the days on which DST is turned on/off, the days do not have 24 hours. In spring, when the timezone is switched from standard time to DST, the clocks are set forward one hour at 2am. This results in a day that only has 23 hours. In autumn, when the switch from DST to standard time is performed at 3am, the clocks are set back one hour, resulting in a day with 25 hours. This can cause problems with models handling days as inputs, as usually a certain input length is expected. If these irregularities are to be avoided, a special treatment for days not having the standard length of 24 hours is to be set up.

However, keeping the local time has one main advantage over the UTC conversion: Days throughout the year can easily be compared. As people usually show behavior that complies with local time instead of UTC time, a 7am value in summer can directly be compared to a 7am value in winter. This can be validated in figure 6.4 for which the days were sliced when the local time was kept. Because of this, only one major morning peak is present in the shown data.

### Choosing a timezone

In our case, the data is the power demand of users who plan their day according to local time and the days show patterns that are somewhat constant throughout the entire year. An example of this is the aforementioned increased demand peak in the morning, starting to rise at about 7am. If the data were transferred to UTC, this morning peak would arise an hour earlier in DST than in standard time. Thus, any model receiving input data in UTC time would also output their predictions in UTC and therefore need to account for the time shift in spring and autumn. This is an additional feature the model must account for. Staying in local time removes this difficulty and, hence, makes it easier for the model, as it does not have to incorporate this shift in time.

As the models should be kept slim, the original time series is not converted to UTC. Instead, the timezone information is dropped, the data is kept in the local time and the days with 23 or 25 hours are taken care of.

In spring, when DST is turned on and the day of switching the timezone only has 23 hours, an artificial "missing" hour is filled in by repeating the last proper value (forward filling). In autumn, when DST is changed to standard time and the day of switching has 25 hours, the "double" hour is removed, by using the first of the two hours from 2am to 3am. There are two occurrences of DST being turned off in autumn but only one event of DST being turned on in spring in the used data. Thus, two hours are removed but only one hour is added. Hence, the converted time series is one hour shorter than the original raw version.

This preprocessing leaves us with a time series in which all appearing days have 24 hours and the values are referenced according to the currently active local timezone.

### Dimensionality reduction for timezone data

We have already seen that the choice of whether to convert the original time series to UTC or to stay in local time and handle the days with 23 or 25 hours has an impact on how easily days in summer and winter can be compared to each other. Using UTC-converted data can even take away some of the potential capacity of slim models. To show this, the data is sliced into days that are stacked as rows of a data matrix: Once after converting to UTC to receive the matrix $\mathbf{V_{UTC}}$ and once keeping the local time but accounting for the 23 hour and 25 hour days to receive $\mathbf{V_{loc}}$. It is then compared how these two matrices behave under decomposition. Decomposing $\mathbf{V_{UTC}}$ with an Nonnegative Matrix Factorization we receive the component matrix $\mathbf{H_{UTC}}$ and the coefficient matrix $\mathbf{W_{UTC}}$.

When applying a NMF with a rank of $r = 4$, looking at figure 6.5, we see that two of the found components (component 2 and component 3) have a morning peak that is approximately one hour shifted. Looking at the coefficient magnitudes over time, we see that the corresponding coefficient 3 of the later peak increases when DST is turned off in autumn. Contrary to this, the magnitude of coefficient 2, corresponding to the component carrying the earlier morning peak, suddenly drops. When DST is turned on in spring, the reverse phenomenon appears.

In contrast, when applying a NMF on $\mathbf{V_{loc}}$, also with rank $r = 4$ to receive $\mathbf{H_{loc}}$ and $\mathbf{W_{loc}}$, the components shown in figure 6.6 all show different shapes, as do the time series of the corresponding coefficients.

It can be concluded that, in the UTC case, the NMF model must account for the existence of DST in the data with part of a component that might otherwise have been used for a different pattern present in the data and, hence, is stripped of some of its potential.

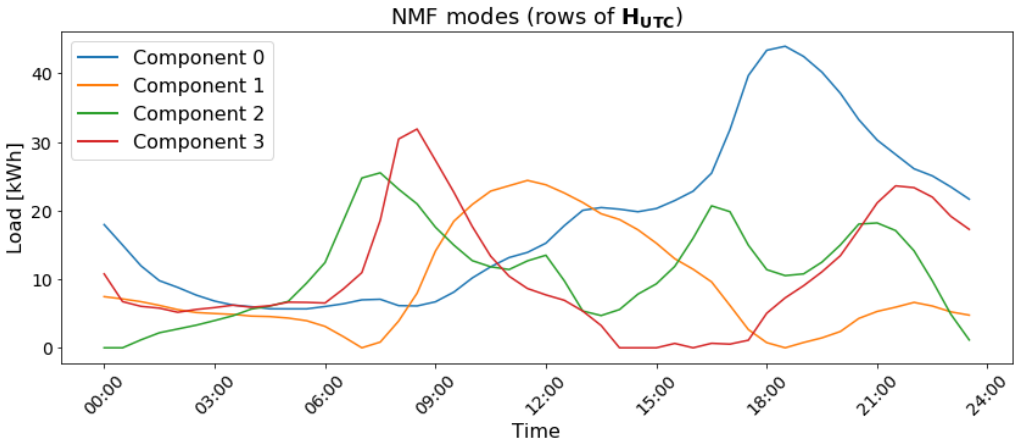
## 6.2. Weather data by the Met Éireann

As local weather patterns can have a significant effect on energy usage and knowledge about it can improve load forecasting, weather data associated with the original data set is needed [173]. For this we use weather data of the Met Éireann, the national meteorological service in Ireland [174]. In addition to current weather data, the Met Éireann also provides past data over several years and measurements from multiple weather stations across Ireland on their website `www.met.ie`. The data is provided under a Creative Commons BY 4.0 license[1] with a Copyright by Met Éireann. Met Éireann does not accept any liability whatsoever for any error or omission in the data, their availability, or any loss or damage arising from their use. The material has been modified from the original, as is described in the following paragraph.

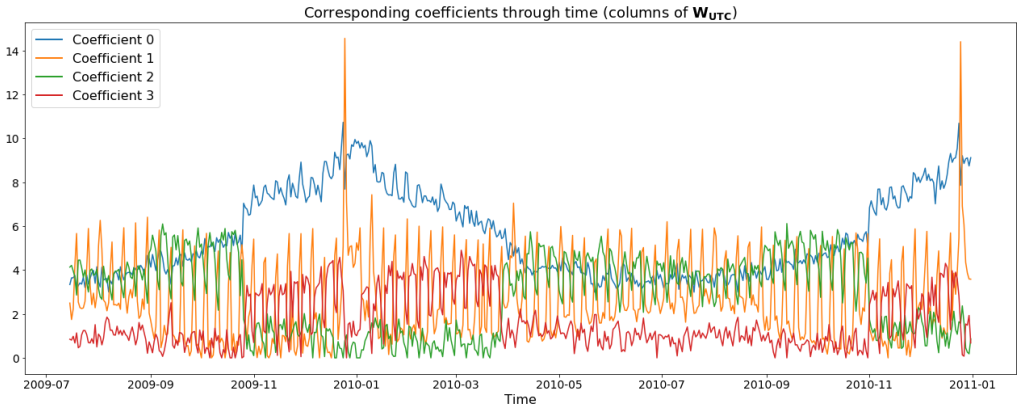
As the exact geographical location of the residential households providing the power usage data is unknown, weather data from all weather stations across Ireland was downloaded, provided these stations had data for the entire time range of the power usage data set [174]. The list of used weather stations can be found in the appendix A.5. The data for each of these weather stations provides data measured in 1-hour intervals, including information about the air temperature, precipitation amount and relative humidity. Each of these time series was resampled to 30-minute intervals by simply repeating each value once.

Of all these stations, the provided time series were aligned with the power usage data and cut to receive time series of the same length and with measurements assigned to the same times as the power usage time series. As the last step, to account for the unknown geographical location of the residential households, a mean was taken over the data of all weather stations, to receive a single mean time series for each kind of weather information, i.e. air temperature, precipitation amount and humidity

[1] `https://creativecommons.org/licenses/by/4.0/`

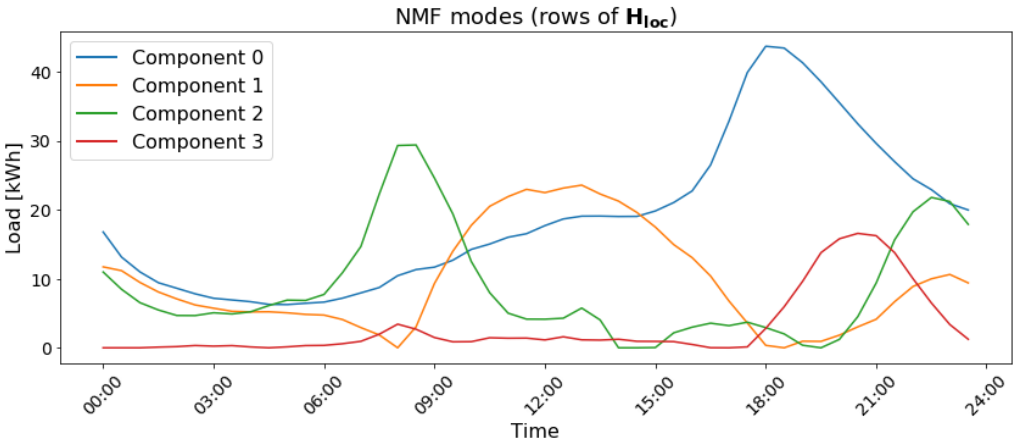



(a) Matrix $\mathbf{H_{UTC}}$ of the $\mathbf{V_{UTC}}$ NMF decomp.

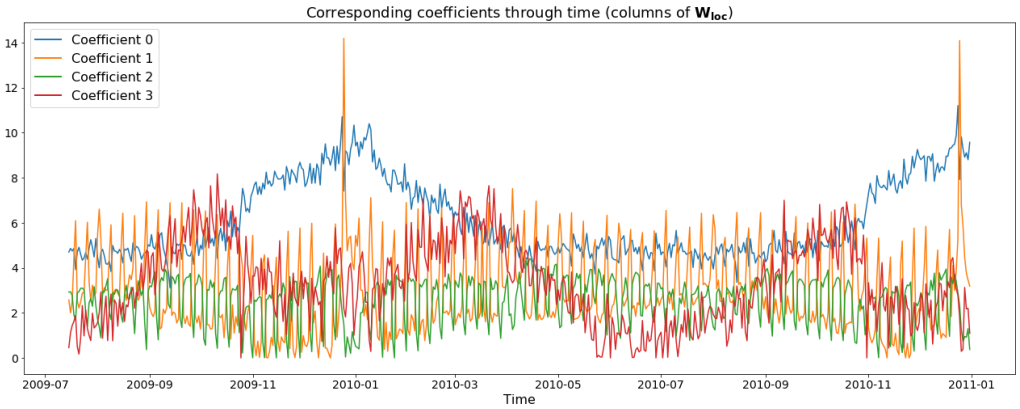



(b) Matrix $\mathbf{W_{UTC}}$ of the $\mathbf{V_{UTC}}$ NMF decomp.

Figure 6.5.: Visualization of the UTC converted data matrix $\mathbf{V_{UTC}}$ decomposed by a NMF with $r = 4$. Components 2 and 3 both show morning peaks that are one hour apart. The corresponding coefficients drastically change on DST changes.




(a) Matrix $\mathbf{H_{loc}}$ of the $\mathbf{V_{loc}}$ NMF decomp.




(b) Matrix $\mathbf{W_{loc}}$ of the $\mathbf{V_{loc}}$ NMF decomp.

Figure 6.6.: Visualization of the local time data matrix $\mathbf{V_{loc}}$ decomposed by a NMF with $r = 4$. All components have different shapes and the coefficient time series do not react drastically on DST changes.

## 6.3. Bank holidays

In addition to the external weather data, date information is used. This ranges from simple features, such as whether a certain day is a working day or a weekend day, to whether a bank holiday occurred. To retrieve information about Irish bank holidays, the Python package `holidays` was used [175].

One of the most important reasons to include holidays in the data set is to find out if the days that are to be predicted are working days. For this analysis, a special consideration has to be made. In Ireland, if a public holiday falls on a Saturday or Sunday, or possibly even coincides with another public holiday, a day off work is added on the next following weekday [176]. These days off are also referred to as observed holidays. In the preprocessing, a boolean value `is_holiday` for each day is set to `True` for public holidays as well as observed holidays.

## 6.4. Analysis of the data situation

In the following section, the full present data situation is described in more detail. This includes an analysis of the daily load data and the influence of the external data sources, as well as a clustering of the daily data. Finally, the components and coefficients retrieved from a NMF decomposition are analyzed.

### 6.4.1. Analysis of daily data

First of all, it is worth noting that the Irish Social Science Data Archive (ISSDA) provided a data set with the load demand of thousands of individual households, each over the course of about one and a half years. These users have different patterns in their load profiles, with partly different characteristics [132]. As an example, figure 6.7 shows three residential households with different kinds of load profiles.

Aggregating the usage of 350 households provides a time series with a more general usage pattern and, therefore, a smoother curve when plotted. Figure 6.8 shows the summed up data over the same week, previously used for the individual time series in figure 6.7.

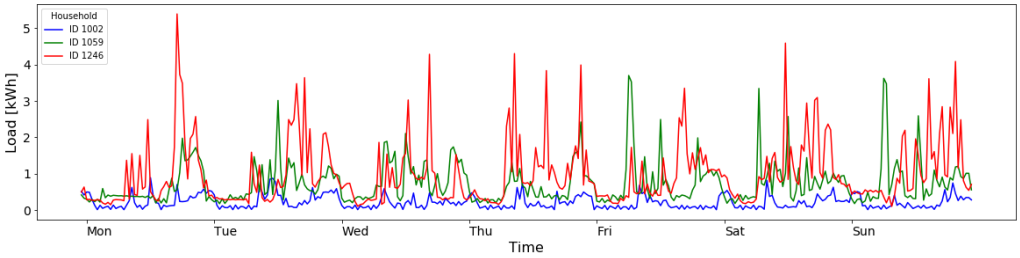



Figure 6.7.: One week of load demand for three individual households.

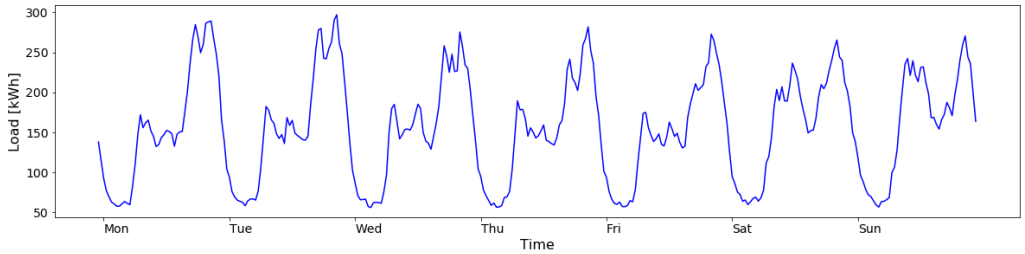



Figure 6.8.: The same week as in figure 6.7, but with the summed up load demand of 350 households.

In the shown week we already see that the working days during the week show a different pattern than the weekend days. This manifests when taking a look at the mean over all days in the data set (figure 6.9) and comparing it to the mean taken only over working days and the mean taken only over weekend days (figure 6.10).

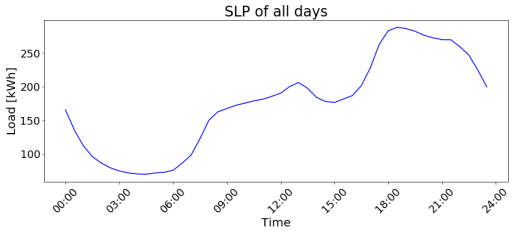
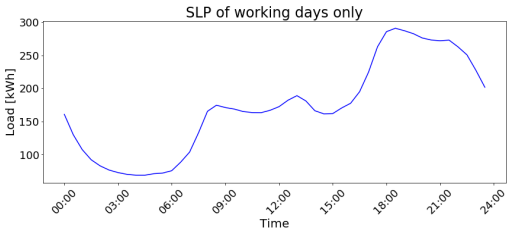
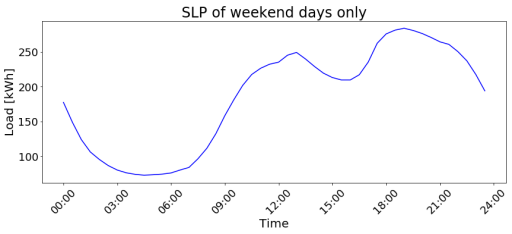



Figure 6.9.: Standard Load Profile (SLP) calculated from all days in the data.




(a) SLP calculated from only the working days in the data.




(b) SLP calculated from only the weekend days in the data.

Figure 6.10.: Standard Load Profiles (SLP) for both working days and weekend days.

We see a clear difference between working days and weekend days. Especially at noon, the average usage is higher on weekend days. We also see that the morning rise in load demand appears later on the weekends. For working days, from morning to the afternoon the power demand is relatively constant, apart from a small rise at noon, whereas for weekend days the load demand varies during this time. The evening peak is present for both working and weekend days.

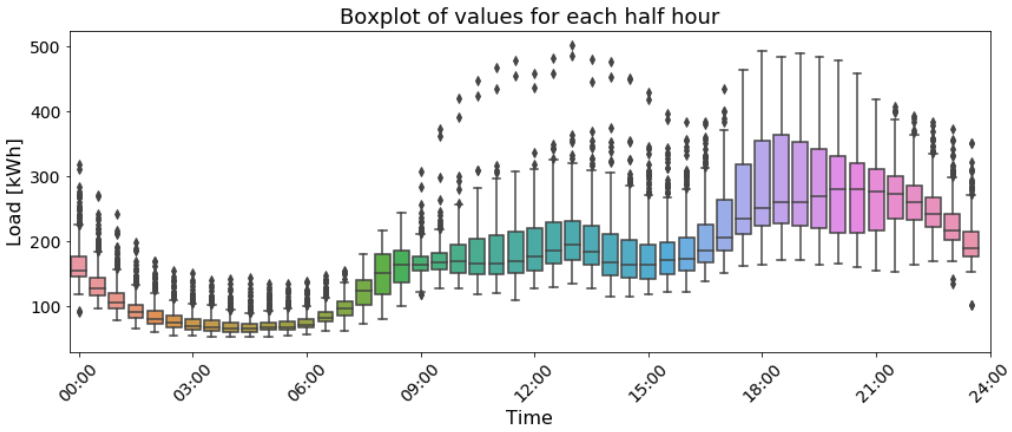



Figure 6.11.: Boxplots for each half hour of the day, calculated from all days in the data.

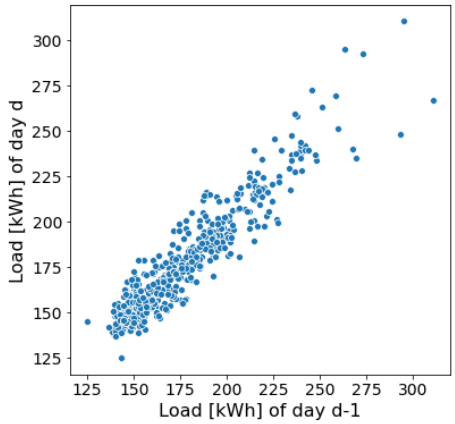



Figure 6.12.: Scatterplot of the average daily load vs. the average load of the previous day.

Not only are the mean values different when comparing working days to weekend days, but the inner-day differences are also high when looking at the variances for each half hour of a day. Figure 6.11 shows a boxplot for each half hour of a day, i.e. 48 boxplots, where each one represents the distribution of values over all days in the data at this particular time of day. We see that the variations of usage are small during the night, whereas the 50%-intervals are most spread out at the times from 5pm to 9pm. There are outliers towards higher usages at almost any time of the day, but a higher variation appears more often in the evening hours of the days.

After looking at the inner-day variations, the focus is now laid on the variations when comparing different days. The first thing catching attention is that the average value of the present day is mostly similar to the average load of the previous day. This connection is shown in the linear trend in figure 6.12. The present variation might come from the

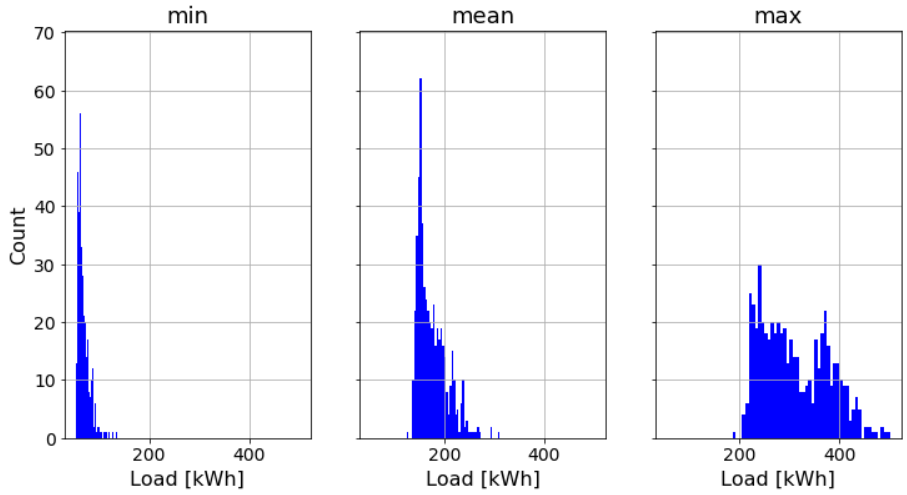



Figure 6.13.: Distributions of the daily minimum, mean, and maximum load demand, calculated over all days in the data.

fact that weekend days follow weekdays and vice versa. Where weekend days on average have a mean load of 184.35 kWh, the average mean load of weekdays is only 171.47 kWh.

The distributions of the minimum values, the mean, and the maximal values on each day of the aggregated power usage time series are presented in figure 6.13. We see that the minimum varies the least, settling on one major peak, the daily mean is more spread out, and the daily maximum covers an even further range of values that regularly appear.

This variance in and among the days is not arbitrary but highly affected by the time of year. Winter and summer do yield different energy user patterns in the data. Dividing the year into two parts, with summer from April to September and winter from October to March and plotting all days in the data, this difference in the usage becomes visible (see figure 6.14). The figure shows that in winter, the usage is generally higher throughout the entire day, with the largest increase being at the evening peak.

Looking at only the daily minima, mean values and maxima (figures 6.15, 6.16 and 6.17) throughout the entire data set, we see this observation confirmed in a less binary way. There is a gradual increase in load demand towards the winter months with the highest peak being present on Christmas Day (December 25th). The load demand then decreases again when going towards summer. In addition, when looking at the mean in figure 6.16 one can also see the slightly increased load demand on weekends.

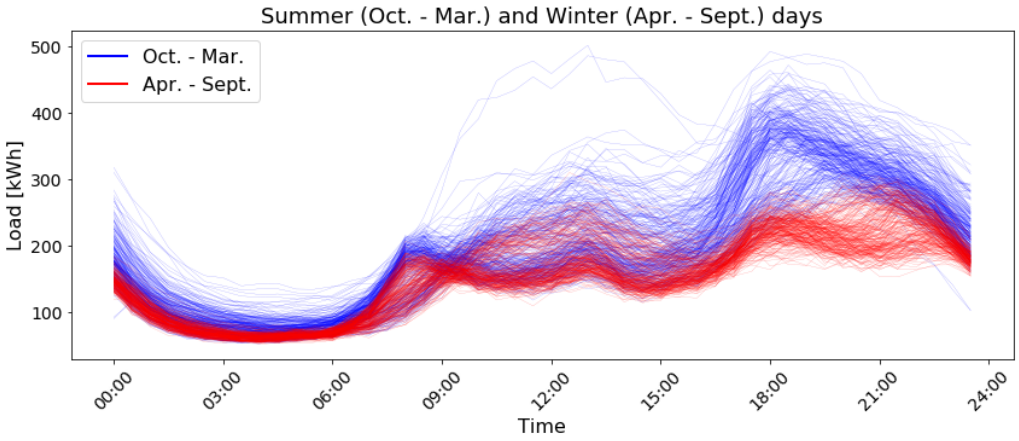



Figure 6.14.: Load demand of summer and winter days.

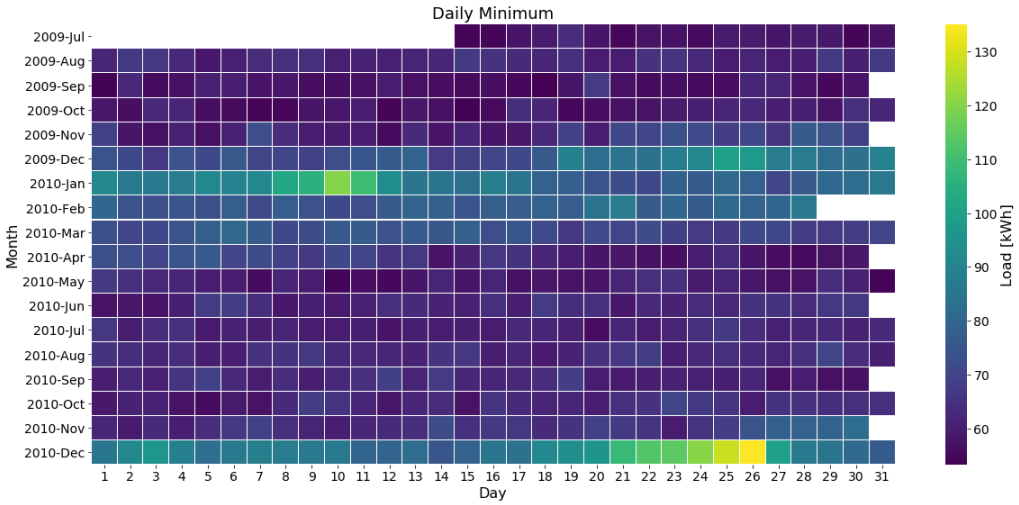



Figure 6.15.: Minimum load demand for all days in the data set.

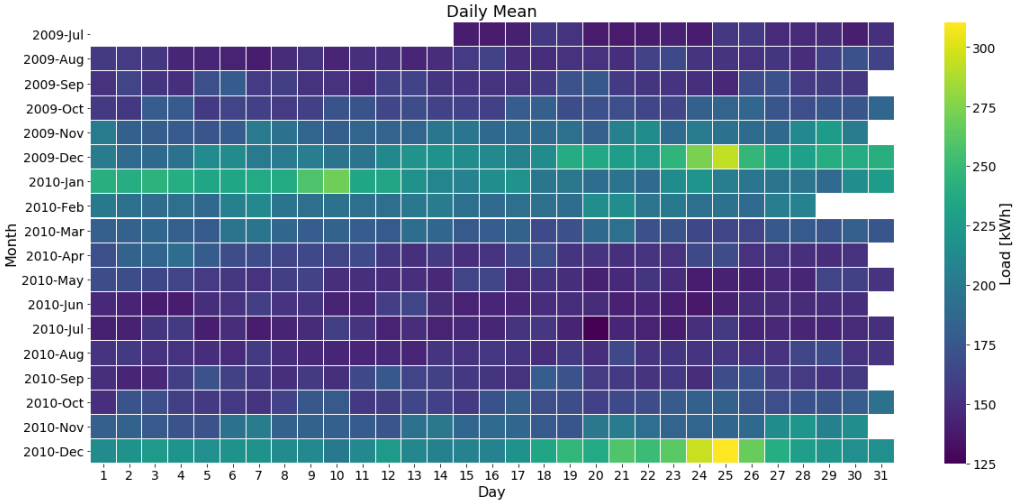



Figure 6.16.: Mean load demand for all days in the data set.

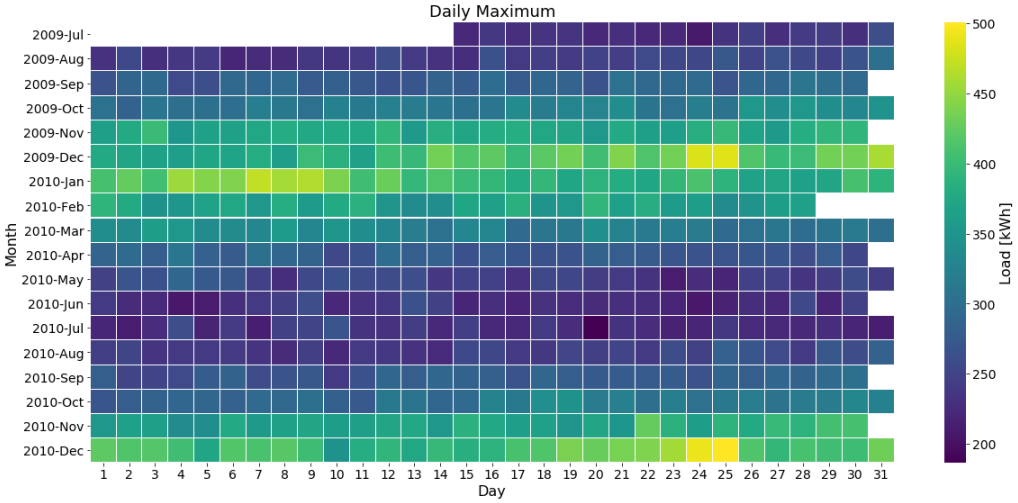



Figure 6.17.: Maximum load demand for all days in the data set.

### 6.4.2. Influence of externals

After looking at the data in its raw form, a more detailed analysis is necessary. On the one hand, to analyze how the days relate to each other and, on the other hand, to find out how external variables, such as weather variables or the day of the week, influence the power demand.

As one of the simplest forms of analyzing the connection between the load data and the weather data, the daily mean of the externals is scatterplotted against the daily average load. For the temperature in figure 6.18, a clear negative correlation between the daily average load and temperature values is recognizable, as the scatterplot is similar to a downward linear trend.

The connection to the relative humidity is less clear. One can see, that higher load values are present more often when the relative humidity is higher, but no linear trend is present. With rain, the issue is that the data of days with a higher precipitation amount is sparse. Thus, the scatterplot representation does not leave much room for information retrieval. The scatterplots of the average daily load against the relative humidity and precipitation amount can be found in the appendix figures A.1 and A.2.

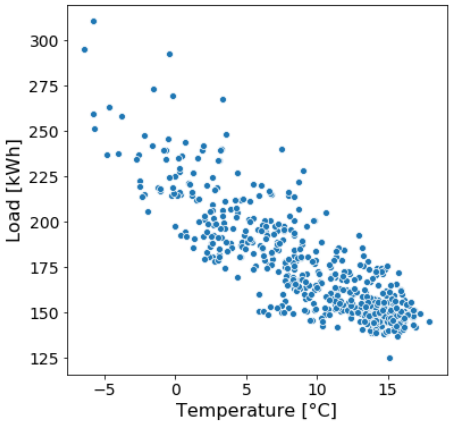



Figure 6.18.: Scatterplot of the average daily load vs. the average daily temperature.

To be able to visualize every single day as a data point instead of a time series plotted over 48 values, but at the same time not reduce each day to a single average, a PCA is applied to the data-matrix $\mathbf{V}$. The data is then projected onto the 2-dimensional space spanned by the first two principal components PC0 and PC1. Reducing the 48-dimensional data to two dimensions, each day can be visualized as a point with the coefficients corresponding to the first two principal components acting as coordinates.

To be able to interpret these points properly, one must look at the principal components PC0 and PC1, shown in figure 6.19. The first component PC0 carries approximately 70.8% variance of the data and describes a slightly increasing daily curve with the evening peak recognizable when plotted and understood as a time series. PC1 carries approximately

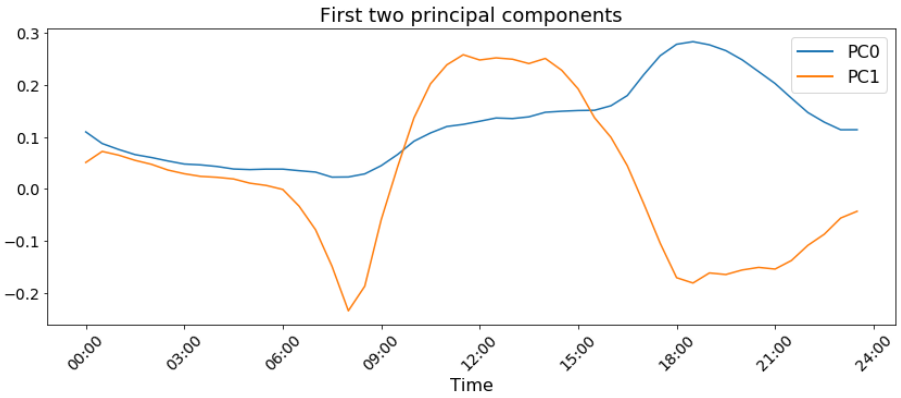



Figure 6.19.: First two principal components of the data.

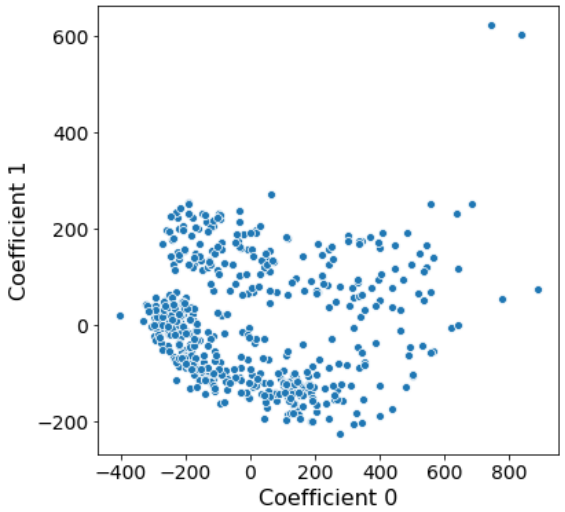



Figure 6.20.: PC 0 vs. PC 1 scatterplot.

18.4% of data variance and has a different shape. When plotted over time, the 48 values of PC1 start off relatively constant, before taking a sudden drop in the early morning, followed by a flat peak during noon. This peak reaches into the afternoon, after which the magnitude decreases again. When reconstructing the original data-matrix $\mathbf{V}$ out of the first two principal components and the two corresponding coefficients for each day, about 88% of the original variance is contained. This is not enough to fully represent the data, but sufficient to give a general overview and a simple graphical representation of how the data is structured. Projecting the data onto the first two components and plotting each day as a point with the x-coordinate being the coefficient corresponding to PC0 and the y-coordinate being the coefficient corresponding to PC1, we get a compact representation of the data.


Figure 6.21.: Working day and weekend days marked in PC 0 vs. PC 1 scatterplot. Weekend days show a higher coefficient 1.

Figure 6.20 shows a more dense area in the lower left corner, spreading out to the right, and a second more spread out area in the upper part. These two main regions are mostly separated by the value of Coefficient 1. Looking at figure 6.19, we see that a higher coefficient 1 corresponds to lower usage in the early morning and a higher power demand around noon. Giving working days and weekend days different colors, this division in the scatterplot becomes more obvious, as can be seen in figure 6.21. We can therefore conclude that a higher coefficient 1 indicates a weekend day with relatively high precision. Looking at the coefficients over time in figure 6.22, a weekly pattern is visible in coefficient 1.

However, figure 6.21 still shows a few working days in the area with a higher coefficient 1. Looking at the data, we see that all the data points in the upper right corner that

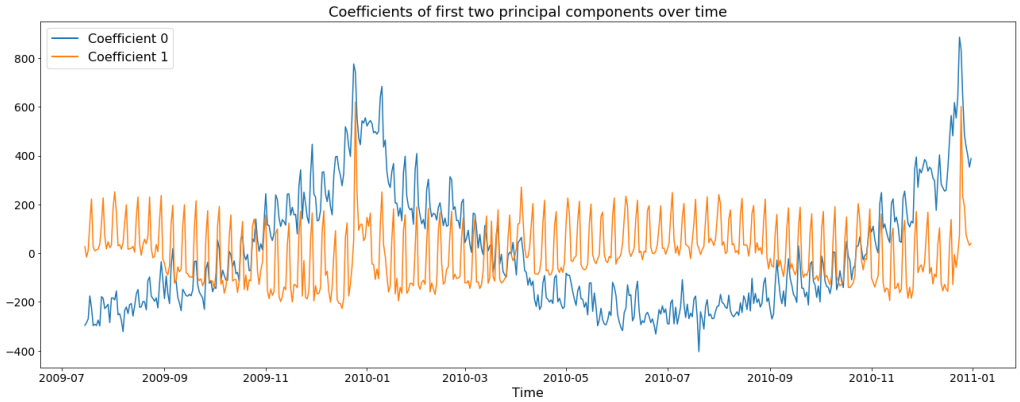



Figure 6.22.: Coefficients of the first two principal components over time.

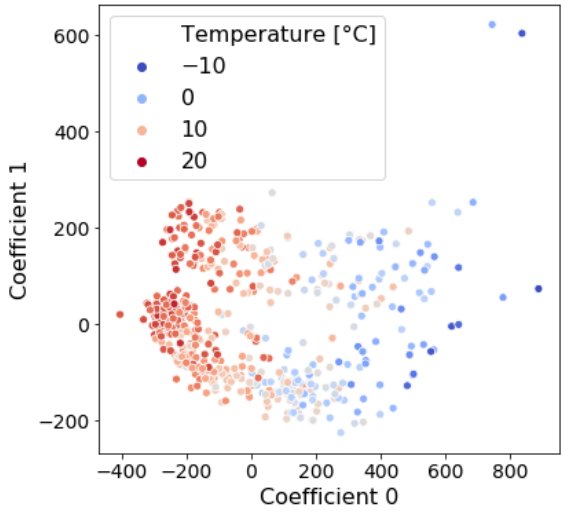



Figure 6.23.: Temperature colored PC 0 vs. PC 1 scatterplot. The colder it is, the higher coefficient 0.

are not weekend days are holidays, indicating that the power demand on holidays is similar to the demand on weekend days (also see figure A.3 in the appendix). The two outliers in the upper right corner, both having a high coefficient 0 and a high coefficient 1, are the points corresponding to the two occurrences of Christmas Day in the data.

We can conclude that coefficient 1 is strongly influenced by the day being a holiday or weekend day. But the main variance in the data, captured in the first principal component is affected by another external factor. In figure 6.22 it is clearly visible that the coefficient 0 is higher in winter. Coloring the scatterplot with the mean daily temperature in figure 6.23 each day is can be seen that the colder it gets, the higher coefficient 0 is. Looking at the structure of PC0, this means that the general usage increases with colder temperatures.

This raises the question if a connection between the two coefficients and relative humidity and the rainfall can also be drawn. But no significant connection that cannot already be explained with the previous analysis is visible (see appendix figures A.4 and A.5).

### 6.4.3. Clustering of days

After having looked at the influence of external variables on the principal components, the data is now looked at from a different perspective. The goal is to verify the intuition that the days in winter and summer show different profiles. To analyze this without drawing time borders by hand, the $K$-means clustering algorithm is applied to the collection of full days, i.e. the rows of the data-matrix $\mathbf{V}$. The number of clusters is set to $K = 4$ and the Euclidean distance is chosen as the distance metric. For two days $v_i$, $v_j$ their distance

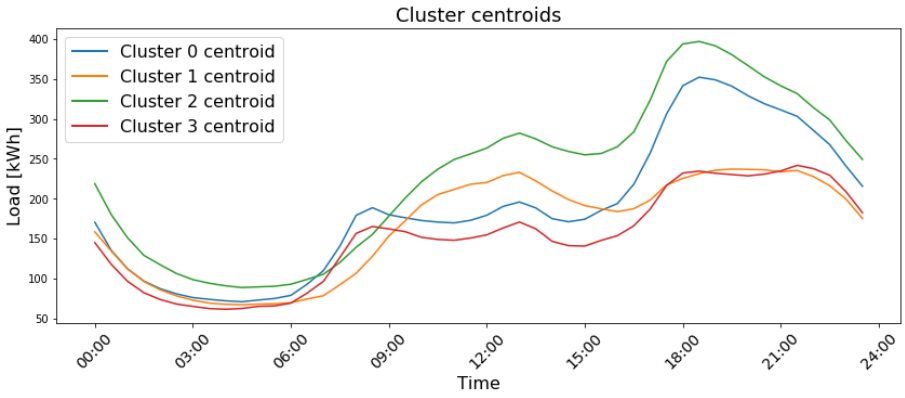



Figure 6.24.: Centroids of $K$-means clusters, found in the daily data matrix $\mathbf{V}$.

is measured as

$$d(\boldsymbol{v}_i, \boldsymbol{v}_j) = \sqrt{\sum_{l=1}^{48} \left(v_{il} - v_{jl}\right)^2}. \tag{6.1}$$

As a result of this clustering, we receive four clusters with centroids presented in figure 6.24. The first noticeable thing is, that the centroids of clusters 1 and 3 have a quite similar magnitude and shape of the evening peak, whereas the centroids of clusters 0 and 2 generally show higher magnitudes. Centroids 1 and 3 mainly differ at noon, where centroid 1 has a slight peak, whereas centroid 3 is rather flat and constant until the afternoon. A similar, comparison can be drawn between centroids 0 and 2. Centroid 0 shows a more constant behavior from the later morning till the early afternoon. The centroid of cluster 2, however, shows a steadily increasing load until noon and then decreases in the early afternoon before rising towards the evening peak, which has a similar shape to the one of centroid 0.

Again using the the PCA representation from before and coloring the data points according to their clusters in figure 6.25, we clearly see the separation of the clusters along PC0 and PC1. This separation along PC1 was earlier explained by the presence of weekend days and the variance along PC0 correlated to the external temperature. This aligns with the results of the $K$-means clustering. In figure 6.26, the affiliation of each day with its cluster is shown over time. Here, we see that the two clusters 0 and 2 with the generally higher magnitudes in their load profiles are present in the winter, and clusters 1 and 3 are found in the summer periods of the data.

Figure 6.27 provides a representation of the affiliations of clusters for each weekday. Clusters 1 and 2, both have a higher peak around noon and are mainly present on weekend days. Clusters 0 and 3 almost always appear on working days, with cluster 0 sometimes being present on Saturdays but never on Sundays. Separating between holidays or weekend days and working days (see appendix figure A.6) validates the view the days can be divided into four clusters by looking at the summer/winter period and whether the day is a working day or not.

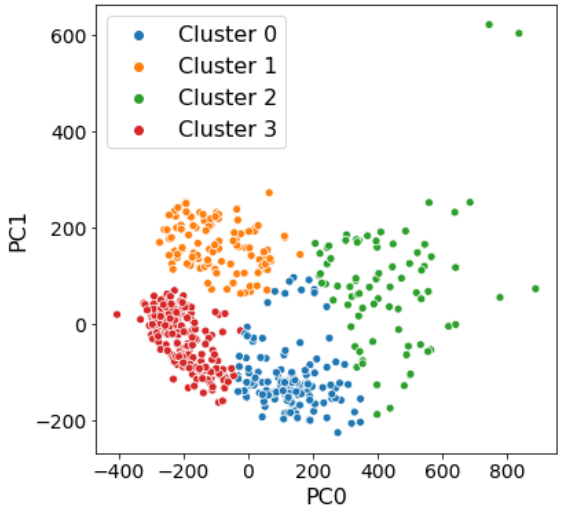



Figure 6.25.: *K*-means clusters colored in PC 0 vs. PC 1 scatterplot.


Figure 6.26.: *K*-means clusters over time.

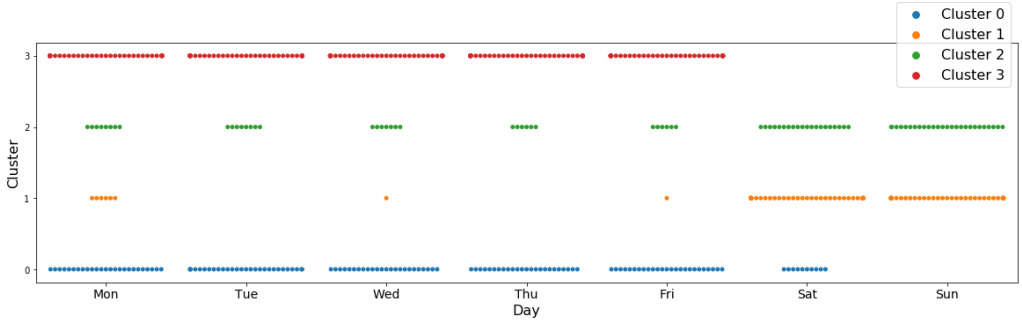



Figure 6.27.: *K*-means clusters grouped by weekday.

### 6.4.4. Analysis of NMF components and coefficients

After this more general analysis of the data, a closer look is taken at the data after having decomposed it with an Nonnegative Matrix Factorization (NMF). For the decomposition, an inner dimension of $r = 4$ is chosen, as this proves to be a number small enough to provide for interpretability and large enough to capture a reasonable amount of variance in the data. Both, $L1$ and $L2$ regularization are set to zero and the Frobenius norm is used as the distance function by which $\mathbf{WH}$ and $\mathbf{V}$ are compared. For initialization the NNDSVD is used.

In our case, each row of $\mathbf{V}$ is a day with 48 power load values measured in 30-minute intervals. Having nearly one and a half years (535 days) of data, the non-negative matrix $\mathbf{V} \in \mathbb{R}_+^{535\times 48}$ is decomposed into $\mathbf{W} \in \mathbb{R}_+^{535\times 4}$ and $\mathbf{H} \in \mathbb{R}_+^{4\times 48}$. As the NMF algorithm only provides an approximation, the product $\mathbf{WH}$ cannot represent the full variance of the original data. However, concatenating the rows of $\mathbf{WH}$ and comparing it to the original time series in figure 6.28, we see that the approximation captures the structure of the original data very well.

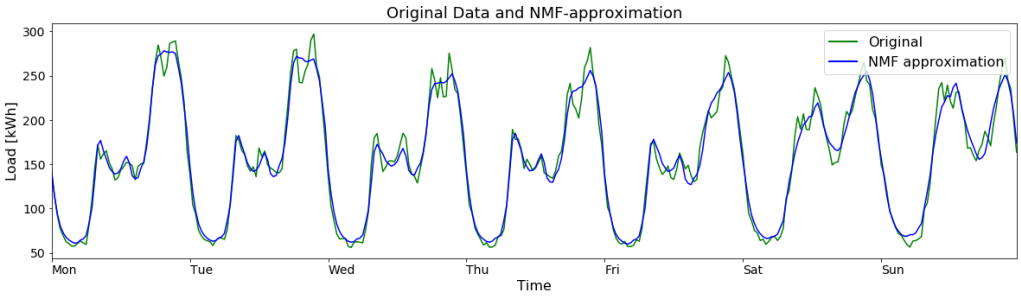
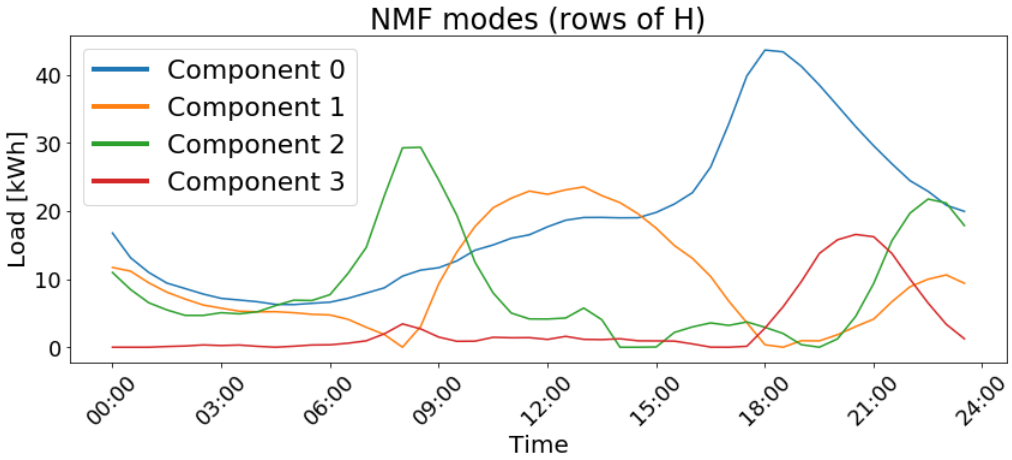


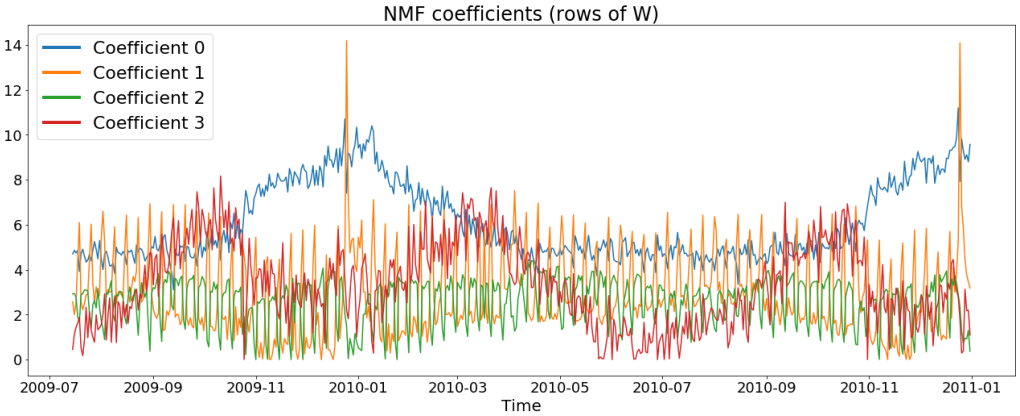

Figure 6.28.: NMF approximation of one week of data, with $r = 4$.

As the four components $\boldsymbol{h}_j$ build the basis for the reconstruction, it is worth taking a more detailed look at them. One may interpret the components $\boldsymbol{h}_j \in \mathbb{R}_+^{48}$ as load curves that are added together in a weighted way, with different weights for each day. In this view, one may then plot these load curves over the course of a day and the corresponding weights as curves over the course of the data set length (figure 6.29).

Analyzing each of these curves individually, we first look at component 0. This main component is similar to the previously extracted first principal component. It reflects the general structure of a daily load profile, slightly rising towards the afternoon and with an evening peak before decreasing again at nighttime. When plotting the corresponding coefficients and the daily mean temperature (both normalized to a mean of 0 and a standard deviation of 1) in figure 6.30, a clear negative correlation of $-0.79$ between the temperature and coefficient 0 time series can be recognized. This negative correlation is plausible, as the use of electrical devices generally increases in winter when it is colder and darker.




(a) NMF components (rows of **H**) plotted over the course of a day.




(b) NMF coefficients (rows of **W**) plotted of the course of the data set length.

Figure 6.29.: Parts of NMF decomposition with inner dimension $r = 4$ represented as time series.

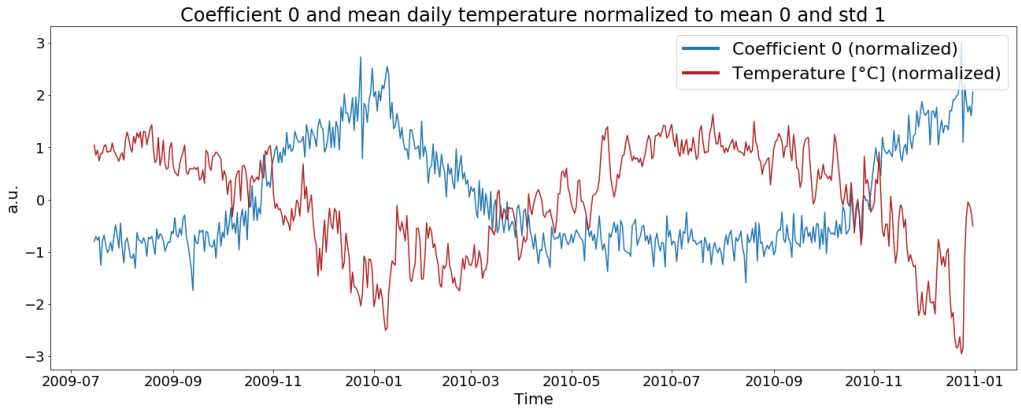



Figure 6.30.: NMF coefficient 0 values and mean daily temperature over time (both normalized to mean 0 and standard deviation 1).

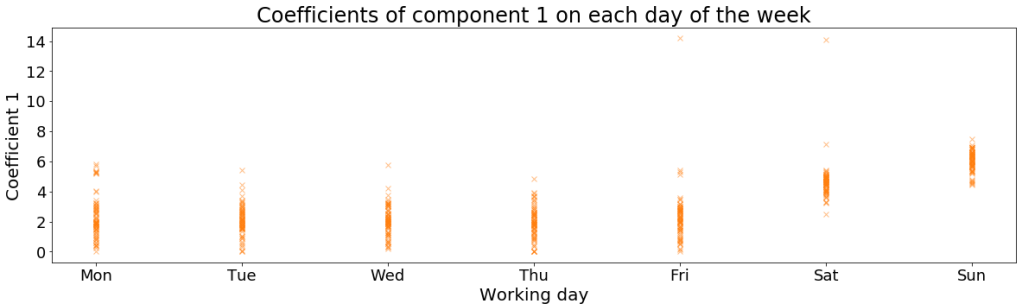



Figure 6.31.: NMF coefficient 1 values, grouped by the days of the week.

Component 1 mainly represents a peak at noon with a smaller peak around midnight. Looking at the distribution of magnitudes of the corresponding coefficient 1 over the different days of the week, we see from figure 6.31 that, on average, the coefficient is about twice as high on weekends, compared to working days. This is also plausible, as the chance of being home around noon is usually higher on weekends and holidays, compared to working days for most people.

Grouping the coefficients of component 2 over the weekdays in figure 6.32 tells a similar story. As component 2 includes a high peak in the early morning, it is plausible that the corresponding coefficients are in general higher on working days, as on weekends most people tend to get up later.

The influence of component 3 on the days is also noticeable. This component only consists of a single peak in the evening. The coefficient 3 of each day, therefore, indicates whether the evening peak is relatively high compared to the rest of the day and relative to other days. Looking at the corresponding coefficients in figure 6.33, we see a high variation over time. Together with figure 6.34 this paints an interesting picture. First of

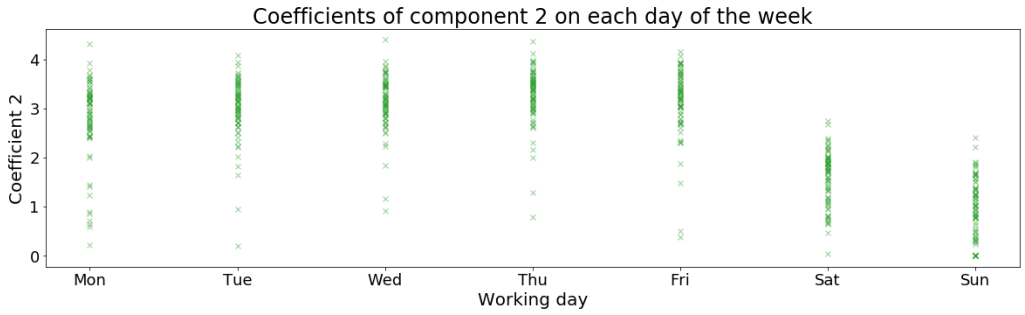



Figure 6.32.: NMF coefficient 2 values, grouped by the days of the week.

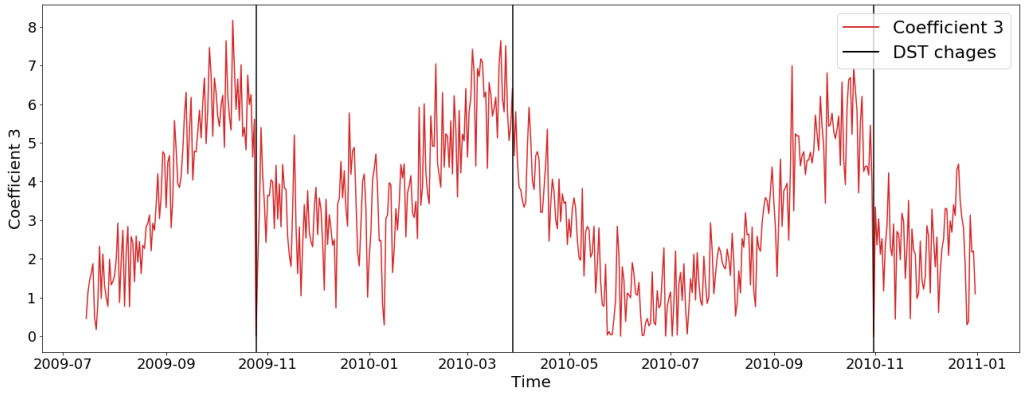



Figure 6.33.: NMF coefficient 3 values over time with changes in Daylight Saving Time marked.

all, the innerweekly differences are less extreme than with the other coefficients. Then, looking at how the series, behaves we see that an influence seems to be given by the Daylight Saving Time (DST). Especially in autumn, when DST ends, the coefficient magnitudes suddenly drop. This can be interpreted such that the clocks are set back for one hour and hence less power is needed at night compared to the days before the timezone switch. Also, as the sun shines longer in summer, this could be a reason that the coefficients for the evening peak usage are lower in summer compared to the rest of the year.

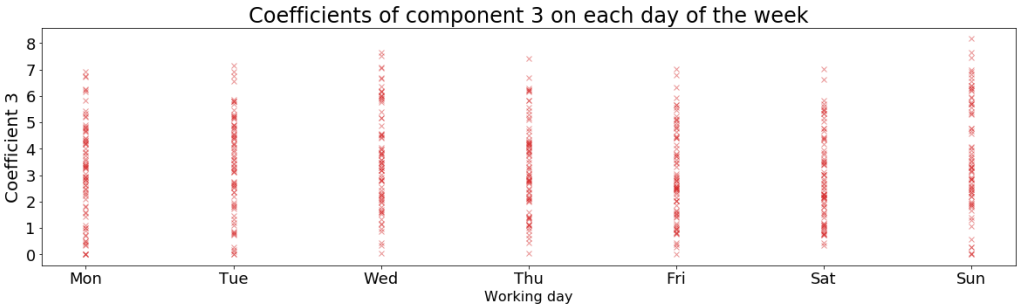



Figure 6.34.: NMF coefficient 3 values, grouped by the days of the week.

# 7. Forecasting models

> Essentially, all models are wrong, but some are useful.
>
> George E.P. Box

In this chapter, the different model types and approaches, developed for the task of load forecasting, are presented. A slight modification to the typical `.fit, .predict` model interface is given and the ideas behind the used architectures as well as some baseline models are presented.

Naïve models are created as a baseline and classic time series models such as Moving Average and Exponential Smoothing are implemented. In addition, a current state-of-the-art TSF model is implemented for comparison.

Various methods are used to examine the load time series for different features, especially shift-features incorporating the behavior of the series on more recent days.

Due to the nonnegativity of the data, the matrix decomposition with a NMF can be used to decompose single days in the time series into different modes and coefficients, describing the magnitudes in which the modes are present on each day.

## 7.1. Desired properties of a model class

When wanting to compare models that are designed to operate in a real-world environment, some properties of these models have to be ensured. The general idea of the proposed models is to predict an entire day in one step and not use an iterative point-by-point approach, possibly propagating prediction errors. For this, it is key to always look at the measured load values associated with the timestamp describing the time it corresponds to. All data points need to carry a timestamp to make sense in a daily context.

Additionally, to be able to easily exchange and compare models, they should be programmed such that they share a mutual application programming interface (API).

As the models should be able to predict entire days, leveraging the daily seasonality of the data is helpful and is going to be heavily exploited in most of the predictive models. Nevertheless, the data from before the target day that can be used reliably is limited. These limitations are also discussed in this section.

### 7.1.1. A mutual model API

All forecasting models should be accessible through a mutual application programming interface (API).

The usage of the models should generally be kept simple. Thus, all relevant preprocessing of data or creation of features should be done inside the model classes, when specific to a model. A model instance should be able to be fitted by only giving it data pairs $(\mathbf{X}, y)$ and be able to predict $\hat{y}$ when given $\mathbf{X}$. How the fitting and predicting is done in detail should be handled inside the model class.

A new method is given to the model classes. It is designed to feed data to the model without actually fitting a possibly existing internal machine learning element. Instead, it is just presenting the data to the overall model, so it can remember and save the data for later usage.

The models should be designed in such a way that no model-specific data preparation should be necessary for any kind of model. This design choice is important for several reasons. First of all, in an application case, it makes things easier for the end-user, as the model class can be used as a kind of black-box model and no deeper understanding is necessary for implementation. Secondly, switching between different models is easy, as they all operate on the same API. Additionally, this mutual API makes it fairly easy to compare models, as all models can be fitted and used in only a few lines of code with all the heavy lifting done internally. In the research of this thesis, especially the last part is important when it comes to a comparison of models that is based on the iterative fitting of models.

The general model structure underlying all different model types is presented in figure 7.1. Each model consists of three parts: A location to save seen data, a part creating features out of given and saved data, and a regression model. The models also support three methods, which can be called: `.fit(`$\mathbf{X}$`, `$y$`)`, `.predict(`$\mathbf{X}$`)` and `.feed_data(`$\mathbf{X}$`, `$y$`)`. When `.fit(`$\mathbf{X}$`, `$y$`)` is called, the data first is saved internally. In addition, a fitting process is performed: The input data $\mathbf{X}$ and $y$ together with the already seen data $\mathbf{X_{seen}}$ and $y_{seen}$ are used to create a set of features $\mathbf{X^F}$. This set of features and the originally input time series $y$ are then used to fit the final part of the model: The regression model. The regression model can be a statistical process or a Machine Learning model. If a learnable regression part is present, it is fitted to be able to predict $\hat{y}$ from the given $\mathbf{X^F}$. Thus, calling `.fit(`$\mathbf{X}$`, `$y$`)` on the model saves the given data, creates a set of features $\mathbf{X^F}$ from all seen data and $\mathbf{X}$ and finally fits a regression model on it.

When calling `.predict(`$\mathbf{X}$`)`, a similar process takes place. The given $\mathbf{X}$ and the saved data $\mathbf{X_{seen}}$ and $y_{seen}$ are used to create the features $\mathbf{X^F}$. These are then given as input to the regression model, which outputs a prediction $\hat{y}$ for the timestamps $t$ given in $\mathbf{X}$.

However, if new data is to be shown to a model with a ML subpart, without refitting the ML part, a new method is needed for this. The `.feed_data` method does exactly that. It adds newly observed data to the internally saved data of the model, without refitting possibly existing ML subparts of the model. `.feed_data(`$\mathbf{X}$`, `$y$`)` adds values and their

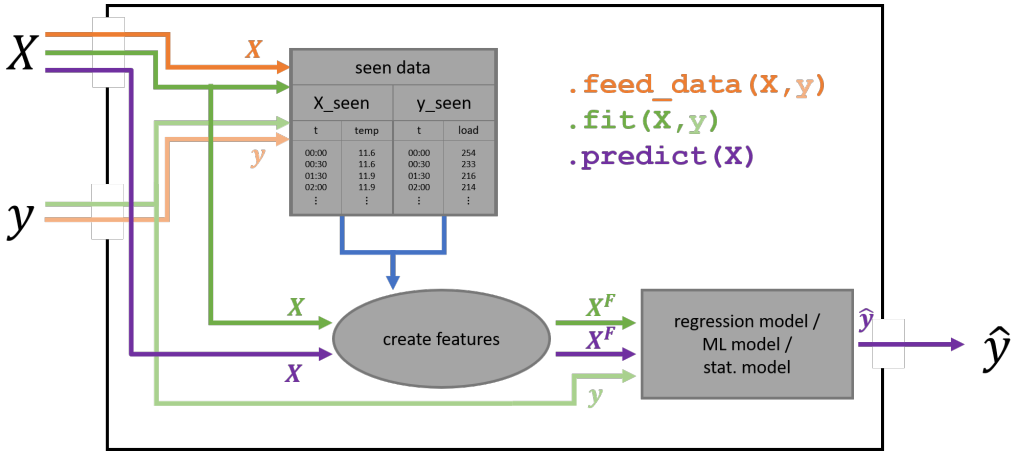



Figure 7.1.: Internal structure of the forecasting models, including three parts: A location to save seen data, a part for feature creation, and a regression model. In addition to the typical `.fit(X, y)` and `.predict(X)` methods, the models also support a `.feed_data(X, y)` method, saving the presented data internally.

timestamps that are not already in the saved data and updates old values with newly retrieved data for timestamps that had been saved before. It can be called with just $\mathbf{X}$, just $y$ or both as input.

### 7.1.2. Looking back at least 48 hours

When talking about a forecasting setup that can be used not only in a research context but also in a real-life environment, it is necessary to look at the restrictions and demands of this situation. Arguably, the most important question to answer in this context is what data is available at the time of the prediction. To answer this, it is important to understand at what time a load forecast is usually requested.

If the models are used in a real-world environment, a forecast is usually requested for a full day, to be able to plan operation schedules of power plants according to the expected power load. The question arises why one does not simply always predict the next 24 hours as seen from the current time. For a forecast of up to 24 hours, one has to be careful not to use any data that is unavailable at runtime. For the presented model designs, this includes the measured data of days that are in the past (relative to the target time for which the forecast has to be given).

To explain this in the context of the upcoming sections, a shift is defined as a certain lookback in time. A shift of length $s$ describes the time delta to look back from a certain timestamp $t$. Thus, for a value $y_t$ a shift of $s$ yields the value of $y_{t-s}$. For example $y_{2010-04-24\ 08:00}$ with a shift of $24h$ would result in $y_{2010-04-23\ 08:00}$.

When wanting to predict a full day, the last data point of the forecast is at least 24 hours in the future. Hence, using shifts below 24 hours technically make no sense, as a range of 24 hours is to be predicted and the data of the used shifts is needed before

making the prediction. Let us assume a shift of 24 hours was allowed. One then would have to wait at least until today at 23:30 to be able to also have a load value 24 hours before tomorrow's 23:30, which is to be predicted. And this scenario does not include the delay that it may take to receive the actually measured power demand at 23:30 of the present day. Allowing 24-hour shifts would thus cause the problem that a prediction for the next day cannot be made until about 30 minutes before that day would start. In an application context, this is too late. Even if it could be ensured to have the data available in time, proper planning on such a short hand's notice is challenging.

Additionally, for power plants interacting with European electricity markets, it is mandatory to register their expected power supply in advance. In Germany, for example, the planned supply schedules of power plants participating in the day-ahead spot market operated by the European Power Exchange (EPEX) in Paris have to be registered one day in advance at 12:00 o'clock [28]. But a forecast of the expected power demand should already be present before the planning of these supply schedules. As the planning of schedules has to account for many different factors, the forecasts should be available even earlier.

For the usage of shifts as features for the prediction, one, thus, must not use shifts below 36 hours. To have a safety buffer for unexpected situations, in this thesis only shifts larger or equal to 48 hours are allowed. This way it is almost certainly made sure that the data need for the calculation of the features is available at the time a prediction is requested. This real-world approach and data availability will also be considered in the design of the following models.

## 7.2. Benchmark models

When trying to measure the performance of models, it is desirable to compare the results with current state-of-the-art models and try to beat their performance on the used data set. However, it is arguably even more important to compare the developed models with basic benchmarks or baseline models, enabling the reader to rank the performance of the models in an understandable way.

A baseline model or benchmark model can serve in multiple ways. First of all, it sets a performance baseline by giving forecasts that are easy to interpret. Secondly, a baseline can help to showcase possible improvements in the forecast quality made by the developed models. Thirdly, having a benchmark model that can be understood without detailed domain knowledge helps to build a communication bridge when talking about the models' performance across multiple expertise fields [30].

Thus, having reliable and easily interpretable models that also set a performance baseline are important when talking about forecasting models. Such models should incorporate a simple idea that is easy to understand but still tackle the prediction problem with a reasonable approach.

In this section, two simple approaches used as baseline models to predict the load of an upcoming day are presented: A group of naïve models that expect the following day

to be exactly like one of the most recent days, and a group of models that expect the target day to be an average over past days. Finally, we will take a look at N-BEATS, one of the current state-of-the-art models in Time Series Forecasting.

### 7.2.1. Repeat Day models

One of the simplest baseline-ideas for creating a forecasting model is to expect the day that is to be predicted to be exactly the same as one of the days the model has recently seen in the past data. These naïve models are based on the idea that the behavior of consumers will not change in a short period of time. A variety of models created from this idea is presented in table 7.1. We will refer to models of this type as Repeat Day models.

| Modelname | Description | Pred. for 2019-04-01 08:00 (Monday) |
|---|---|---|
| RepeatLast Day Forecaster | A forecaster that forecasts the value for the given timestamp by expecting it to be the same value as at the last corresponding time in its seen data. | Value of the last seen day at 08:00 |
| RepeatLast WeekdayWeekend Forecaster | A forecaster that forecasts the value for the given timestamp by expecting it to be the same value as at the last corresponding time at the last corresponding working day or weekend day in its seen data. | Value of the last seen working day (Mon-Fri) at 08:00 |
| RepeatLast DayOfWeek Forecaster | A forecaster that forecasts the value for the given timestamp by expecting it to be the same value as the last corresponding time at the last corresponding day of week in its seen data. | Value of the last seen Monday at 08:00 |

Table 7.1.: Models based on the idea of repeating a single day as forecast.

### 7.2.2. Smoothing models

The next extension to the idea of repeating a single day as the forecast is to take an average over the days already seen. This can be a weighted average, looking at all past days, or an average calculated over only a few past days. Models averaging over days tend to give a smoother forecast than the models simply repeating single days, but are still based on a simple idea. Models of this kind will be referred to as smoothing models.

In this context, it needs to be specified what kind of averaging is done. As the smoothing models should still capture the general shape of a day's power demand, it is not enough to simply take a single value as the average and predict the requested day to have a constant power demand. Instead, averaging in this context means taking the average for each inner-day timestamp individually. This means that, for example, for the 08:00 time step of the target day, the smoothing model predicts it as the average over all 08:00 time steps of the past days.

The simplest implementation of the smoothing idea is to take the average over all seen days or all days corresponding to the requested day in a certain manner, like the day of the week. However, these models, based on an average over all seen days, also called a Standard Load Profile (SLP), are hardly able to react to changes in data retrieved more recently. It, therefore, makes sense to also take a look into the direction of smoothing and ARIMA models and not only take an unweighted mean over all past days, but instead use Exponential Smoothing or a Moving Average.

To not use a vast amount of baseline models, only a pure Moving Average model and a model based on simple Exponential Smoothing are used in addition to the SLP based approach. All used smoothing models are presented in table 7.2.

### 7.2.3. N-BEATS

When comparing models against benchmarks, on the one hand, need simple and understandable models are needed, giving a baseline of errors that should be beaten by more sophisticated approaches. On the other hand, it is useful to also look at the current state-of-the-art and see how the models' performance compares to it. For this, Neural Basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS) developed by Oreshkin et al. is applied on the load data, to receive a state-of-the-art performance as a benchmark on the forecasting task [68]. The implementation of N-BEATS is based on a Python project by Remy [177].

The way N-BEATS is set up, its forecast is a continuation of the input time series. However, in the real-world application scenario, we do not want to predict the 24 hours following the available data, as this would imply using shifts under 24 hours. There are two possible solutions to this. One option is to simply skip the gap and prepare the data such that the target day is shown to the model as the continuation of the input time series. However, this might be harder to learn and also would be a severe alteration to the original N-BEATS training. Instead, 48 hours are used as the horizon length. Afterward, only the forecast for the originally requested timestamps is used. This way the model can continue the time series with its forecast, as is the idea of the original paper. However, it has to be noted that the horizon of 48 hours (corresponding to 96 data points) is relatively long.

For the implementation of the N-BEATS model, the choice of hyperparameters is oriented on the generic model of the original paper [68]. Four linear layers per block, each with a width of 512 units are used. The model is built out of 30 stacks, each with one single block, having its own weights. In each block, the dimension of the expansion

| Modelname | Description | Pred. for 2019-04-01 08:00 (Monday) |
|---|---|---|
| Simple SLP Forecaster | A simple forecaster that creates a single SLP. When predicting, it returns the mean of the corresponding values of all its seen data. | Mean over all 08:00 values of the seen data. |
| WeekdayWeekend SLP Forecaster | A forecaster that creates separate SLPs for weekdays and weekend days. When predicting, it returns the mean of the corresponding values of its seen data. | Mean over all 08:00 values of all working days (Mon-Fri) in the seen data. |
| Daily SLP Forecaster | A forecaster that creates separate SLPs for every single weekday. When predicting, it returns the mean of the corresponding values of the seen data. | Mean over all 08:00 values of all Mondays in its seen data. |
| MovingAverage Forecaster (window_size) | A forecaster using Moving Averages over the corresponding values as forecast out of a sliding window. The parameter `window_size` determines the size of the sliding window. | Mean over the last `window_size` 08:00 values of the seen data. |
| ExpSmoothing Forecaster ($\alpha$) | A forecaster using a simple Exponential Smoothing over the corresponding values as forecast. The parameter $\alpha$ describes how much weight is given to the latest value in the data. With $y$ describing the smoothed version of $x$, the Exponential Smoothing is calcuated by $$y[t] = \alpha * x[t] + (1 - \alpha) * y[t - 1].$$ | An Exponential Smoothing of the past 08:00 values of the seen data. |

Table 7.2.: Models based on the idea of averaging past days as the forecast.

coefficient $\theta$ is chosen to be 32. As the loss function, the MSE is used and optimized by an Adam optimizer with batch-size 1024.

In contrast to the usage of N-BEATS in the original paper, in this thesis, no ensembling strategy is used. Oreshkin et al. use an ensembling strategy over N-BEATS instances with different input window lengths and initializations to improve their forecast accuracy [68, 109]. But in the iteration-based test setup used in this thesis, using ensembles would largely exceed reasonable computation times. Instead, only a single instance is trained for 300 epochs.

## 7.3. Shift-based models

After the Repeat Day models and the smoothing approach, the next kind of models goes one step further. As usually the power demand does not rapidly change in extreme ways, the data of past days is still incorporated, but instead of making the weights given to past days fixed, they are made learnable.

Additionally, external variables are included. Time-corresponding variables, like information about holidays and weekends and external weather information, are utilized. These variables are collected about the past days as well as about the days that are to be predicted. As we already saw in section 6.4.2, of the available external weather data only the temperature plays a major role, so the models focus on this information as the main external data source. At this point, the following assumption is made: To eliminate an additional factor of uncertainty in the prediction process and only focus on the quality of our models, the measured weather data is used as a weather forecast. Thus, it is assumed that the entire external data even for future days is available in advance.

### 7.3.1. General idea behind the model design

The general idea behind the models is to predict on a daily basis, rather than only single points. But it is not desirable to train a separate regression model for every time step in a day. Instead, the goal is to use one single ML regression model that can predict a data point for all innerday timestamps. If the predictions should still be specific, though, the time of day needs to be given to the model for it to be able to react accordingly.

There are two main options to incorporate the time of day. The first idea is to use the time of day that is requested as a direct input variable to the regression model. This could for example be done by using an integer value 0-47 to account for the 48 values of each day. The second option is to use one-hot-encoding with a binary variable for each time of day. The problem with the latter approach is that it largely increases the number of features and, hence, the dimension of the feature space. In the case of fewer samples, this might lead to poor performance of the models as they have to operate on a more sparse data set in high dimensions. Thus, one-hot-encoding in this case is not the perfect choice for modeling the time of day.

The issue with the integer encoding is that some regression models, like Linear Regression, see an integer value not as a categorical encoding, but as a numerical variable. This is fine for the middle of the day but suggests a linear increase of a variable that encodes the time of day, which does not hold at midnight. Additionally, using this important feature of the time of day in only one variable could cause the model to not react to it accordingly when too many other features are present.

Hence, while encoding the time in a variable is an option, a second approach is also possible. Instead of directly modeling the time via one or many variables, one may also input the time in an indirect way, similar to the way it was done for the smoothing models. These models also had no explicit knowledge of a time variable, but implicitly used the assumption that a value in the morning will at least be similar to past values in the morning and not suddenly show values of an evening peak. Thus, instead of putting the time into a direct variable, past features, like shifts of the same corresponding time are used. This way the models implicitly get information about the expected magnitude of power load. The implicit connection of the daily similarity of values in similar times is something that the models then do not have to learn. This way a lot of information can be given to the model in only a few features and the integer description of the daytime can still be used as an additional feature if wanted.

### 7.3.2. Internal feature creation

When using the information of the originally given inputs $\mathbf{X}$ and $\boldsymbol{y}$ in the way described above, features like shifts have to be created before they can be used by the internal regression model. In general, the target power load $\boldsymbol{y}$ as well as the given externals $\mathbf{X}$ come into the model in the form of time series. Thus, each value has to have a corresponding timestamp $t$. The goal is to create a data set $(\mathbf{X^F}, \boldsymbol{y})$ consisting of tuples $(\boldsymbol{X_t^F}, y_t)$ with an input $\boldsymbol{X_t^F}$ and target $y_t$, on which the internal regression model can be fitted.

It is important that when wanting to predict the power load for the time $t$ in the future, it is ensured that the needed external data to create $\boldsymbol{X_t^F}$ is already available ahead of time. Additionally, it has to be made sure, that the past data needed for the feature creation has already been seen by the model. For creating the features of $\mathbf{X_F}$, the input time series $\mathbf{X}$ as well as the already seen and internally saved data are used.

The procedure of preparing the data $\boldsymbol{X_t^F}$ for the regression model is needed for fitting the model, as well as for predicting $\hat{y}_t$ when only the external information $\boldsymbol{X_t}$ for this timestamp $t$ is given.

Each $\boldsymbol{X_t^F}$ is created by

- looking up $y_{t-s}$ for all shifts $s \in S$ in the already seen data (e.g. $S = \{2days, 1week, 2weeks\}$),
- adding externals (namely the temperature) $\boldsymbol{X_t}$ of the timestamp $t$,

- adding shifts of externals $\boldsymbol{X}_{t-s'}$ with shifts $s' \in S'$ where usually $S \subset S'$,
- adding temporal information about the day for which the value is meant (whether or not the day is a working day and how long the duration from sunrise to sunset is on that day),
- adding statistical features about past, horizon and future time ranges (see more in section 7.3.5).

The flow of data for creating the features is also shown in figure 7.2.

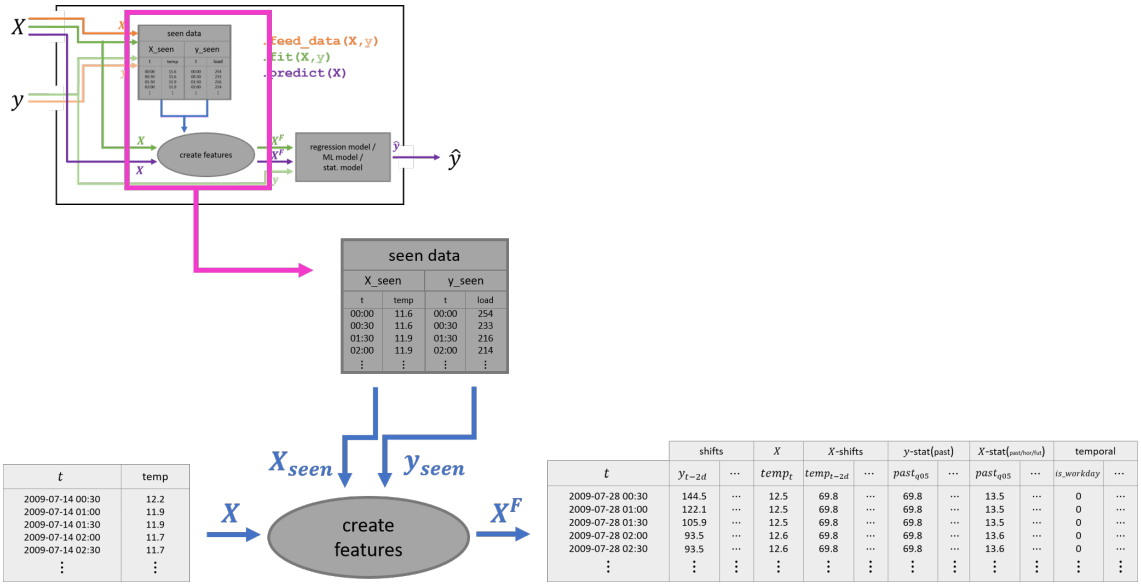



Figure 7.2.: Feature creation part of the internal structure of forecasting models.

Having started at a time series $\mathbf{X}$ together with the already seen data that was saved internally, a feature matrix $\mathbf{X^F}$ with columns described in figure 7.5 is retrieved. Note that the index of the processed data in figure 7.4 starts later than the index of the originally input data shown in figure 7.3. This is due to the reason that for the first two weeks of the given index, no shifts could be calculated (the largest shift was $2weeks$). These indices are therefore dropped when creating $\mathbf{X^F}$. This way the regression model is not fitted on incomplete data, which would otherwise need to be handled.

### 7.3.3. Choice of proper shifts

With the model design of this thesis, it is not the goal to predict iteratively. Only shifts larger than 48 can be used. However, the shifts should also not go back too far, as the models should be able to already learn in a situation where a smaller amount of data is available. Thus, looking back over long time ranges is not an option, as it would limit the amount of data that could be used for model training.

| $t$ | temp |
|---|---|
| 2009-07-14 00:30 | 12.2 |
| 2009-07-14 01:00 | 11.9 |
| 2009-07-14 01:30 | 11.9 |
| 2009-07-14 02:00 | 11.7 |
| 2009-07-14 02:30 | 11.7 |
| ⋮ | ⋮ |

Figure 7.3.: $\mathbf{X}$ initially given to the model.

| | shifts | | $X$ | $X$-shifts | | $y$-stat(past) | | $X$-stat($_{past/hor/fut}$) | | temporal | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $t$ | $y_{t-2d}$ | ⋯ | $temp_t$ | $temp_{t-2d}$ | ⋯ | $past_{q05}$ | ⋯ | $past_{q05}$ | ⋯ | $is\_workday$ | ⋯ |
| 2009-07-28 00:30 | 144.5 | ⋯ | 12.5 | 69.8 | ⋯ | 69.8 | ⋯ | 13.5 | ⋯ | 0 | ⋯ |
| 2009-07-28 01:00 | 122.1 | ⋯ | 12.5 | 69.8 | ⋯ | 69.8 | ⋯ | 13.5 | ⋯ | 0 | ⋯ |
| 2009-07-28 01:30 | 105.9 | ⋯ | 12.5 | 69.8 | ⋯ | 69.8 | ⋯ | 13.5 | ⋯ | 0 | ⋯ |
| 2009-07-28 02:00 | 93.5 | ⋯ | 12.6 | 69.8 | ⋯ | 69.8 | ⋯ | 13.6 | ⋯ | 0 | ⋯ |
| 2009-07-28 02:30 | 93.5 | ⋯ | 12.6 | 69.8 | ⋯ | 69.8 | ⋯ | 13.6 | ⋯ | 0 | ⋯ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

(a) Part of $\mathbf{X^F}$.

| $t$ | $y_t$ |
|---|---|
| 2009-07-28 00:30 | 121.3 |
| 2009-07-28 01:00 | 99.7 |
| 2009-07-28 01:30 | 82.6 |
| 2009-07-28 02:00 | 83.2 |
| 2009-07-28 02:30 | 78.7 |
| ⋮ | ⋮ |

(b) Corresponding part of $\boldsymbol{y}$.

Figure 7.4.: $\mathbf{X^F}$ coming out of the feature creation, together with the corresponding target $\boldsymbol{y}$.

| shifts | | | $X$ | $X$-shifts | | | $y$-stat(past) | | | | $X$-stat(past/hor/fut) | | | | | | | | | | | | temporal | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $y_{t-2d}$ | $y_{t-1w}$ | $y_{t-2w}$ | $temp_t$ | $temp_{t-2d}$ | $temp_{t-1w}$ | $temp_{t-2w}$ | $y\_past_{q05}$ | $y\_past_{q95}$ | $y\_past_{mean}$ | $y\_past_{std}$ | $temp\_past_{q05}$ | $temp\_past_{q95}$ | $temp\_past_{mean}$ | $temp\_past_{std}$ | $temp\_hor_{q05}$ | $temp\_hor_{q95}$ | $temp\_hor_{mean}$ | $temp\_hor_{std}$ | $temp\_fut_{q05}$ | $temp\_fut_{q95}$ | $temp\_fut_{mean}$ | $temp\_fut_{std}$ | $is\_workday$ | $sunshine\_dur$ |

Figure 7.5.: Columns of $\mathbf{X^F}$ coming out of the feature creation, when the shifts are chosen to be $S = \{2days, 1week, 2weeks\}$ and $S' = S$.

To eliminate redundant features fed to the forecasting models, it is important to choose proper shifts [178]. Bouktif et al. report overfitting of their LSTM based forecasting model when too many shifts were chosen [73]. To avoid such problems, a process for selecting shifts by looking at correlations and the Fourier Transform of the data is presented. To not simply use all possible shifts up to a certain upper boundary of e.g. 2 weeks, but only selecting a few, can even be understood as a kind of preset unlearned attention mechanism.

Applying a Discrete Fourier Transform (DFT) to the load time series is helpful to analyze the frequencies prevalent in the data. Especially plotting the magnitude of the complex Fourier coefficients over the associated periodicities, i.e. the inverse of the associated frequencies, is useful to visually inspect the data.

A second way to analyze the importance of shifts is by looking at the correlation of the original time series with its shifted version. To evaluate which shifts are important by looking at autocorrelations of the raw data, we would have to look at $(14-2)\cdot 48 = 576$ shifts when setting an upper limit of 14 days and a lower limit of 2 days. But due to the structure of the data and the strong daily cycle, it is reasonable to focus on shifts that are multiples of 24 hours. However, it has to be noted that this suppresses possible inner-day correlations.

Nevertheless, with this idea in mind, another method for choosing shifts was performed. This procedure was designed to fit the novel NMF-based models described in section 7.4. However, to receive a proper comparison among shift models and NMF models, the shift models will use the same shifts that are also chosen as lags for the NMF-based models. This way the shift-based models described here can be seen as baseline models for the NMF models.

The process for choosing the shifts is laid out in detail in section 7.4.5.

### 7.3.4. Temporal features

In addition to the shifts, the use of temporal information about the day of the target values is one of the main processes in feature creation. As these features can be calculated based only on the date information contained in the index, they will be referred to as `index_features`. Using index features includes two features: The first one, a boolean value, is `is_workday`. This unites two questions: a) is the day a weekend day; and b) is the day a (public or observed) holiday. If both are answered with No, `is_workday` is set to True. If either one is answered with Yes, `is_workday` is set to False.

The second feature that does not rely on past values of power usage or external time series is the sunshine duration on the target day. The idea for using this feature is taken from Caro et al. [179]. As no external weather information, like cloud coverage, etc. is used for this feature, the sunshine duration is calculated as the number of seconds from sunrise to sunset. To get the time of sunrise and sunset, the Python Package `astral` is used [180]. For this package to work, a specific geographical location for which sunrise

and sunset should be calculated has to be given. As the location of where the load data is collected, apart from it being in Ireland, is unknown, the center of Ireland is used as the location for the sunshine calculation. According to the Ordnance Survey Ireland, the geographical center of Ireland is where the 8° Meridian West meets the $53° - 30'$ North Latitude [181].

In this thesis, no feature is designated to explicitly incorporate the seasons, as is done in some other works. This decision is based on two main points: First of all, a hard cut between seasons does not represent the way of nature, as spring does not immediately end with summer starting the next day. A soft transition between seasons is more accurate. Secondly, looking at the power demand and the corresponding weather of past times, as well as the weather information of the target day, already gives a sense of the season. The yearly temperature curve of Ireland has a strong seasonal connection and, thus, the seasons should already be captured by the information about temperatures. Such a connection between seasonality and temperature was also mentioned by Haben et al. [31].

### 7.3.5. Statistical features

Thus far the models were given features with which they could potentially make a prediction fitting to the time of day. But what about the time of the year when the seasons are changing? To capture this, some additional features are created with the idea of supporting the information already contained in the temperature data.

The idea behind these features is rooted in the analysis of the data done in section 6.4. In this analysis, we saw that the power demand in the winter months is generally higher than in summer. The inverse phenomenon is present when looking at the temperature. This leads to the fact, that e.g. a load value that is considered high in summer would be considered low in winter. Adding some statistical features about the time series at hand should help the model to rank the past values. In other words, these features could help the model to rank a certain value as high or low compared to other values in this time range. The same concept applies to externals, like the temperature. A temperature relatively low in summer can be considered high in winter. Computing the statistical features should allow the model to rank the temperature at the considered time $t$ and use this to its advantage.

Multiple statistical quantities are calculated over three main time ranges, called past, horizon and future. The idea of these features is inspired by the work of Hopf et al. [113]. Noting the index $t$ as the target index at any target day, the term future is used for the 24 hours of the target day containing $t$. The term horizon describes the 24 hours of the day right before the target day. The third time range, the past, describes the second to last day before the target day containing $t$.

This means that the three ranges future, horizon and past only depend on the date information of $t$ and not on the time. E.g. $\boldsymbol{X}^{\boldsymbol{F}}_{\mathbf{2010-04-23\ 08:00}}$ will carry the same statistical feature values as $\boldsymbol{X}^{\boldsymbol{F}}_{\mathbf{2010-04-23\ 17:30}}$, but different ones than $\boldsymbol{X}^{\boldsymbol{F}}_{\mathbf{2010-04-24\ 08:00}}$.

The ranges are also presented in figure 7.6.

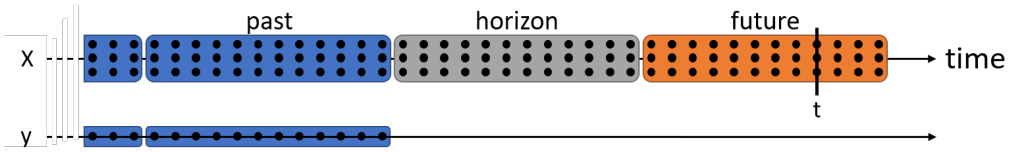



Figure 7.6.: Time ranges future, horizon and past used for the calculation of statistical features, relative to the target time $t$. The external data $\mathbf{X}$ is available for all three time ranges. The load data $\boldsymbol{y}$ is only available for the past time range.

For horizon and future, only the externally given temperature can be used. The values of $\boldsymbol{y}$ are neither available for the horizon nor for the future time range (also see section 7.1.2). For the past time range, however, the values of $\boldsymbol{y}$ are available and features can be calculated from them.

For all three ranges the features `q05`, `q95`, `mean` and `std` (i.e. the 5th percentile, 95th percentile, the mean and the standard deviation) are calculated for each time step $t$. The percentiles are chosen instead of `min` and `max` values, because they are less prone to outliers. For the past range, features are created for both $\boldsymbol{y}$ and the externals $\mathbf{X}$, for horizon and future features are only created from the values given in $\mathbf{X}$.

For each time step $t$ the corresponding past, horizon and future time ranges are calculated, as well as whether all needed data is available. To use these features it should be the case that not just any day is predicted but rather the day after tomorrow, to have the past values of $\boldsymbol{y}$ available. Additionally, it is noted that for calculating the future statistical features, the external data going ahead of $t$ must be given to the model via the `.feed_data` method, even if the entire day is not requested.

### 7.3.6. Choice of regression models

Now that the creation and choice of features in $\mathbf{X}^{\mathbf{F}}$ is defined, a regression model, able to use these features to give a prediction $\hat{\boldsymbol{y}}$, needs to be chosen.

As a simple linear model, a Linear Regression with $L2$ regularization, also commonly known as Ridge Regression, is chosen. The strength of the regularization parameter is determined by 5-fold cross-validation and chosen from the range of $10^{-3}$ to $10^{4}$. A second model that also works on limited amounts of data is the Random Forest (RF) Regressor. For the RF, each tree is built on a bootstrap of $n$ samples drawn with replacement, where $n$ equals the number of totally available samples. Additionally, at each node of the $m$ total features, $m/3$ are sampled when choosing the split. In total, 100 trees are grown this way, with a maximum depth of 15. The minimum number of samples needed for a split is set to 2, and only 1 sample is needed for a leaf. Both regression models are implemented in Python using the `scikit-learn` library.

The chosen regression models are the same ones that will also be used in the later-described NMF-based models, for which the shift-based models can be considered as benchmark models.

To also compare the performance of these two simpler models to the use of a Neural Network, a model version with a dense NN is also trained. For this network, four hidden layers with 50, 25, 15 and 8 neurons and ReLU activation function are used. The model is trained using an Adam optimizer to minimize the MSE between forecast and target time series. The training is done for 50 epochs with a batch-size of 32.

## 7.4. NMF-based models

Usually, when the schedules for a local power plant have to be planned, the expected power load of an entire day needs to be predicted. But all models presented thus far predict single points that are then concatenated to a longer prediction. These models hence need to get some kind of information about where in a day the time step that is to be predicted is located. Even though the usage of shift features leverages the daily periodicity of the data, it is still tried to describe the future day by 48 separate predictions.

The main idea of the models in this section is that, instead of predicting each single data point of the day separately, the information of a day can be reduced into a few coefficients. These coefficients can be predicted to reconstruct the forecast from them. In this scenario, only a few coefficients, instead of 48 individual values, have to be forecasted for each day. Furthermore, a single coefficient does not necessarily represent a certain time during the day but instead gives more of a general statement about the entire day.

For this reason a NMF-based model is proposed that does not work on individual data points, but on full days instead. The model slices the time series into days, stacks these days into a matrix and applies a NMF on it, yielding coefficients $\boldsymbol{w_i}$ for each day $i$. These coefficients can then be used as targets of an internally fitted regression model.

### 7.4.1. Inherent periodicity assumption

The general idea behind this model design is to exploit the daily periodicity of the load demand time series data. All days have at least a similar structure. They show a recognizable shape with variations for each day $i$. As we already saw in section 6.4, not all 48 dimensions are needed to describe a day with most of its variance captured. Instead, a more efficient way to describe a day can be achieved via dimensionality reduction.

When reducing a day with 48 values to $r$ dimensions, for each day only $r$ instead of 48 values need to be predicted. The entire day can then be reconstructed from these few values. However, it should be remembered that for this entire process, a periodicity of 24 hours in the data is assumed. This assumption is valid for the used data set but does not hold for all time series data sets.

When it comes to the choice of methods used for dimensionality reduction, multiple options come to mind. But as an interpretable, simple method operating on non-negative

data is wanted, the decision falls to the Nonnegative Matrix Factorization (NMF).

### 7.4.2. From a single point to an entire day

For the NMF-based approach, the point of view is changed to no longer look at single data points, but at entire days instead. Therefore, the time series $\boldsymbol{y}$ is no longer viewed as a single time series with measurements $y_t$, but instead, the time series is sliced into days with 48 data points each. The sliced days are stacked together as the rows of a data-matrix $\mathbf{V}$. Henceforth, it can be talked about days $\boldsymbol{v_i} \in \mathbb{R}_+^{1\times 48}$, where the index $i$ is still related to time but only carries the date information and no innerday time information like index $t$ did.

The NMF decomposition $\mathbf{V} \approx \mathbf{W} \cdot \mathbf{H}$ yields the coefficient matrix $\mathbf{W} \in \mathbb{R}_+^{n\times r}$, carrying the coefficients $\boldsymbol{w_i} \in \mathbb{R}_+^{1\times r}$ for each day $i$ and the component matrix $\mathbf{H} \in \mathbb{R}_+^{r\times 48}$, storing the modes $\boldsymbol{h_j} \in \mathbb{R}_+^{1\times 48}$. After the decomposition, the coefficients stored in $\mathbf{W}$ can be used as the target for an internal regression model.

As with all models, when fitting, the input time series of externals $\mathbf{X}$ and the target load time series $\boldsymbol{y}$ are first saved internally. In the next step, $\boldsymbol{y}$ is sliced into days and stacked into a matrix $\mathbf{V}$, afterward decomposed by an internal NMF. The coefficients $\mathbf{W}$ are stored, as they will later be needed as targets for the regression model, and the components $\mathbf{H}$ are saved as well.

For fitting the internal regression, a feature matrix carrying features $\boldsymbol{X_i^F}$ for each day is created by looking at the stored data as well as at the data $\mathbf{X}$ originally given to the model. The regression model then fits a multidimensional regression, given a set of daily features $\boldsymbol{X_i^F}$ as the input, and a set of coefficients $\boldsymbol{w_i}$ of day $i$ as the target.

Upon inference, the model receives a time series $\mathbf{X}$ of externals. First, a feature matrix $\mathbf{X^F}$ is created from $\mathbf{X}$ and the stored data. For each requested day $i$, the corresponding row $\boldsymbol{X_i^F}$ is then given to the regression model. Its output is a prediction of NMF coefficients $\hat{\boldsymbol{w}}_i$, from which a 48-point time series $\hat{\boldsymbol{v}}_i = \hat{\boldsymbol{w}}_i \cdot \mathbf{H}$ can be reconstructed by using the internally stored components matrix $\mathbf{H}$. This is done for all days $i$, for which $\mathbf{X}$ carried timestamps. To receive a prediction, the matrix $\hat{\mathbf{V}}$ is then flattened by concatenating the rows, yielding the time series $\hat{\boldsymbol{y}}$. Finally, $\hat{\boldsymbol{y}}$ is cut to only include the timestamps that were also requested by the index of $\mathbf{X}$ and output as the prediction of the model.

The NMF is implemented in Python by using the `scikit-learn` library.

### 7.4.3. Choice of internal NMF dimension

When performing a NMF decomposition, the internal dimension $r$ must be set beforehand. This choice of $r$ is a tradeoff between capturing enough variance of each day to allow exact predictions and keeping the model slim and interpretable at the same time. If $r$ is large and the coefficients are predicted well, the reconstruction gets closer to the actual day. But, at the same time, predicting more coefficients is potentially more difficult for

the regression model, as it also has to capture more variance in the original day. If a smaller $r$ is chosen, the model is slimmer and has to handle fewer coefficients. This is potentially easier for the regression model, but even if the coefficients for a requested day are predicted well, the reconstruction error will be larger than with a larger $r$.

To get a feeling for the influence of the internal dimension of the NMF decomposition inside the proposed model architecture, the entire data set will be decomposed and reconstructed with different choices of $r$. From the analysis of the corresponding reconstruction errors and the contained variance in the reconstructions, it can be proceeded with a reasonable choice for the inner dimension.

Furthermore, in order to represent such reconstructions in the model analysis, a perfect coefficient mock forecaster is constructed. This mock model still fits the internal NMF on the training data but gets fed the target time series $\boldsymbol{y}$ upon inference. The mock model slices the received target $\boldsymbol{y}$, decomposes it with the previously learned NMF to receive the actual target coefficients, and then reconstructs the time series right away with these coefficients. The prediction error of this perfect coefficient mock forecaster for each day is hence the reconstruction error of the internal NMF model for this day. By design, the performance of these mock models sets the lower boundary of forecasting errors of NMF models.

### 7.4.4. Drawing daily features from external data

To be able to predict on a daily basis, the entire data fed into the internal regression model must be daily as well.

Multiple daily features are drawn from the externals, starting with information about the day itself, like whether it is a working day, up to statistical measures like mean and standard deviation of different parts of the day (morning, midday, afternoon, evening). To create the set of features $\mathbf{X^F}$, saved past data ($\mathbf{X_{seen}}$ and $\boldsymbol{y}_{seen}$) and given externals $\mathbf{X}$ are used.

The externals are analyzed with daily statistical measures. For each given external time series (here only temperature), the data is also split into days. In these daily slices, mean, standard deviation, 5th percentile, and 95th percentile of the entire day are calculated. These percentiles are chosen, instead of the more commonly used minimum and maximum, to be more robust to outliers. Afterwards, the day is split into 4 equally large parts:

- morning (00:00 - 05:30 / data points 0-11)
- midday (06:00 - 11:30 / data points 12-23)
- afternoon (12:00 - 17:30 /data points 24-35)
- evening (18:00-23:30 / data points 36-47)

| | $X$-stat | | $X$-time-stat | | temporal | | $X$-stat lags | | $X$-time-stat lags | | $W$ lags | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $t$ | $temp_{q05}$ $(i)$ | ... | $morning_{mean}$ $(i)$ | ... | $is\_workday$ $(i)$ | ... | $temp_{q05}$ $(i-2)$ | ... | $morning_{mean}$ $(i-2)$ | ... | $w_0$ $(i-2)$ | ... |
| 2009-07-29 | 10.9 | ... | 11.7 | ... | 1 | ... | 12.3 | ... | 13.0 | ... | 11.0 | ... |
| 2009-07-30 | 9.8 | ... | 10.4 | ... | 1 | ... | 12.5 | ... | 12.5 | ... | 11.9 | ... |
| 2009-07-31 | 12.2 | ... | 12.4 | ... | 1 | ... | 10.9 | ... | 11.7 | ... | 10.8 | ... |
| 2009-08-01 | 12.1 | ... | 13.0 | ... | 0 | ... | 9.8 | ... | 10.4 | ... | 12.5 | ... |
| 2009-08-02 | 10.3 | ... | 10.9 | ... | 0 | ... | 12.2 | ... | 12.4 | ... | 11.2 | ... |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

(a) Part of $\mathbf{X^F}$.

| $t$ | $w_0$ | $w_1$ | $w_2$ | $w_3$ |
|---|---|---|---|---|
| 2009-07-29 | 10.8 | 4.2 | 4.0 | 5.9 |
| 2009-07-30 | 12.5 | 1.2 | 1.5 | 1.8 |
| 2009-07-31 | 11.2 | 3.4 | 3.0 | 8.2 |
| 2009-08-01 | 9.9 | 10.6 | 3.2 | 3.5 |
| 2009-08-02 | 8.7 | 14.5 | 3.6 | 2.7 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

(b) Corresponding part of $\mathbf{W}$.

Figure 7.7.: $\mathbf{X^F}$ coming out of the feature creation, together with the corresponding target $\mathbf{W}$. For this presentation, an inner NMF dimension of $r = 4$ was chosen.

| $X$-stat | | | | $X$-time-stat | | | | | | | | $temporal$ | | $X$-stat lags | | | | | $X$-time-stat lags | | | | | $W$ lags | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $temp_{q05}$ $(i)$ | $temp_{q95}$ $(i)$ | $temp_{mean}$ $(i)$ | $temp_{std}$ $(i)$ | $morning_{mean}$ $(i)$ | $morning_{std}$ $(i)$ | $midday_{mean}$ $(i)$ | $midday_{std}$ $(i)$ | $afternoon_{mean}$ $(i)$ | $afternoon_{std}$ $(i)$ | $evening_{mean}(i)$ | $evening_{std}(i)$ | $is\_workday$ $(i)$ | $sunshine\_dur$ $(i)$ | $temp_{q05}$ $(i-2)$ | $temp_{q05}$ $(i-7)$ | $temp_{q05}$ $(i-14)$ | $temp_{q95}$ $(i-2)$ | ... | $morning_{mean}$ $(i-2)$ | $morning_{mean}$ $(i-7)$ | $morning_{mean}$ $(i-14)$ | $morning_{std}$ $(i-2)$ | ... | $w_0$ $(i-2)$ | $w_0$ $(i-7)$ | $w_0$ $(i-14)$ | $w_1$ $(i-2)$ | ... |

Figure 7.8.: Columns of $\mathbf{X^F}$ coming out of the feature creation, with lags chosen to be $2, 7, 14$.

Then, for each of these parts, mean and standard deviation are calculated. These features will be referred to as `daytime_features`. The idea of looking at the different parts of each day is inspired by Hopf et al. [113]. Additionally, the same temporal features as for the shift-based models (i.e. the information whether the day is a working day and the sunshine duration at this day in seconds), are used as daily features. This group of features will be referred to as `index_features`.

And as the past data $\mathbf{X_{seen}}$ is saved internally in the model, all of the mentioned features can not only be created for the requested day, but also for past days, e.g. for 2 days or 1 week ago.

Similar to the shift features of the previous models, the values of past coefficients $w_{i-l}$ for certain lags $l$ are used as the main features.

Thus, in total, statistical features about the entire day and separate parts of the day are calculated for the target day and past days. Additionally, the features include information about whether the day is a working day and about the sunshine duration. Furthermore, the coefficient values of past days are used in $X_i^F$.

Figure 7.7 shows a representative part of the final created $\mathbf{X^F}$ and the corresponding part of the target NMF coefficients $\mathbf{W}$, on which the regression model will operate. Figure 7.8 shows the entire list of features included in $\mathbf{X^F}$.

### 7.4.5. Choice of proper lags

After looking at how information can be drawn from the external time series, information is added about how the past days have behaved. However, here not the original time

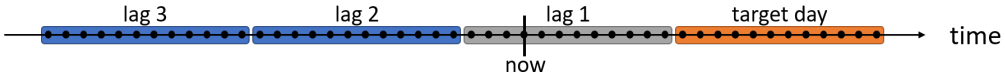



Figure 7.9.: Days referred to by the lags, as seen from the target day.

series $\boldsymbol{y}$ is shifted, but instead, it is looked at past values of the daily coefficients $\boldsymbol{w_{i-l}}$ for certain lags $l$. Therefore, the lags here are integer multiples of days and not a free time delta, as for the shift-based models.

In this scenario, to not cause confusion with the shifts of the shift-based models, slightly different notation is used. For the shift-based models, the shifts were relative to each time step $t$, including its innerday time information. E.g. $y_{2010-04-24\ 08:00}$ with a shift of $48h$ would result in $y_{2010-04-22\ 08:00}$. As the coefficients $\boldsymbol{w_i}$ in the NMF models refer to the entire day $i$, the lags will be denoted as simple integers, referring to the number of days the coefficients are shifted. E.g. $\boldsymbol{w_{2010-04-24}}$ with a lag of 5 would result in $\boldsymbol{w_{2010-04-19}}$.

The remaining question is: How can proper lags be chosen? As a first limitation, just like shifts under 48 hours must not be used, lags smaller than 2 are also not allowed. A lag of 1 is not allowed upon inference, as the day that is seen as lag 1 from the target day is usually the current day (also see figure 7.9). That means that the coefficients for this day cannot be calculated, as the data of the full day is not available yet. Lags of 2 and larger, however, refer to days that are already fully in the past. Their coefficients are therefore available and can be used as features for the prediction of the target day.

Thus, all lags $l \geq 2$ are generally allowed up to the start of data. But which ones make sense? For this, a detailed analysis can be performed by looking at autocorrelation function (ACF) and partial autocorrelation function (PACF) for each series of coefficients and by looking at the cross-correlations among the coefficients in the columns of $\mathbf{W}$.

### 7.4.6. Internal regression models

Just like for the shift-based models, many different features are internally created. With these features at hand, the goal is to predict the coefficients $\hat{\boldsymbol{w}}_i$ of the target day $i$. As each target, $\hat{\boldsymbol{w}}_i$ is of dimension $r$, a multidimensional regression problem must be solved for the NMF-based models, whereas the shift-based models only had to predict one-dimensional targets. Two kinds of models are applied to this task.

The first internal regression model is a Ridge Regression, i.e. a Linear Regression with $L2$ regularization. The regularization parameter is chosen from a range of $10^{-3}$ to $10^4$ and determined by 5-fold cross-validation. The second model used to predict the target coefficients $\hat{\boldsymbol{w}}_i$ is a Random Forest, grown from 100 individual trees with a maximum depth of 15. For each tree, at minimum 2 samples are needed for a split and only 1 sample for a leaf. Thus, the splitting is only limited by the maximum depth of each tree. The characteristic bootstrapping for random forests includes two sampling processes. First, for each tree $n$ samples are drawn with replacement. The tree is then trained on

these bootstrap samples. Here $n$ is the number of all available samples. Note that this procedure does not guarantee that all samples are used, but only that the number of used samples equals the number of total samples. The second bootstrapping happens at each node of a tree. Here, only 1/3 of the available features are considered when calculating the locally optimal split. Both Ridge Regression and RF Regression are implemented in Python by using the `scikit-learn` library.

With the internal regression models trained, one may predict the coefficients $\hat{\boldsymbol{w}}_i$ for target day $i$. From these coefficients, the 48-point time series $\hat{\boldsymbol{v}}_i = \hat{\boldsymbol{w}}_i \cdot \mathbf{H}$ can be reconstructed with the internally stored components matrix $\mathbf{H}$. This time series $\hat{\boldsymbol{v}}_i$ is then output as the load forecast of day $i$.

As the shift-based models are benchmark models, designed to be comparable to the NMF models, their regression models were chosen to match the ones used in the NMF-based models. Thus, the two regression models presented here are the same ones as described above in section 7.3.6. Nevertheless, it should be noted that for the shift-based models, the target was only a single, thus one-dimensional, time step.

# 8. Comparing models on an iterative basis

> Dankeschön. Wir sind die Cantina Band, wenn ihr Songwünsche habt ruft sie einfach! - Spielt den selben Song nochmal! - Alles klar den selben Song. Und los!
>
> Cantina Band

After describing the structures of different forecasting models their performance needs to be evaluated. This chapter lays out the iterative process by which the used models are being evaluated and compared.

## 8.1. Iterating models

Comparisons of forecasting models are usually performed in a classical approach of splitting the used time series into three parts: training, validation and test. The models are fitted on the training data, some parameter choices can be done by looking at the performance on the validation set, and the final results that will be published are computed on the previously unseen test set. This approach is established, well spread, and also used quite often when working with independent samples of tabular data or labeled images. However, when working with time series, the samples are usually not independent of one another. Thus, a random sampling of e.g. 70% of the data for training is not useful, as it does not resemble a realistic scenario. If a forecasting model is used in practice, the data given for training will, in most cases, be the data that is available up to this date, which will mostly be a continuous time series over a certain time range. Hence, it makes sense that if any comparison for models working on time series data is done, the training data is continuous and not sampled randomly.

But whereas many papers only compare model performances on a single value computed on one single test range, the approach used in this thesis is more detailed and results in multiple error measures. It allows for an evaluation of the models' performance over the course of the data set.

Each model is trained multiple times, instead of splitting the available time series $\boldsymbol{y}$ into three parts once, afterward training all models on this single chosen training block and evaluating them on the last part of the data. In this thesis, each model is trained repetitively on an increasing number of days of training data and evaluated on one week following the training data.

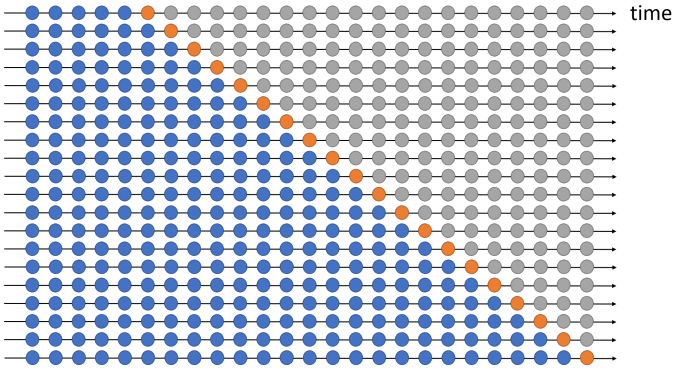



Figure 8.1.: Time series cross-validation for one-step forecasts, with blue dots marking the training data and the orange dots marking the targets (representation after [38]).

This procedure is similar to a procedure known as time series cross-validation [38]. For time series cross-validation, a larger number of training and corresponding test sets are used. The training sets only consist of observations prior to the ones in the test sets and have increasing lengths. The final forecast accuracy is then usually computed by averaging over the individual test set accuracies [38]. The general idea behind time series cross-validation is also illustrated in figure 8.1.

In the used approach, time series cross-validation is combined with "rolling" comparisons that keep individual error measures. Like in time series cross-validation, the amount of training data is increased at each iteration step. But unlike it, the performance measures are not simply averaged at the end, but their values at each cross-validation step are looked at in detail, similar to what Kuo and Huang did [91]. This in-depth analysis gives the opportunity to evaluate how the errors of models evolve with increasing amounts of training data. Such an analysis is especially helpful in the context at which this thesis aims, i.e. to find a model that works in the case where only a small amount of past data is available.

As far as known, such an extensive analysis of the length of training data and its influence on forecasting accuracy has not been published before.

### 8.1.1. Idea behind the iteration setup

Going back to the goal of this thesis, it must be kept in mind that when a new local power plant system is equipped with a measuring, control and regulation unit, usually nearly no past data is available. But as reliable forecasts are required early on, the models should not need a lot of training data to give accurate forecasts (also see section 2.3). Thus, to be close to a real-world scenario when evaluating the models, they should not receive a lot of data right away. Instead, their performance is evaluated in situations with only a small amount of training data. However, it should not only be analyzed how the models perform when being trained with little data, but also how they perform when

more training data is presented. Do they improve their performance significantly or do the performance measures saturate quickly?

Furthermore, in a real-world situation, a STLF model would not be used to predict more than a few days without adding new information. Hence, to train a model on e.g. 9 months of data and then predict the next three months without refitting the model is not something that would usually be done with a STLF model. Instead, only the next day or the next few days are predicted. To reflect this, only the load of seven days following the given training data is predicted.

Starting with little training data and increasing it iteratively, gives multiple options for analyzing the resulting forecasting errors. First of all, the error progression of a model over time can be analyzed. Secondly, one can identify times in the year that have unusual high errors and may also compare models in different training ranges instead of just one.

When doing these iterations, caution is needed. Usually, the data is split into three disjunct parts: Training, validation and test. In this setup, however, the data is split into training and test for each iteration. Though, each of these test sets is not a test set in a classical way, as it contains data already seen when developing the models. This cannot be avoided when doing such an iterative process during the development of models.

But a different design choice helps to minimize overfitting the models by information inference. In the results presented in this thesis, the iterations were done multiple times with different starting dates, in each case only using the data from this starting date on and dropping all data before. The used starting dates, the length of the time series from that starting date and the fraction of the total length are presented in table 8.1.

| Starting date | Length of time series from that starting date to the ending date 2010-12-31 | Fraction of total time series length |
|---|---|---|
| 2009-07-15 (development only) | 25681 observations | 99.817% |
| 2009-10-15 | 21265 observations | 82.653% |
| 2010-01-15 | 16849 observations | 65.489% |
| 2010-04-15 | 12529 observations | 48.698% |
| 2010-07-15 | 8161 observations | 31.720% |

Table 8.1.: Starting dates for model iterations, length of the corresponding time series and the fraction of the total 25728 observations contained in the load time series. The ending date of the series is always 2010-12-31.

The approach of using multiple starting dates comes with two advantages. The first

one is that a mean can be taken over the predictions done with these different starting dates. This way it can be seen how the models performed, e.g. with two months of training data, while being less prone to outlier error values being caused by weather, holiday, etc. not connected to the amount of training data. The second advantage is that the models can be developed by looking at a starting date that will not be used for the final results. The data setup using this single starting date can be seen as a validation setup. Calculating the final results by using the other starting dates can then be seen as using a test setup. In this case, the models were developed by only looking at the data starting from 2009-07-15. The results shown in this work are from the calculation of the later starting dates 2009-10-15, 2010-01-15, 2010-04-15 and 2010-07-15. This ensures that the final comparisons of the models are fair results, done on a data setup that was thus far unseen.

### 8.1.2. Iteration process

This next part explains the procedure followed in the iteration process. This process takes in all available data $\mathbf{X}$ and $\boldsymbol{y}$, as well as a model class and its given hyperparameters. The iteration is done with increasing amounts of training data and a constant length of test data as can be seen in figure 8.2.

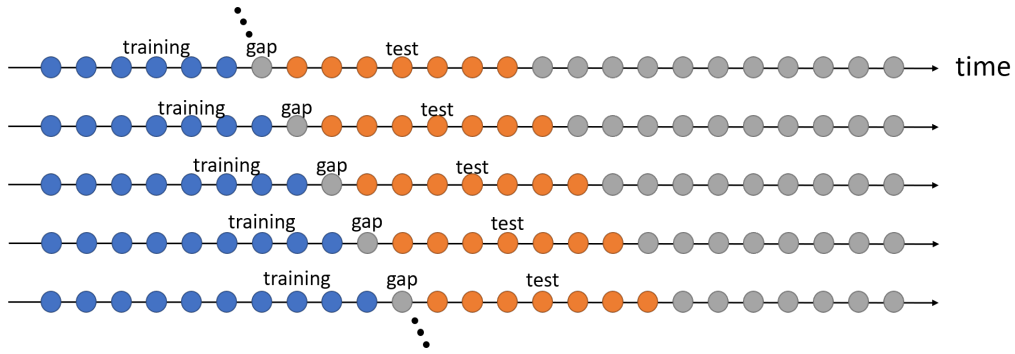



Figure 8.2.: Iteration process for model evaluation.

In each step of the iteration, a model from the given class with the given hyperparameters is instantiated. This instance is then fitted on $n_{days}$ of training data, starting with an initial value of $n_{days} = 1$. If the model cannot be fitted, e.g. because the model uses a lag of 14 days but only seven days of training data are given, this step in the iteration process is skipped and $n_{days}$ is increased by one.

If the fitting process was successful, a gap of one day is left out after the training data. This is done to avoid information inference, as the data of this day is usually not known in advance (also see section 7.1.2). The next seven days after this gap are used as test data. The model makes a prediction $\hat{\boldsymbol{y}}_{train}$ on the training externals $\mathbf{X}_{\mathbf{train}}$ and a prediction $\hat{\boldsymbol{y}}_{test}$ on the test externals $\mathbf{X}_{\mathbf{test}}$. Having predicted $\hat{\boldsymbol{y}}_{train}$ and $\hat{\boldsymbol{y}}_{test}$, performance measures are calculated by comparing the predictions to $\boldsymbol{y}_{train}$ and $\boldsymbol{y}_{test}$ respectively. After saving the

performance measures, predictions and some meta-information, the number of training days $n_{days}$ is increased by one.

This process is repeated until the training and test data would exceed the available data (from the given starting date to the last day in the data set). The iteration process is also presented in algorithm 1.

**Algorithm 1** Iteration process for model evaluation

$n_{days} \leftarrow 1$
**while** end of data set not exceeded **do**
    $\text{idx}_{train} \leftarrow Index(\text{first } n_{days} \text{ days of data})$
    $\text{idx}_{test} \leftarrow Index(\text{days 2-8 after idx}_{train})$
    $X_{train},\ y_{train} \leftarrow X[\text{idx}_{train}],\ y[\text{idx}_{train}]$
    $X_{test},\ y_{test} = X[\text{idx}_{test}],\ y[\text{idx}_{test}]$
    $model \leftarrow ModelClass(hyperparams)$
    **if** training data insufficient to fit model **then**
        $n_{days} \leftarrow n_{days} + 1$
        **continue**
    **else**
        fit *model*
        predict $\hat{y}_{train}$ and $\hat{y}_{test}$
        calculate error measures
        $n_{days} \leftarrow n_{days} + 1$

Inside this main iteration loop, one more loop is present. The prediction of $\hat{\boldsymbol{y}}_{\boldsymbol{test}}$ on the seven days of test data $\mathbf{X_{test}}$ is done day by day. After the prediction of each day, the model gets fed (not fitted on) the data ($\mathbf{X}$ and $\boldsymbol{y}$) of the day following the data that was already seen. This means that the first day fed to the model is the gap day, the next day is the first day of the test data, afterward the second day of the test data is fed, etc. This process is also illustrated in figure 8.3. The feeding of data is necessary for models that internally calculate features from seen data (such as shifts, lags, etc.). It has to be noted that as the target day in the test data changes in steps of days and the data is also fed to the model day per day, the gap between the data the model has seen and the current day stays present. This prevents unwanted information inference. The detailed prediction process done for each train/test setup is presented in algorithm 2.

After the predictions are done, the predictions, multiple error measures (see section 5.6) and some meta information are saved. This includes the length and dates of the training data, the dates of the test data, the models hyperparameters and the durations of training and predictions.

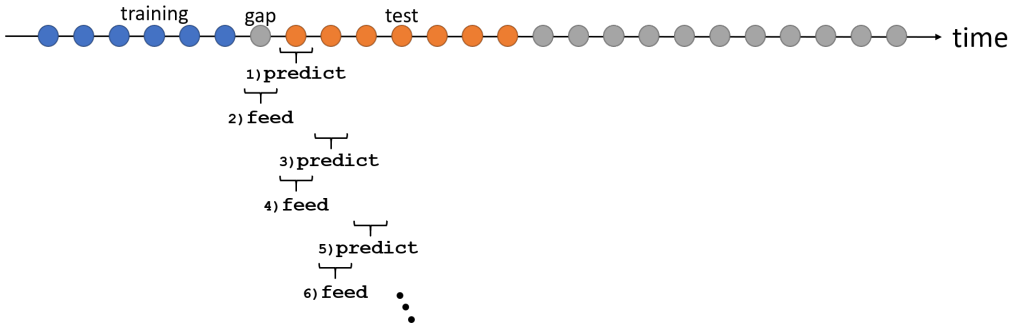



Figure 8.3.: Process for predicting the test set day by day, by feeding the data of the previous day once it is available.

**Algorithm 2** training and prediction process for each iteration step

$model \leftarrow model.fit(X_{train}, y_{train})$
$\hat{y}_{train} \leftarrow model.predict(X_{train})$
$err_{train} \leftarrow error\_measures(y_{train}, \hat{y}_{train})$

$preds_{test} \leftarrow [\,]$
**for** $i := 1$ **to** $7$ **step** $1$ **do**
    $\text{idx}_{target} \leftarrow Index(\text{day } i \text{ of idx}_{test})$
    $preds_{test}.append(model.predict(X_{test}[\text{idx}_{target}]))$
    $\text{idx}_{feed} \leftarrow Index(\text{day before idx}_{target})$
    $model.feed\_data(X[\text{idx}_{feed}],\ y[\text{idx}_{feed}])$
$\hat{y}_{test} \leftarrow concat(preds_{test})$
$err_{test} \leftarrow error\_measures(y_{test}, \hat{y}_{test})$

**return** $err_{train}, err_{test}$

# 9. Results

> Anyone hating any genre of music simply doesn't know the context in which it's meant to be enjoyed.
>
> Madeon

In this chapter the results of this thesis are presented, before being discussed in chapter 10. It starts with an analysis of the inner dimensionality of the data set, by looking at the contained variance after a PCA dimensionality reduction. Furthermore, the reconstruction errors of NMF decompositions, with different values for $r$, are analyzed.

Afterward, the focus is set on the models and their performance in the iteration setup. In this part, first, an analysis of the baseline models is done. Next, it is analyzed which shifts and lags are to be used in the more elaborate models, before also looking at the performance of shift-based and NMF-based models. Finally, the results of models of all categories are compared. In this chapter only the most important results of the done research are presented. More details and plots undermining the stated results can be found in the appendix A.2.

## 9.1. Inner dimensionality of the data set

When wanting to use NMF-based forecasting models, the inner dimension $r$ of the decomposition must be set as a hyperparameter. To be able to choose $r$, an analysis of the inner dimensionality of the data set is conducted.

First, the daily data matrix $\mathbf{V}$ is decomposed with a PCA. Looking at the cumulative contained variance in figure 9.1, we see that only three principal components are needed to contain 90% of variance and, with six principal components, 95% of variance are contained in the projected data.

However, PCA is not NMF, thus it is also looked at the effect that the choice of $r$ has on the NMF decomposition. By looking at the reconstruction error $\|\mathbf{V} - \mathbf{WH}\|_F^2$ for different values of $r$ in figure 9.2, we recognize that after four included NMF components the decrease in error per added NMF component flattens out. Additionally, when looking at the explained variance score (see eq. 5.30) in figure 9.3, it comes fairly close to 90% with just four NMF components. Using six NMF components, 90% of variance are contained in the reconstructed data. From this analysis, it can be concluded that both $r = 4$ and $r = 6$ are reasonable choices that will be used for the NMF-based forecasting models.

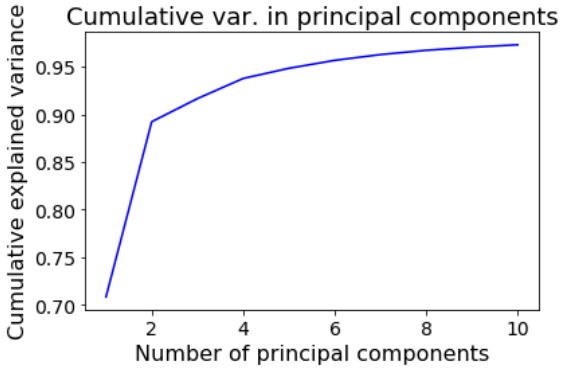



Figure 9.1.: Cumulative explained variance for the first ten principal components.

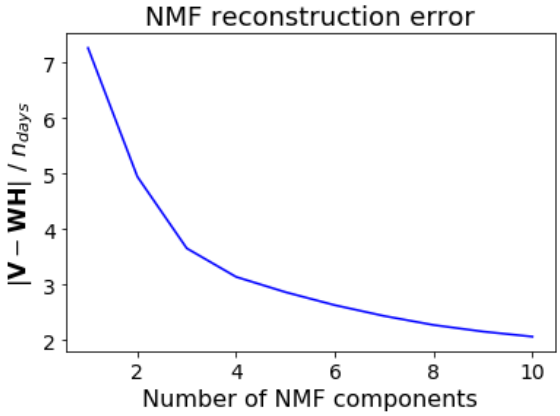



Figure 9.2.: NMF reconstruction errors $\frac{1}{n_{days}} \|\mathbf{V} - \mathbf{WH}\|$ for $r = 1$ to $r = 10$.

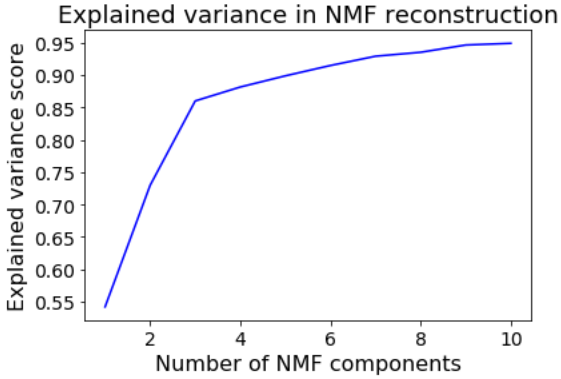



Figure 9.3.: Explained variance in the NMF reconstruction for $r = 1$ to $r = 10$.

It should be noted that all three figures 9.1, 9.2 and 9.3 only show the range from $r = 1$ to $r = 10$. This is, because with more than ten components, the interpretability of the individual NMF modes gets lost. Thus, a value of $r = 10$ is not exceeded for the potential candidates of the inner dimension. The NMF modes for $r = 4$ and $r = 6$ are shown in figure 9.4.

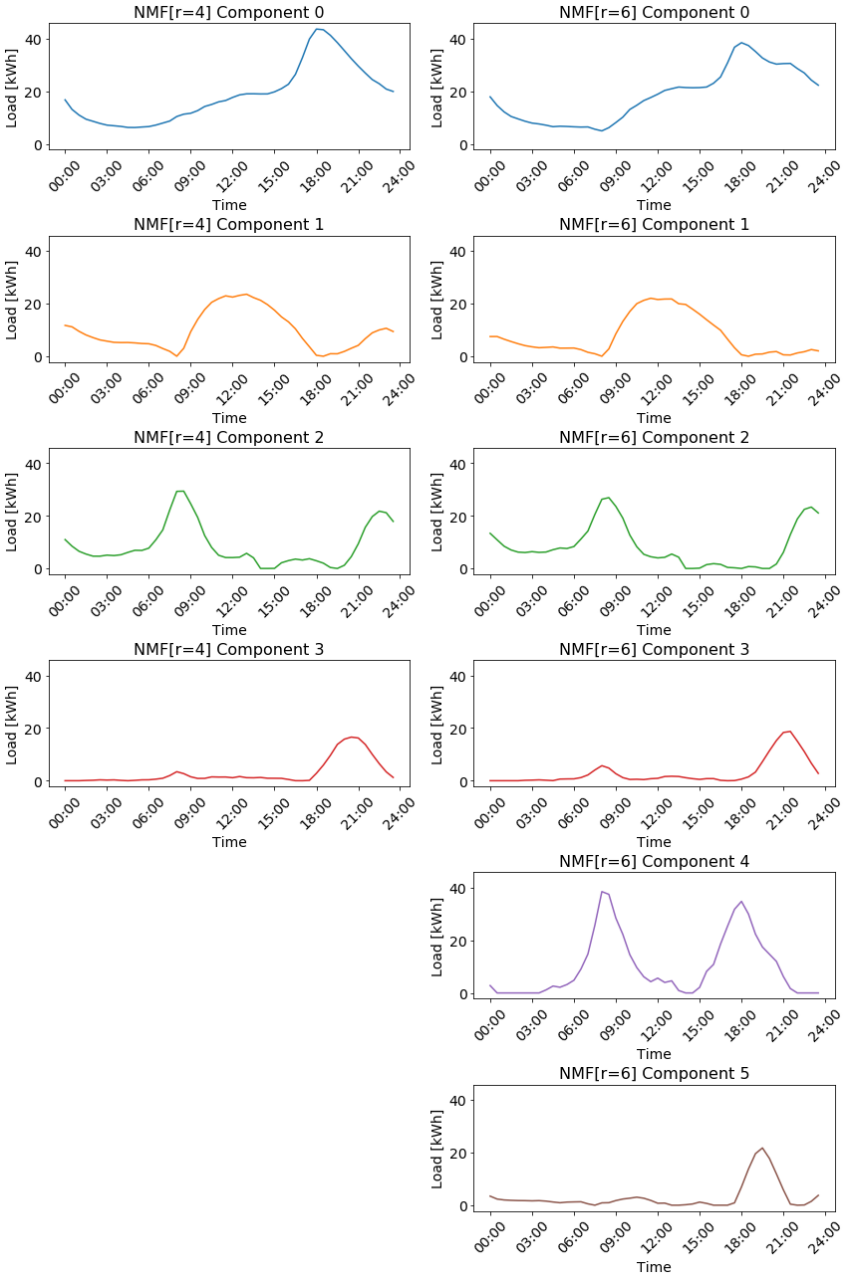



Figure 9.4.: NMF components (rows of $\mathbf{H}$) for inner dimensions $r = 4$ and $r = 6$.

## 9.2. Iterating baseline models

As explained in chapter 8, training models iteratively is important, when wanting to analyze how much data each model needs to be shown, to give accurate forecasts. It also enables to take seasonal bias into consideration.

In this section, the first results of the model iterations are presented by looking at the performance of chosen baseline models. These include the Repeat Day models explained in table 7.1, the smoothing models explained in table 7.2, the current state-of-the-art model N-BEATS and mock models based on NMF reconstructions.

For comparing the performance of these baseline models, it is looked at the models' forecasting accuracy over the number of training data points the models received. Each model was trained multiple times with increasing amounts of training data. This iteration process was repeated four times with different dates at which the start point of the training data was set. The four training data starting dates are 2009-10-15, 2010-01-15, 2010-04-15 and 2010-07-15. The following plots show RMSE values for each model, averaged over all four starting date setups, after having averaged the measures over all seven test days (the error measures were calculated for each day in the test set individually and then averaged).

Generally, the errors of the SLP models rise with an increasing number of training data points. In contrast, the performance of the Repeat Day Forecaster models is relatively constant, apart from a larger peak at around 3200 data points. We will see this peak of higher errors in the plots of all model errors. It is caused by the Christmas week 2009. Even though the peak's height is decreased by averaging over four starting date setups, it is still clearly visible. For the smoothing models, analyses show that the models perform better when leveraging days that are a week or more in the past, instead of only looking at the last few days.

Next, it is looked at the performance of one of the current state-of-the-art models in Time Series Forecasting: Neural Basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS).

Figure 9.5, shows that, overall, N-BEATS has a more noisy error curve and needs more data to reach lower RMSE values. Only after approximately 5500 training data points the errors are lower than the ones of the best Moving Average model MA[ws=7]. However, apart from the last two peaks, the errors are consistently higher than the ones of the Repeat Day baseline model.

Going into more detail, figure 9.6 shows the errors for each of the four starting date setups individually. We can see that after more training data has been given, N-BEATS is able to achieve errors as low as the ones of the Repeat Day model. Especially for the first two starting dates, the errors after October 2010 are consistently lower than or on par with the Repeat Day baseline.

As the analysis of the dimensionality of the data set showed, a value of $r = 4$ or $r = 6$ for the NMF-based models is a good choice. The question remains, whether it

makes sense to add some regularization on the NMF decomposition. To answer this, one may interpret the reconstruction errors of different NMF decompositions as forecasting errors of mock models. These can then be used as a lower bound of the NMF-based models' forecasting errors. For both values of $r$, the in terms of interpretability chosen $L1$ regularization parameter $\alpha = 400$ increases the error.

Overall, with all baseline models, the errors depending on the training data length are not constant but show some clear peaks.

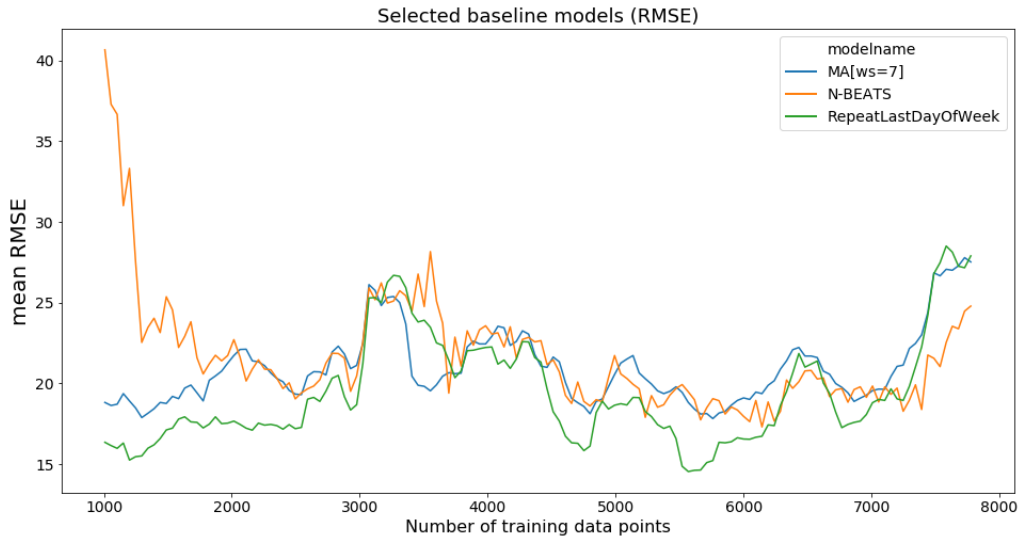



Figure 9.5.: RMSE of selected baseline models (calculated for each day in the test set individually and then averaged), averaged over all four starting date setups. 48 data points correspond to one day.

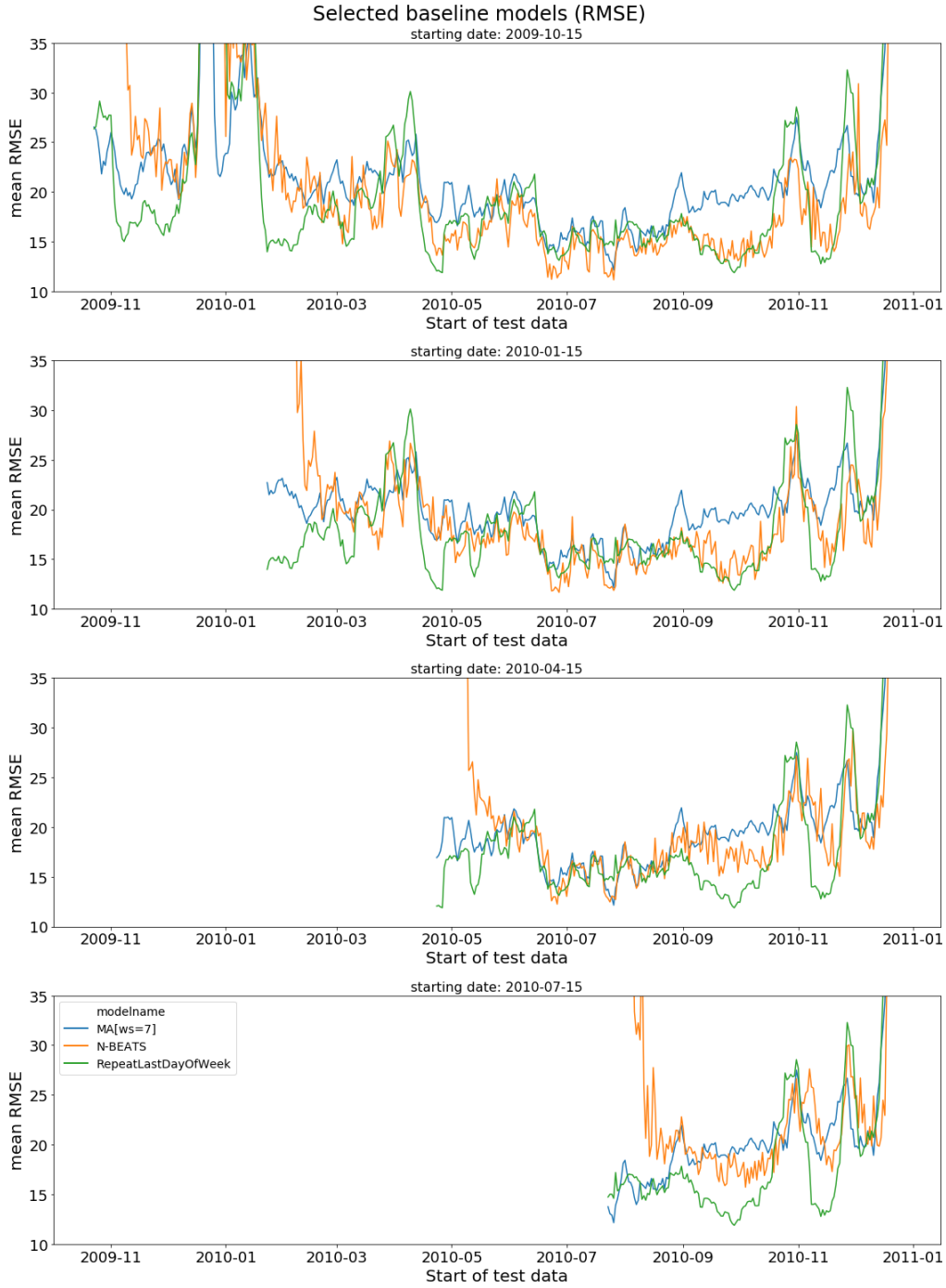



Figure 9.6.: RMSE of selected baseline models for all four starting date setups (full plots in appendix figure A.65).

## 9.3. Choice of lags

Before continuing the comparison between shift-based and NMF-based models, it needs to be evaluated which shifts or lags are to be used by these models.

For this, first, a Discrete Fourier Transform is applied on the load time series.

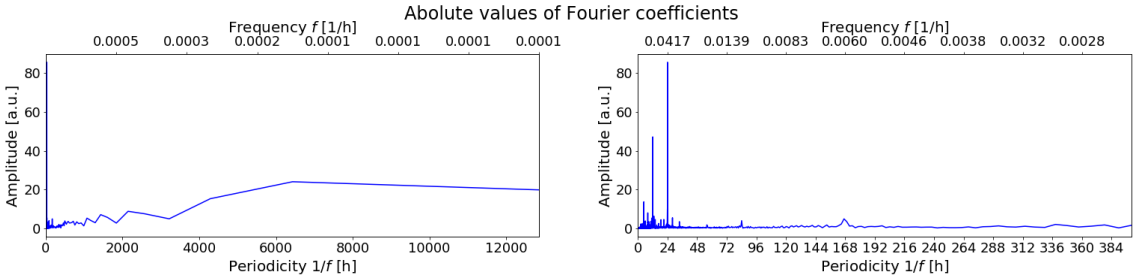



Figure 9.7.: Absolute values of Fourier coefficients after applying a Discrete Fourier Transform of the load time series. The right plot is a zoomed in part of the left one.

Figure 9.7 shows that the most prominent cycles are 24h and inner day cycles which are not allowed due to the 48h-rule. Some smaller magnitude peaks are visible for periods of 84h=3.5d and 168h=7d, however, in comparison to the higher frequencies they are almost negligible. In terms of choosing shifts above 48h, the Fourier Transform should, thus, be backed with a different analysis about shift importance, based on correlations.

For this, all days of the data set were stacked into a data matrix $\mathbf{V}$. This matrix was then decomposed by a NMF decomposition $\mathbf{V} \approx \mathbf{W} \cdot \mathbf{H}$. For the presentation here an inner dimension of $r = 6$ was used, as it is a number small enough to provide for interpretability, while still large enough to capture 90% of variance in the data.

After decomposing the data with the NMF, one can look at the autocorrelation function (ACF) and partial autocorrelation function (PACF) for each series of coefficients and also take the cross-correlations among the coefficient series in the columns of $\mathbf{W}$ into account.

Analyzing both, it can be concluded that the shifts/lags of 2, 6, 7, 8 and 14 days should be used for the forecasting models. The NMF coefficient time series (columns of $\mathbf{W}$) with these lags show a significant correlation towards the non-lagged series. Therefore, it can be expected that these lags carry information about the day that is to be predicted.

A detailed analysis is presented in the appendix A.2.2.

## 9.4. Iterating shift-based models

After having decided to use the shifts and lags of 2, 6, 7, 8 and 14 days, it can be continued with the comparison of model performances. The next models to analyze are the shift-based models, described in section 7.3. These models can be seen as baseline models directly trimmed towards the NMF-based models. This means that the shift-based model design is similar to the NMF-based one, but lacks the internal NMF decomposition. They, therefore, provide a way to analyze how much of the performance of the NMF-based models can actually be attributed to the decomposition.

Trying out a dense NN, a Random Forest (RF) Regression and a Ridge Regression on the shifts and additional features, it is shown that both Ridge Regression and RF Regression give smaller and more stable errors than the NN. Thus only these two are used in further analyses.

As different feature ideas were presented, their effect on the model performance will also be analyzed. Thus, a basic feature setup of shifts, target temperature, and temperature shifts was set. Then, the effect of adding further features, like statistical or index features, and the effect of removing the target and shift temperatures on the forecasting accuracy are analyzed.

Adding different feature combinations, we see that for the RF models the biggest improvement comes from adding the index features, and the best model is the one only using the index features in addition to the shifts and target temperature. For the Ridge Regression models adding more features on average does not help to reduce the RMSE. The most successful shift-based Ridge Regression model is the one operating on only the shifts, target temperature, and temperature shifts. Removing the target temperature led to worse performance for both regression variants.

In figure 9.8 we see the average RMSE of a selection of shift-based models with different feature combinations.

The errors for the individual starting date setups are presented in figure 9.9. From these plots, we can see that, with more training data, the RF based model leveraging the index features and target temperature gives lower errors than the Ridge Regression model operating on the basic features. The Ridge Regression only operating on the load shifts shows higher RMSE values than the other two and is plotted for comparison.

Overall, we see that the index features, including the `is_workday` information and the sunshine duration of the target day, boost the model performance when using a RF Regression, whereas the temperature lags are features not needed when using a RF Regression. RF and Ridge Regression, generally, both work well, with an advantage towards the RF models. A more detailed analysis of different feature combinations can be found in the appendix A.2.3.


Figure 9.8.: RMSE of selected shift-based models, averaged over all four starting date setups. 48 data points correspond to one day.

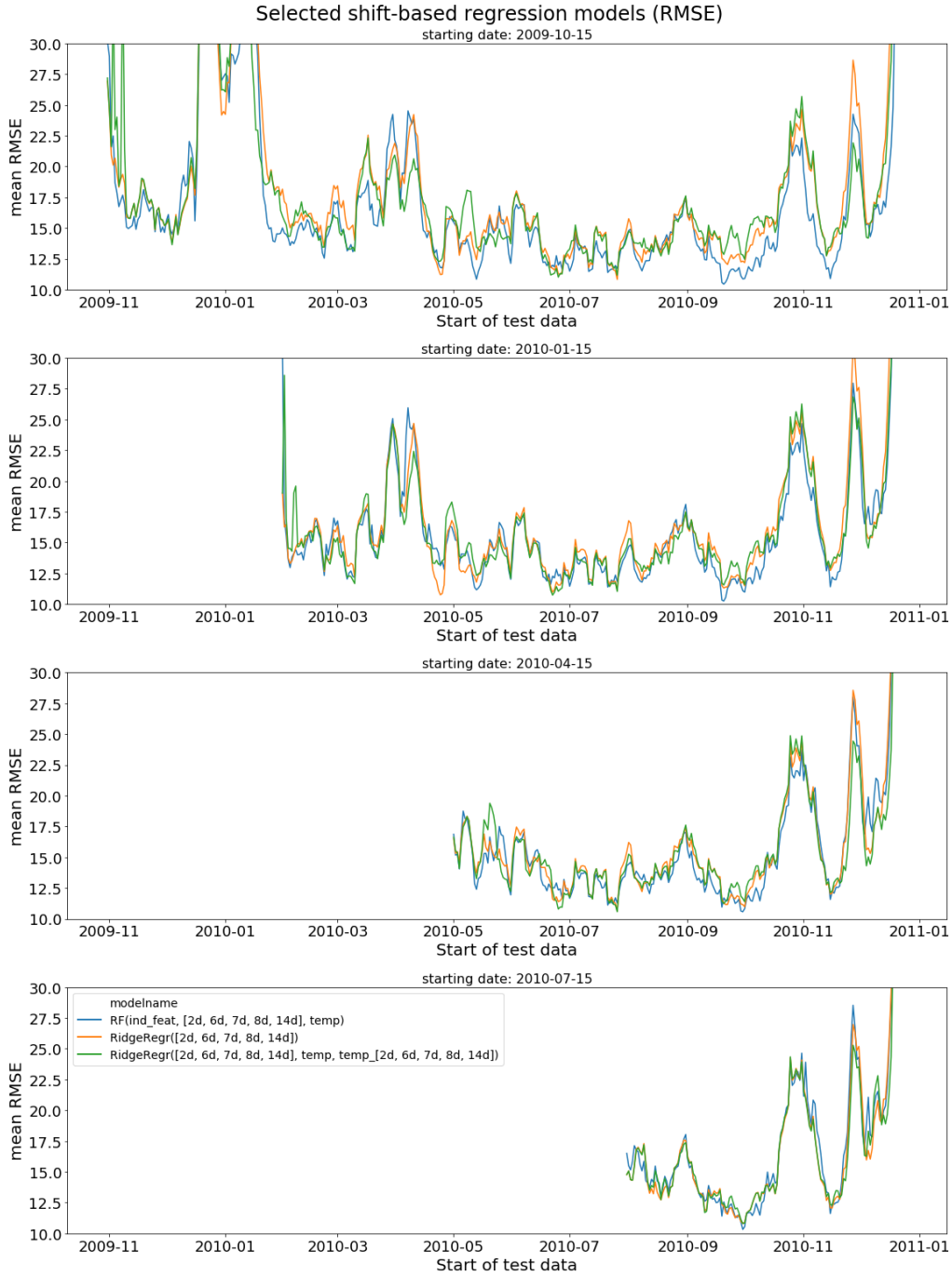



Figure 9.9.: RMSE of selected shift-based models for all four starting date setups (full plots in appendix figure A.67).

## 9.5. Iterating NMF-based models

After having looked at the different kinds of baseline models and shift-based models, next, the forecasting performance of the NMF-based models (described in section 7.4) are analyzed. For this purpose, it is again looked at the RMSE measure averaged over the seven days of test data, depending on the length of training data. In contrast to all prior Machine Learning models, the NMF-based models operate on a daily basis instead of predicting each time step individually. The used lags are 2, 6, 7, 8, 14, as determined in section 9.3, and $r = 4$ and $r = 6$ are used as inner dimensions.

First, a comparison is done of the performance of NMF-based models using a Random Forest or a Ridge Regression, both with $r = 4$ and $r = 6$. For this, the basic features of coefficient lags, target temperature, and temperature lags are used.

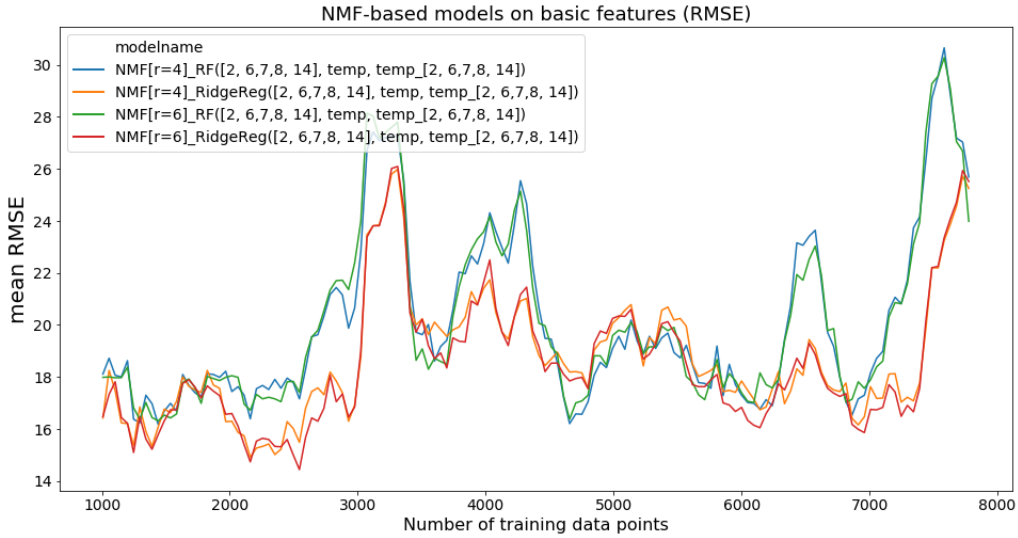



Figure 9.10.: RMSE of NMF-based models on basic features, averaged over all four starting date setups. 48 data points correspond to one day.

Figure 9.10 shows that with these basic features, for both values of $r$, the use of Ridge Regression as the regression part is superior to using a Random Forest. At the same time, we can see that when operating on the basic features, a value of $r = 6$ for the Ridge Regression gives a slight advantage over the use of $r = 4$.

Similar to the shift-based models, different feature combinations are tested on their effect on the forecast errors. It is again started from the basic features and then index and daytime features are added. Additionally, the performance changes when removing the target temperature or temperature lags are analyzed.

In figure 9.11, we see the average errors of a selection of NMF-based models with different feature combinations. Figure 9.12 shows the same models' errors for the four individual starting date setups.

When using the NMF model with $r = 6$ and applying a Ridge Regression on the basic features for comparison, reworking the feature selection decreases the error. Removing the temperature lags also helps to decrease the RMSE values, allowing for slimmer models. Additionally, adding the index features to the models largely decreased the errors as well. However, these error differences are smaller for the RF Regression and the RF models need more training data to make use of the index features. A more detailed analysis about the use of different feature combinations can be found in the appendix A.2.4.

Overall, using a Ridge Regression with the NMF models seems to be of a slight advantage, as the RF Regression is more prone to peaks that also occur in the NMF reconstruction mock models. Concerning the inner dimension of $r$, the errors are mostly similar for $r = 4$ and $r = 6$.

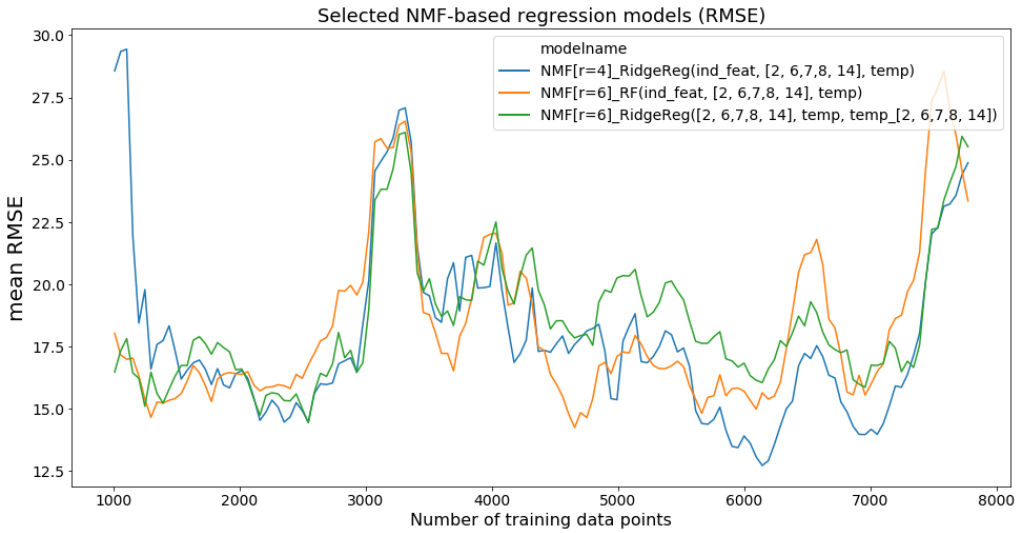



Figure 9.11.: RMSE of selected NMF-based models, averaged over all four starting date setups. 48 data points correspond to one day.

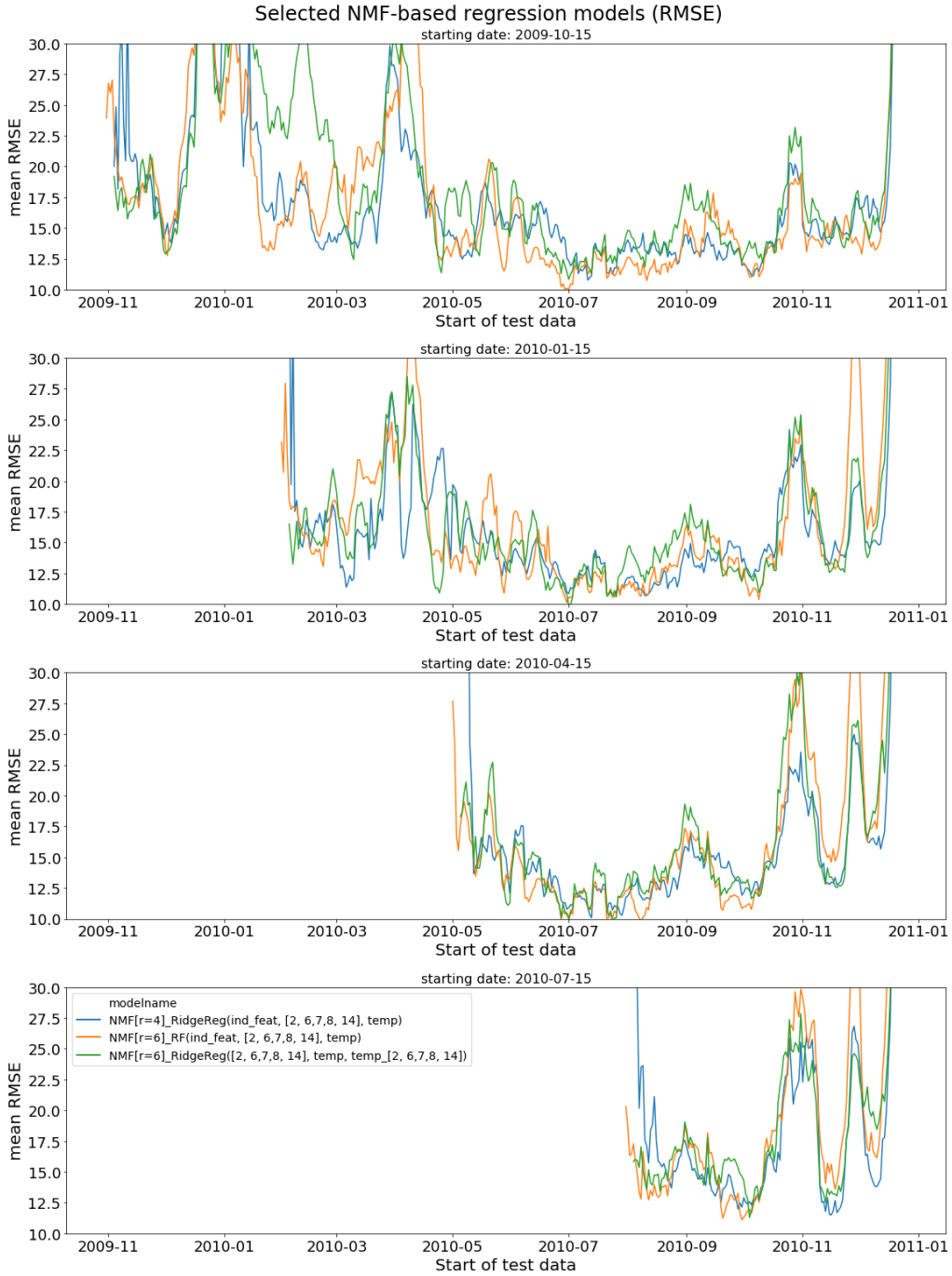



Figure 9.12.: RMSE of selected NMF-based models for all four starting date setups (full plots in appendix figure A.80).

## 9.6. Comparing model iterations from all model categories

After having looked at models from each category separately (baseline models, shift-based models and NMF-based models), in this section, their forecasting errors are compared to each other.

For this comparison, again the RMSE values averaged over the seven test days are considered. These errors are plotted over the length of training data for all four starting dates 2009-10-15, 2010-01-15, 2010-04-15 and 2010-07-15. Finally, it is analyzed if the errors differ across the seven days of test data.

### 9.6.1. Mean metrics over all test range

For the comparison, the most successful models from each model category are chosen. These are the RepeatLastDayOfWeekForecaster as simple baseline and the N-BEATS model as state-of-the-art comparison. Of the shift-based models, the one using a Random Forest operating on the shifts, target temperature, and index features is chosen. And of the NMF-based ones, the model using a Ridge Regression on the lags, target day temperature, and index features while using an inner dimension of $r = 4$ is chosen. A detailed analysis with more model comparisons can be found in the appendix A.2.5.

In figure 9.13 we see that both, the shift-based and NMF-based models, show lower errors than the baselines. Of the baselines, the N-BEATS model needs approximately half a year of data before reaching errors as low as the ones of the other models. The error curve of this state-of-the-art model is also noisier than other models' error curves.

Comparing the NMF-based and shift-based models, we see that, in summer, the NMF-based model gives lower errors than the shift-based one. It is also to be noted that both model types have peaks in the same regions. Overall, the NMF model starts with higher errors than the shift-based one but lowers to the same range when more training data is available. The model then even shows the lowest errors during the winter peaks at the end of the data set.

To be able to compare the models' performances to the ones of other works not using the same data set, the average MAPE values over all seven test days are calculated for these selected models. The same model can give MAPE values being as high as 14 and as low as 6, depending on the time of year of the test data and the length of the training data. Of all models in this comparison, the shift-based RF model operating on the shifts, target temperature, and index features, over all, gives the lowest MAPE values. The chosen NMF-based model gives good results for the first two starting dates, but higher errors than the shift-based one. The N-BEATS model consistently shows a higher MAPE with all starting dates.

More details about the MAPE values are presented in the appendix, including the average MAPE values for each starting date in table A.1.

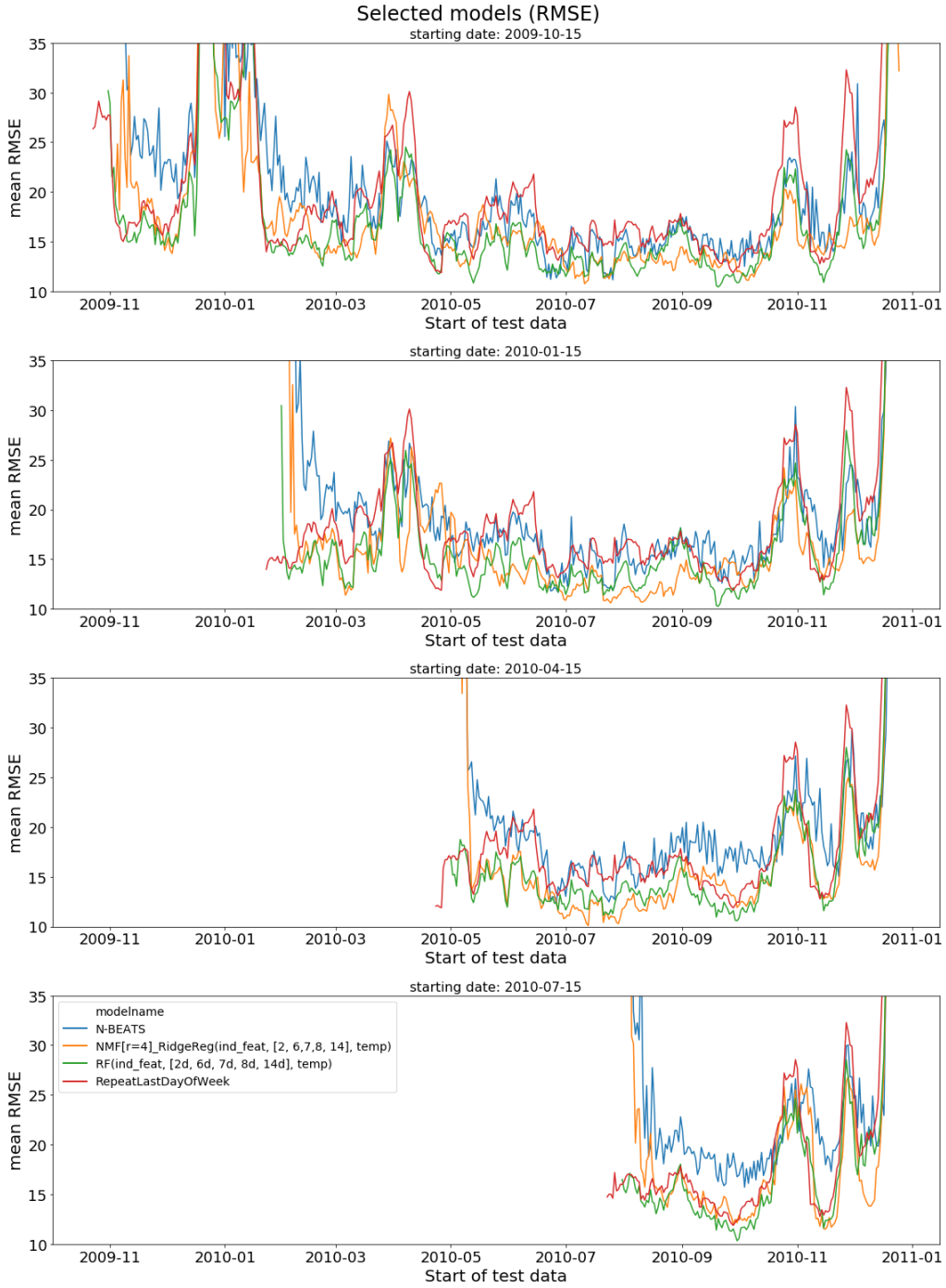



Figure 9.13.: RMSE of selected models for all four starting date setups (full plots in appendix figure A.81).

### 9.6.2. Metrics for each of the seven days of test range

All plots so far have shown the error measures averaged over all seven test days. For the models of the final comparison, it is also looked at at the RMSE for each of the seven test days individually, to analyze if any larger differences among the days occur during this week. Again, the errors are plotted over the training duration. As in this case, the magnitude of the error measures is less interesting than the relative differences among the individual days, the errors are averaged over all four starting date setups to get a more consistent picture.

In figure 9.14 we see several plots showing the RMSE values of the NMF-based model, using a Ridge Regression on the coefficient lags, target temperature and index features. Each colored line presents the Root Mean Squared Error (RMSE) of one test day over the length of training data. The plots for the individual days are all very similar, though shifted, with only slight differences in the heights of their peaks. Similar plots for other selected models can be found in the appendix A.2.5. The largest differences among the daily plots can be found for the N-BEATS model. For the RepeatLastDayOfWeekForecaster, apart from the shift, no differences occur.

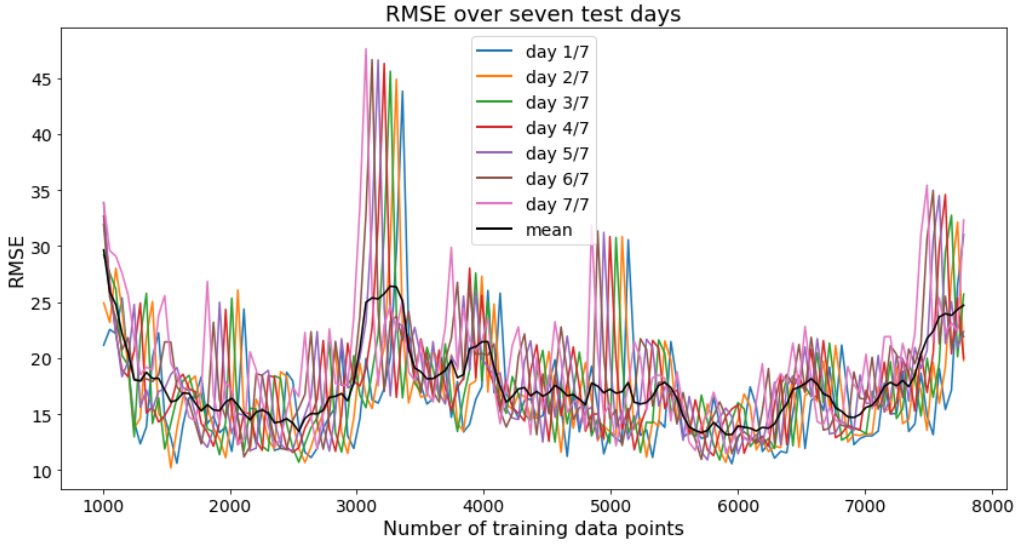



Figure 9.14.: RMSE of NMF Ridge Regression model with $r = 6$, and using lags, target temperature and index features, averaged over all four starting date setups, for each of the seven test days. 48 data points correspond to one day.

# 10. Discussion

> Many of the truths that we cling to depend on our point of view.
>
> Yoda

After presenting the results of this thesis in the previous chapter, in this chapter, they are discussed and interpreted.

First, the choice of hyperparameters is discussed. Afterward, interpretations for the results of the model iterations are given.

## 10.1. Choice of hyperparameters

The discussion of the results is started by looking at the chosen inner NMF dimension and at the shifts and lags that are used with the models.

### 10.1.1. Inner dimension

The first thing we see from the analysis about the inner dimension of the data set in chapter 9.1 is that, when looking at the contained variance, only 90% are captured with six NMF components, while with six principal components more than 95% of variance are contained (see figures 9.1 and 9.3). This is to be expected, as, by design, PCA is optimal for containing variance in the dimensionality reduction. The NMF with $r = 4$ even has an explained variance score of below 90%, however, the models perform well, justifying this selection. This can be explained with the idea that not all variance in the entire data set is explainable, but some proportion can be seen as noise. If the variance is not explainable, it is also not predictable and hence the models do not have to capture it. Nevertheless, while reducing the number of used NMF modes helps with the interpretation of the individual modes, dropping too much of the variance comes with some potential increase in errors, as can be seen from the performance of reconstruction mock models (see figure A.7).

### 10.1.2. Shifts and lags

Looking at the Fast Fourier Transform, the PACF values and at the cross-correlations suggests the use of the shifts/lags of 2, 6, 7, 8 and 14 days. One may therefore say that most information from the lags comes from either a weekly seasonality or a weekday-weekend-relation. Having looked at the data analysis in chapter 6, this is not surprising.

The 3.5-day periodicity found by the Fourier Transform is closely related to the 7-day one, which was also recognized (see figure 9.7). Furthermore, the highest absolute value of the Fourier coefficients can be attributed to the 24-hour periodicity, strongly underlining the value of daily slicing.

In addition, the PACF values of lags 6 and 8 surrounding lag 7 (see figure A.15) are likely caused by the relation of weekend days and normal working days. This connection can also be attributed to the lag of 2 days. Worth noticing is that, even though it was looked at the PACF and not the ACF values, lag 14 gave an extra contribution. Thus, we see that looking back more than one week gives information that is not already present in one week of past data.

Looking at the cross-correlations of the NMF coefficients in detail (see figures A.16 and A.17), we recognize effects caused by the different load shapes of weekends compared to working days.

Overall, the importance of weekly seasonality and weekday-weekend-relations are the most important effects to look out for in the data.

## 10.2. Iteration of models

After having trained more than 130 000 individual model instances, a discussion of the iteration results is inevitable. Thus, the model performances of the simple baseline models, N-BEATS, the NMF reconstruction mock models, the shift-based models and NMF-based models are discussed here. Afterward, the model comparisons are commented and a short summary of the findings is given.

### 10.2.1. Simple baseline models

Starting with the baseline models, we saw that the errors of the simple SLP models rise over time (see figure A.9). This makes sense, as an unweighted average is taken over longer periods, leading to less influence of recent changes.

The Repeat Day models showed results that highlight the importance of considering the weekends, as including weekday information into the models led to a major increase in performance. By contrast, the difference between RepeatLastDayOfWeekForecaster and RepeatLastWeekdayWeekendForecaster was relatively small. So treating all seven days individually is an option, but the more important information is to see whether a day is a weekday or not.

Where the Repeat Day models support the claim that weekends are important, the smoothing models showed that information about the future is not only contained in the last few days. For the Moving Average models, a window size of 7 gave the best results and larger window sizes of 14 or 21 also achieved lower errors than window size choices of below one week. Similar results can be found for the Exponential Smoothing models, where focusing only on the most recent past did not lead to good results. This goes well together with the decision to use lags of up to 14 days (see figures A.11 and A.12).

### 10.2.2. N-BEATS

Comparing the baseline results with the performance of the N-BEATS model, on a first look it might not look like N-BEATS performed well (see figure 9.6). But given that this model must predict 48 hours ahead without knowing anything about a daily seasonality in advance, its performance is good. It was able to keep up with a good smoothing model, that, by design, operates on daily slices. These similar errors showcase how important the daily slicing is as a method, as it enables simple models to keep up with one of the current state-of-the-art TSF methods. Even though N-BEATS needs about one year of data to be able to get errors as low as the ones of the Repeat Day models, this was to be expected, as many parameters have to be fitted in its NN design. From the results we have seen, N-BEATS can be expected to perform even better with more data. Additionally, using an appropriate ensemble strategy, as was done in the original paper, is suggested. However, training large ensemble models iteratively would have exceeded reasonable computation times even with GPUs available. Nevertheless, even without using ensembles, N-BEATS is a highly competitive model once enough data is available.

### 10.2.3. NMF reconstructions

The analysis of the NMF reconstruction mock models revealed some interesting aspects about the data (see figure A.8). First of all, the decomposition already works well on little amounts of data. It does not need multiple months of training data to be able to properly reconstruct the seven days of test data. This can be understood such that the variations in the data from one week to the next are usually not too big. However, a few peaks in the error graphs show that some changes in the data appear, which can not be captured with the previously trained NMF modes. These peaks are present with all starting date setups at the same dates and, thus, were likely caused seasonally. Therefore, it could be expected that these peaks also appear with the NMF forecast models, as is the case (see figure A.50).

The peaks at Christmas are easily explainable, as the model was simply not trained to capture very high loads in its modes and the highest load demand of the year appears on Christmas day. However, while the peaks appearing at the Daylight Saving Time changes can be explained with errors that possibly appear by using more energy at night, the size of these peaks is unexpected, as the models operated on data in local time and, thus, no extra modes have to be fitted like in a UTC case.

Looking at the reconstruction mock models, generally, gives a good outlook on the errors of the NMF-based models and helps to only look at the contribution of the decomposition individually.

### 10.2.4. Shift-based models

Next, the results of iterating the shift-based models are discussed. It can be seen that when using a Neural Network as regression for the shift-based models, the results

showed some high peaks (see figure A.18). This indicates that the used NN models were unstable. This is likely because in the applied algorithm no advanced strategies for controlling the learning process were used. The high errors were caused by bad fitting results that can happen when many parameters have to be fit. While with better fits the NN can keep up with RF and Ridge Regression, it also does not beat them. This is an indicator that sometimes the usage of simpler regression models can be an advantage.

As for the other two regression models, both RF and Ridge Regression give satisfactory results. The Ridge Regression technically should be able to achieve smaller errors in unseen load ranges, as the RF can only predict values in regions it has already seen. However, such an effect is not present in the data. Both models show comparable error peaks.

Given the previously presented results, it was to be expected that the index features increase the model performance, as they explicitly include information about days being workdays. This increase in performance is present in the results (see figures A.19 and A.20). That the statistical features do not help to reduce the errors could be caused by the number of additional features. Possibly, adding these features results in too many features for the model to work with. On the other hand, it could also be that the model internally figures out a way to rank the load values on its own, making the statistical features obsolete. Especially for the RF models, finding optimal split points in the data follows a similar idea, possibly making the statistical features unnecessary.

While the temperature shifts also could be too many features for the models, a second, more likely, explanation of why they do not help to decrease the forecasting errors is that the carried information was not needed. If the temperature shifts can only be used to give information about the time of day, this would already be implicitly encoded in the shift features. By looking at the effect of using temperature features, we can see that using the target day temperature, in contrast to the temperature shifts, contributes to lower errors (see figures A.21 and A.23). This seems logical, because the target temperature contains new information, whereas the information contained in the temperature shifts can possibly be captured in the load shift features. It has to be noted that the RF models can make better use of the temperature features, which might indicate that the temperature effects can be better modeled with a non-linear function, of which the Ridge Regression is incapable.

### 10.2.5. NMF-based models

Looking at the results of the NMF-based models we, first of all, see that the choice of $r = 6$ over $r = 4$ only very slightly gives lower errors (see figure 9.10). This is an indication that, based on the given features, the future load can only be predicted up to a certain variance level that is already captured with four coefficients.

With the shift-based models, both Ridge Regression and Random Forest Regression perform similarly, whereas with the NMF-based models the Ridge Regression mostly shows lower errors. This is most likely is due to the range restriction of RF. By design RF

is not able to predict values outside the range of already seen values. With NMF-based models this is a larger drawback as for the shift-based models, as it limits the predictions for multiple coefficients. Looking at the results, RF, generally, seems to be able to better and more consistently leverage features like the index features (see figures A.32 and A.34). With the Ridge Regression, some models could not use these to their advantage. One possible reason for this is that Ridge Regression can only model linear dependencies and the effect of index features on the NMF coefficients is not necessarily linear.

Looking at the feature choices it can be seen that including the index features again mostly helps. However, it cannot be answered if only the workday information helps or if the sunshine-duration also contributes to learning about seasonal effects. A more detailed analysis of this is open for future work. Adding the daytime features did not help. This can be explained by the idea that information about the past values at certain time ranges was already included in the coefficient lags.

Similar to the shift-based models, it looks like temperature features of lags do not carry additional information that can be utilized by the NMF-based models (see figures A.40 and A.42). One interpretation of this is that, similar to the daytime features, the main information about past values is already implicitly carried in the coefficient lags. The use of target day temperature is useful, but the performance increase is not large. While it seems that including weather data can help, the amount by which the performance increases is likely limited by data quality. Considering that the used weather data was averaged over many weather stations across Ireland, the temperatures are not local to the household locations. However, in practice, local weather data is accessible and might give more of an improvement for the forecasts.

Comparing the NMF-based models to their reconstruction mock model baselines, we see that the mock models show errors clearly lower than the one of the actual forecasting models (see figure A.50). The error difference between the NMF reconstruction and model errors represents the error caused by missing the exact coefficient values in the prediction. Looking at the plots, we see that the error peaks of the NMF-based models occur in the same regions as for the reconstruction mock models, so the error attributed to the coefficients stays more or less constant.

Unfortunately, looking at the high-error regions of the NMF-based models in the first starting date, it is difficult to say if the models have trouble fitting when the training starts in fall, or whether data in spring is generally harder to predict (see figure 9.12). To gain information about this interesting effect, a detailed analysis with more data at hand would have to be done.

### 10.2.6. Model comparison

Overall, the results of the model iterations show that the general idea of leveraging shifts with learned weights seems to work well, as they outperform the smoothing models using fixed weights on the shifts. While the shift-based models give relatively low errors right away, the NMF based models need more data to be able to give low forecasting

errors (see figure 9.13). This makes sense as these models operate on a daily basis, so the amount of training samples is drastically decreased. For example, when the NMF models only have 50 daily samples of training data, the shift-based models can train on $50 \cdot 48 = 2400$ samples. Nevertheless, with more training data the combination of NMF decomposition and Ridge Regression manages to keep up with the errors of the shift-based RF models and even shows the lowest errors in early fall for the first starting date and in the summer months with the second and third starting dates. Thus, the NMF-based models prove to be promising and it would be interesting to challenge the models in a scenario with more data available.

However, while the results for the first starting date setup look promising, the NMF-based models are not ideal for a situation with only little training data and better options, like the shift-based models or even the Repeat Day models, are available. Especially the Repeat Day approaches proved to achieve high accuracy, even when only one week of data is available. With their simplicity they are easy to understand and provide a great baseline that is not beaten easily, showing that models do not have to be highly complicated if they manage to leverage data characteristics.

For the error plots of each individual test day, the differences among the daily plots are small (see figures 9.14, and A.61 - A.64). The error curves for the RepeatLastDay-OfWeekForecaster are even equal for all seven days, as in all cases the load of one week ago was used as forecast. N-BEATS is the only model in this analysis where the seven days of test data show larger differences in their errors. This might indicate that in the used model configuration, with only a single model instance, the results of N-BEATS are relatively noisy, suggesting to use ensemble strategies. But as with most models, the error differences among the test days are fairly small, it can be argued that the forecasting error depends more on the actual target day usage and less on when this day appears in the test range.

This is a desirable result, as it shows that, at least for the range of seven days, it makes no difference how long the model has not been fitted anew. Nevertheless, seven days is a relatively short time and from the done research no information can be retrieved about which length of a test horizon works well for the models.

Looking at the average MAPE of selected models for the four starting date setups in table A.1, we see that the presented models give highly competitive errors. While the N-BEATS model achieves an average MAPE between 9.62% and 11.81%, the best shift-based model gives errors as low as from 7.21% to 7.59%. Even the Repeat Day model achieves an average MAPE in the range from 8.20% to 8.64%. While the focus in this thesis is not on such averages, these are competitive values. This again highlights how strong even simple baseline models can perform with daily slicing as preprocessing.

### 10.2.7. Summary

Summarizing the findings, it can be said that using a NMF-based model is not necessarily an improvement for the prediction, though the data suggests performing an analysis with more data. Nevertheless, NMF proves to be a useful tool to analyze data and find good lags that also work well with shift-based models. The main advantage of the shift-based models is that they operate point-wise and can, therefore, make use of data more quickly than the daily NMF-based approach. While both of these model categories were created specifically for the STLF problem at hand, N-BEATS was not tailored to this problem. However, even without using ensemble techniques, it still proves to be a good model, once enough training data has been provided.

The results in this thesis suggest that daily slicing is a simple yet effective preprocessing step and that one should not simply use NN models because it is today's flavor of doing so. Even the simple Repeat Day or smoothing approaches give good results without the use of any modern ML. Especially with only little data available, this can be an advantage. With a good sense for how Repeat Day models work, comparing forecast results of other models to the ones of such naïve methods helps to put the errors of other models into perspective.

From the seen results, it can be suggested that, when only little data is available, simpler models, like the Repeat Day or shift-based models, should be used. Additionally, a comparison pipeline should be built to compare the used models' results to the ones of other model structures. This way, when more data has been collected, the used model can be changed once more elaborated models beat the simple ones.

One large advantage of simple models is that they can easily be implemented, trained and run without the use of heavy GPU computations. In theory, they, thus, could even be run on an edge device.

As we can see from the seasonal effects, transition phases of Daylight Saving Time and holidays should always be monitored with care. Larger errors should be expected at these events and thus special models should be used. These could then e.g. also take in information about such events in other similar household situations, as hardly sufficient training data for these extreme events is available.

# 11. Conclusion

> The road goes ever on and on.
>
> J.R.R. Tokien, The Hobbit

This thesis aims to find an answer to the question of how to achieve high forecast accuracy in Short Term Load Forecasting (STLF) tasks, using electricity load data at a local aggregation level, measured in sub-hourly intervals. The focus is laid on the situation where not much data has been collected and it is evaluated what effect the amount of available training data has on the model performance. Additionally, the models were only allowed to use data that would be available in a real-world setting.

The research shows that while, for many models, the amount of training data plays a critical role, simple models, leveraging the idea of daily slicing, yield satisfying results, even when only little data is available. Including information about working days in contrast to weekends and holidays can help to further improve the performance. By looking at NMF reconstruction models it was shown that the variance contained in the forecasts can already be captured in only a few NMF modes. However, leveraging this with NMF-based forecasting models only gave promising results when more training data was available.

By iteratively increasing the training data and using multiple training start dates, it could be shown that forecast errors heavily depend on seasonal effects. Therefore, several test ranges should be used when comparing models.

For a practical application in a setting where not much data has been collected yet, simple forecasting models strongly relying on the daily periodicity and information about workdays are recommended. Although the Repeat Day approaches are primitive, they can be used with as little as one week of data. The presented shift-based models should be used once a few months of data have been collected. Once significantly more data is available, a detailed comparison should be made to see if these models can be replaced with more sophisticated models that require more data to train properly.

This thesis contributes to the scientific discourse in multiple ways. First of all, it presents a framework for iterating the model training and evaluation process with an increasing amount of training data. This allows for further research on the effects of training data length. Using multiple starting dates for the iterations presents a simple yet effective method to showcase that many error effects are caused seasonally and not by the lack of training data.

It was shown that even simple models can achieve low forecasting errors. While some models need a lot of data to reach their full potential, the simple shift-based models gave good results even with only little data available. Furthermore, with the NMF-models a promising new model design was presented. Lastly, the great importance of features carrying information about working days was shown.

These contributions are significant as usually no detailed analysis about the influence of training data length is done in the field of STLF.

Like all research, this work has some shortcomings and limitations. While the amount of training data was extended in the iteration process, no statement can be given about how long of a test range can be forecasted without refitting the models. Additionally, the models were only tested on a single aggregation level of 350 households. To generalize to an even more local setting, experiments on lower aggregation levels would have been necessary as well.

The largest limitation to this thesis was the availability of data. While the used data set features thousands of individual households, the data only covers 1.5 years. A longer time range would have been helpful, especially to learn more about the presented NMF-based models. Additionally, not knowing the exact locations of the households was another limiting factor, as no detailed weather information could be used.

Lastly, a few suggestions on what future research might cover are presented: Firstly, it would be interesting to see how the models perform on even lower aggregation levels. Secondly, using ensembles of some of the presented models could increase the performance. Thirdly, as the NMF-based models showed promising results once more data was available, analyzing these models on a different data set spanning a longer time range is also suggested. Furthermore, for these NMF-based models, a detailed analysis of the errors in the forecasted coefficients could help to uncover more about the structure of forecast errors. Especially in the context of over- and under-estimations at certain times of the day, looking at the coefficient errors could be useful. Additionally, with the recent developments in CNNs for the use with time series, it would be interesting to see the performance of such models operating on the daily NMF coefficient series.

Overall, this thesis showed that simple forecasting models using daily slicing are very powerful, even when only little data is available. In the context of the energy transition, these models can help to integrate local structures into the power grid more quickly and thus support the transformation to a more modern and greener power system.

# A. Appendix

## A.1. Figures supporting the data analysis

This section of the appendix shows some additional figures, complementing the data analysis of chapter 6.

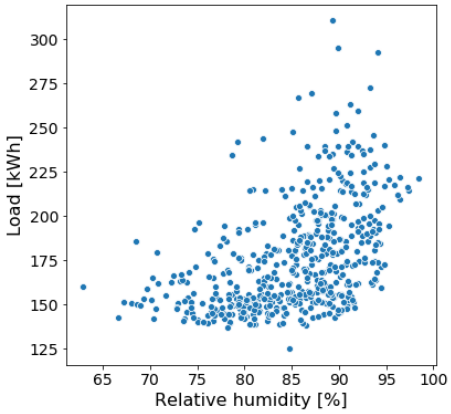



Figure A.1.: Scatterplot of the average daily load vs. the average daily relative humidity.

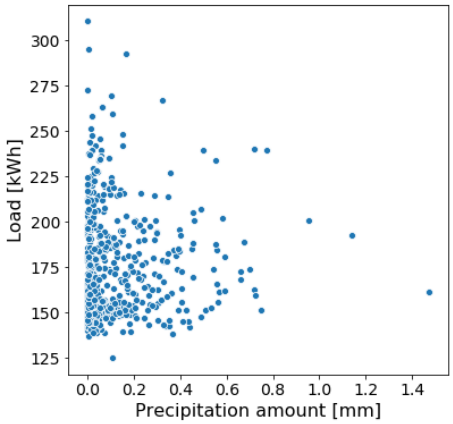



Figure A.2.: Scatterplot of the average daily load vs. the average daily precipitation amount.

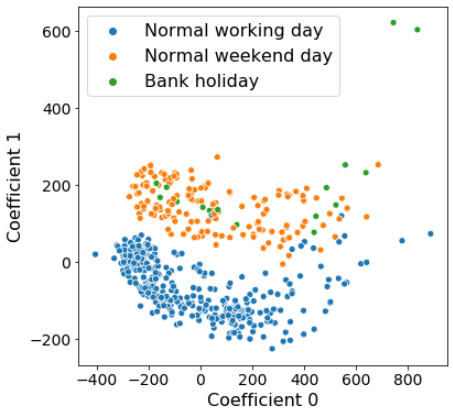



Figure A.3.: Weekends and holidays marked in PC 0 vs. PC 1 scatterplot.

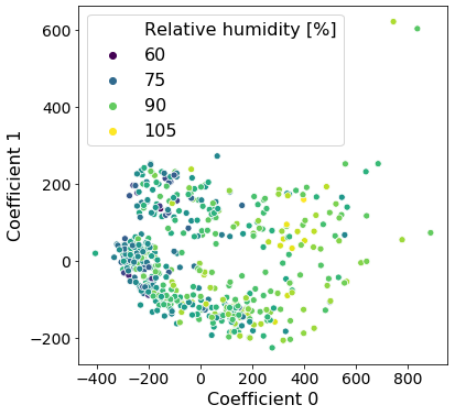



Figure A.4.: Relative humidity colored PC 0 vs. PC 1 scatterplot.

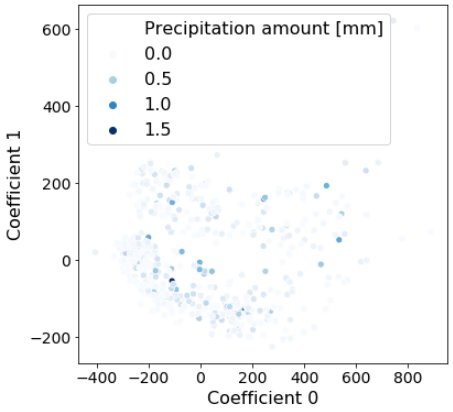



Figure A.5.: Precipitation amount colored PC 0 vs. PC 1 scatterplot.


Figure A.6.: $K$-means clusters over time with weekends and holidays marked.

## A.2. More detailed results

In chapter 9 the main results of this thesis were already presented. However, not all details and error plots could be shown there. Therefore, this section of the appendix is designated for including more details of the results, though with less description. The structure of this section is the same one, as in chapter 9.

### A.2.1. Iterating baseline models

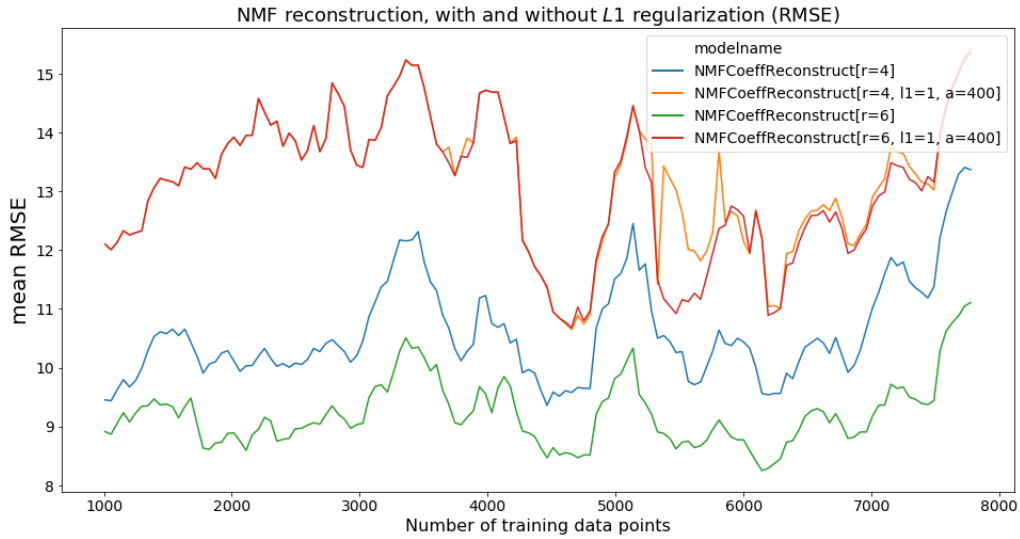



Figure A.7.: RMSE of NMF reconstruction mock models, averaged over all four starting date setups. 48 data points correspond to one day.

Figure A.7 shows the effect of regularization on the reconstruction mock models. We can see that, for both $r = 6$ and $r = 4$, adding the, in terms of interpretability chosen, $L1$ regularization parameter $\alpha = 400$ increases the error. With regularization, the error for both used internal dimensions $r$ is very similar. Without it ($\alpha = 0$), the reconstruction

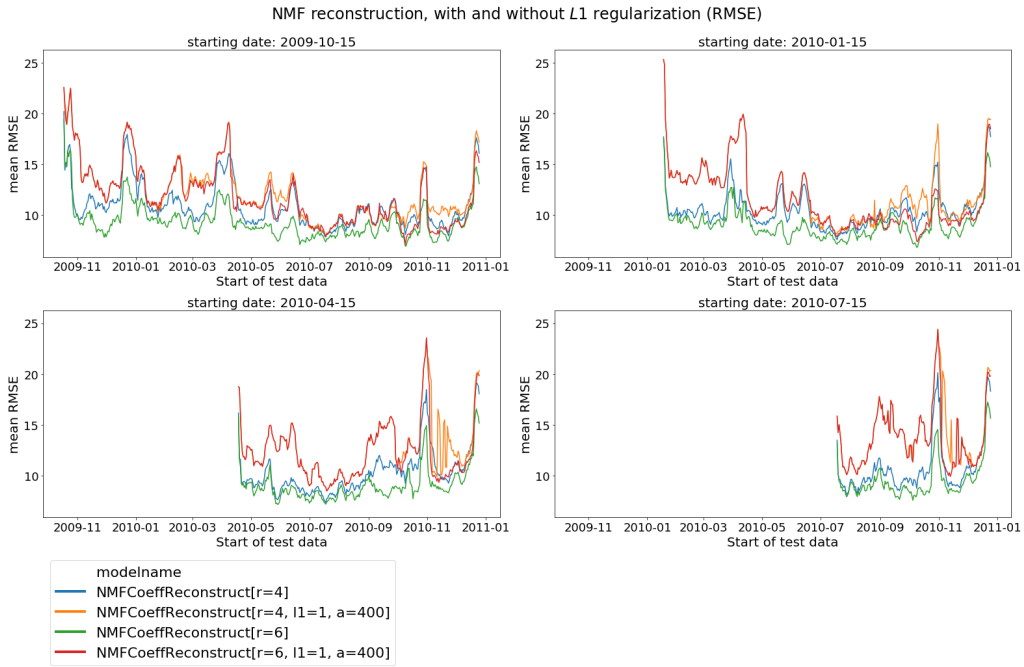



Figure A.8.: RMSE of NMF reconstruction mock models for all four starting date setups.

error is lower for $r = 6$.

Figure A.8 shows the RMSE values for each starting date of selected NMF reconstruction mock models. Comparing the plots of the individual starting dates we see that the higher error peaks are at the same times throughout the year, including Christmas and the DST changes.

In figure A.9 we see the RMSE of the Repeat Day and SLP models, averaged over the seven days of each test data range and averaged over the four starting dates. We see that, generally, the errors of the SLP models rise with an increasing number of training data points. In contrast, the performance of the Repeat Day Forecaster models is relatively constant. Comparing the Repeat Day models to one another, we see that RepeatLastDayOfWeek and RepeatLastWeekdayWeekend outperform RepeatLastDay. However, the two models using weekday information show similar performances.

Next, we look at the performance of the used smoothing models. From figure A.10 we see that for of the models using simple Exponential Smoothing or Moving Average, the choice of hyperparameters is critical. The Exponential Smoothing with $\alpha = 0.9$ shows higher RMSE than the smoothing with $\alpha = 0.1$. Also, the Moving Average models with a window size of 7 or 15 outperform the Moving Average with window size of 3. In both cases, the models giving larger weights to days in the further past perform better than those only looking at the last few days. Additional hyper parameter choices for both Exponential Smoothing and Moving Average models can be found in figures A.11 and A.12.

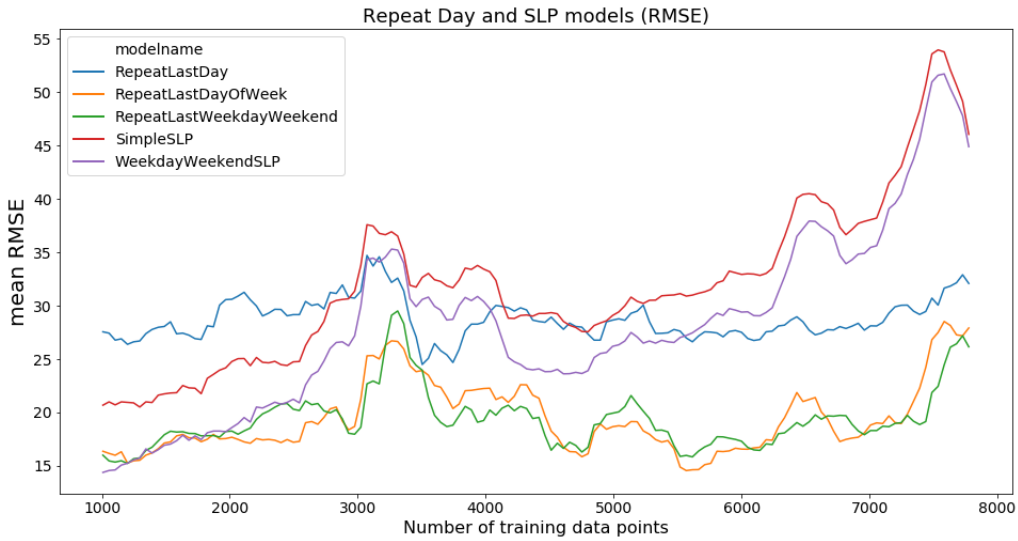



Figure A.9.: RMSE of Repeat Day and SLP models, averaged over all four starting date setups. 48 data points correspond to one day.

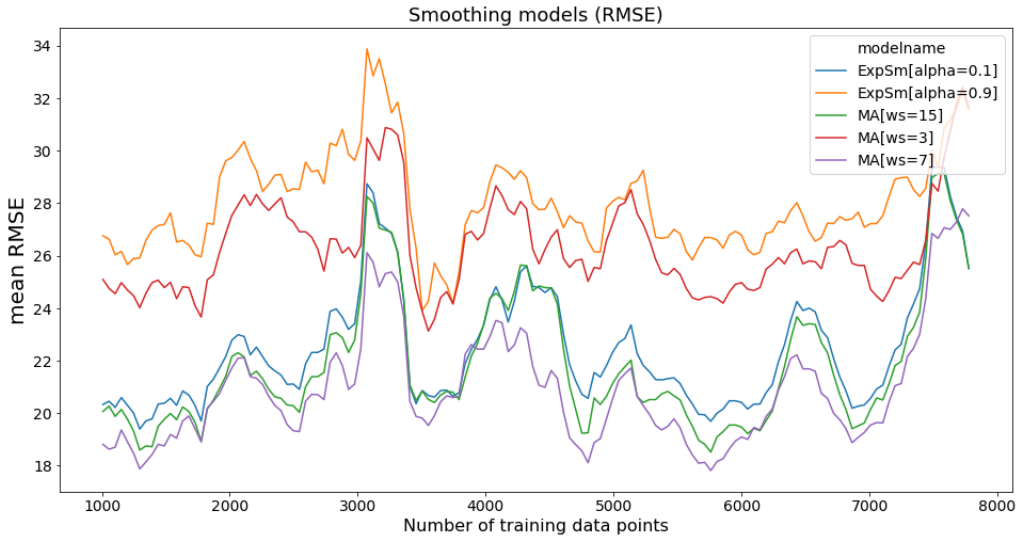



Figure A.10.: RMSE of smoothing models, averaged over all four starting date setups. 48 data points correspond to one day.

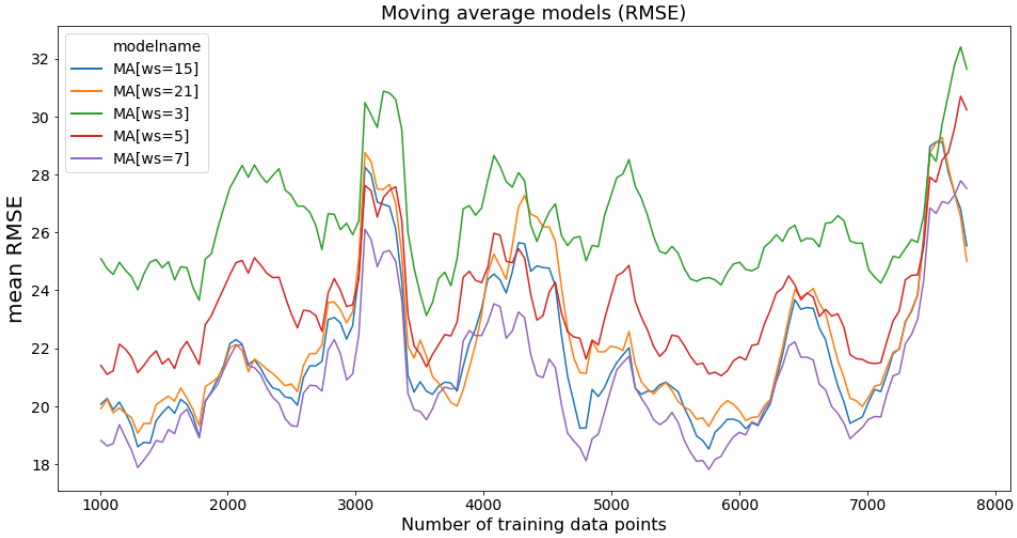



Figure A.11.: RMSE of Moving Average models, averaged over all four starting date setups. 48 data points correspond to one day.

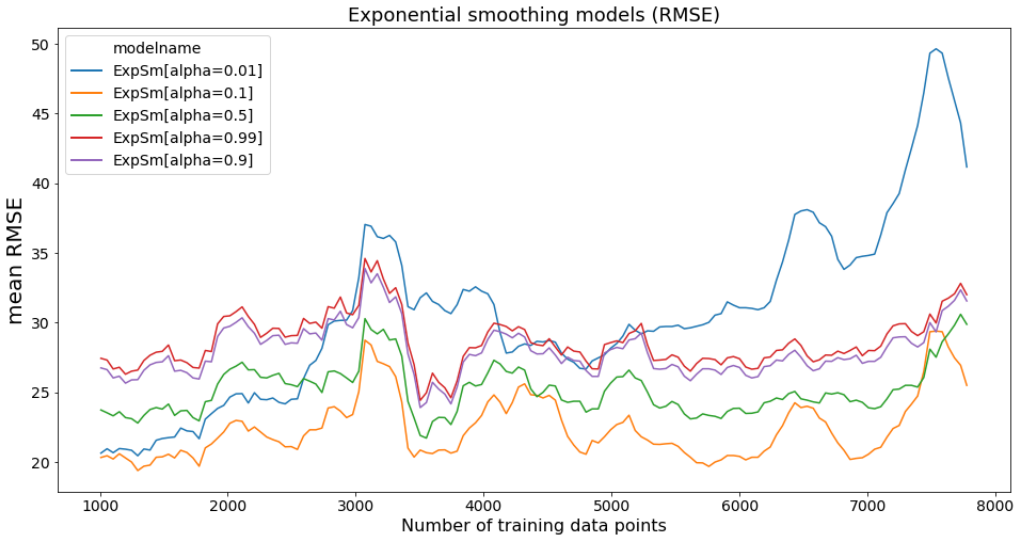
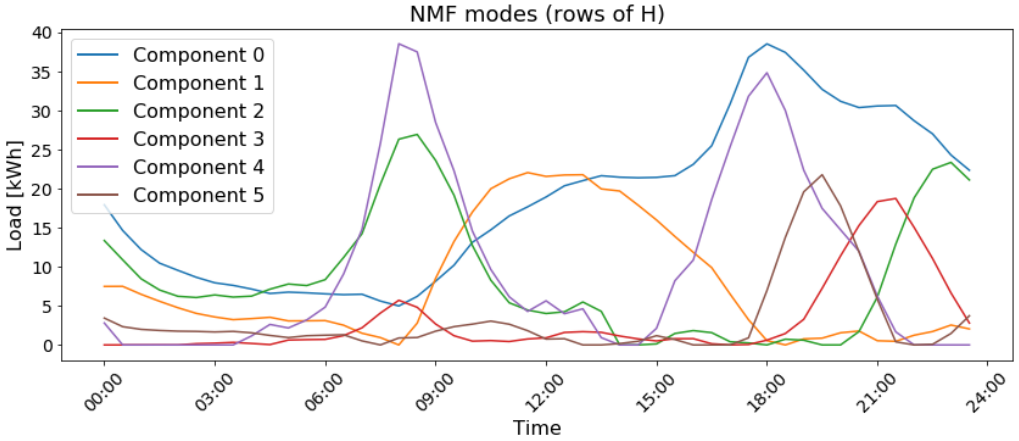


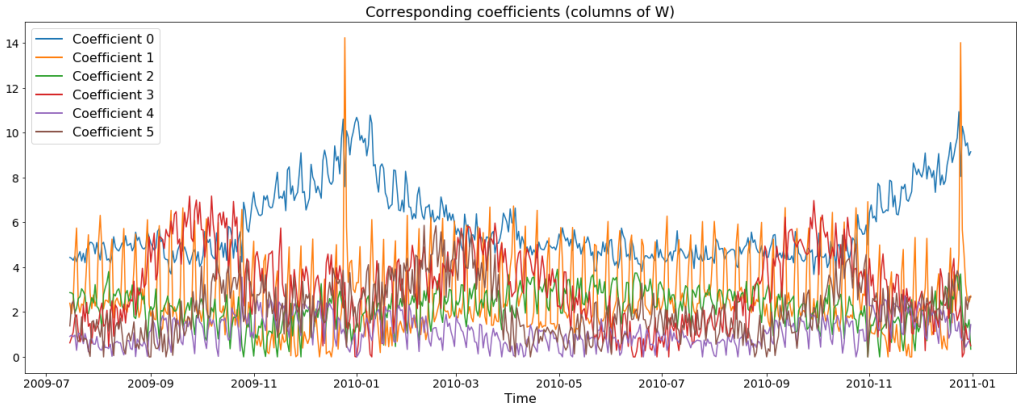

Figure A.12.: RMSE of Exponential Smoothing models, averaged over all four starting date setups. 48 data points correspond to one day.

### A.2.2. Choice of lags

For the calculation of the correlations of NMF coefficients, the larger one of the options $r = 4$ and $r = 6$ was used, as potentially more correlations among the lags of the different components can be found. The NMF components (rows of **H**) and corresponding coefficients (columns of **W**) of this decomposition are shown in figure A.13.




(a) Components (rows of **H**) found by the NMF




(b) Coefficients (columns of **W**) over time

Figure A.13.: NMF decomposition with inner dimension of $r = 6$.

Figure A.14 shows plots of the ACF and PACF for each individual column of **W**. The ACF and PACF plots show the lags from 0 up to 28, referring to daily time shifts of up to four weeks. The blue area in each plot marks the 95% confidence interval. As in the ACF plots, this interval is naturally increasing with a larger number of lags, most of the analysis is based on the PACF plots. We see that the PACF, generally, is strongest for lags of up to 7 days. Afterwards, it drops and after 14 days only a few lags show a PACF value, still in the confidence interval.

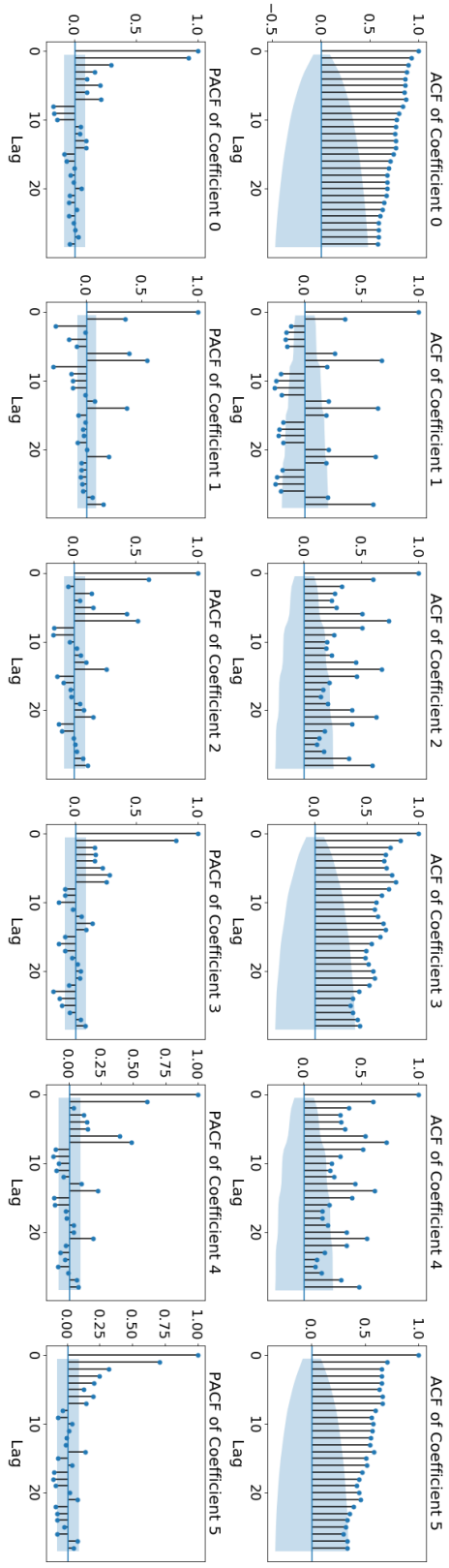



Figure A.14.: ACF and PACF plots for columns of $\mathbf{W}$ with $r = 6$.

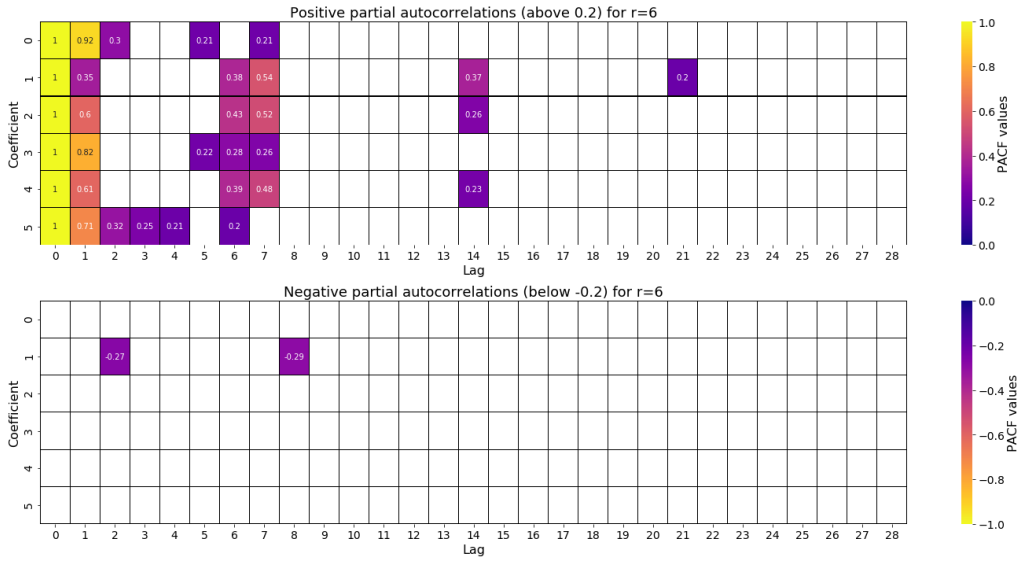



Figure A.15.: PACF values with absolute values above 0.2 for columns of $\mathbf{W}$, for inner NMF dimension $r = 6$.

In the coefficients for component 0, we find a larger positive PACF for lags of 1-7, 13 and 14 and a larger negative PACF for lags 8, 9 and 10. For coefficients 1, the PACF shows positive values for lags of 1, 6, 7, 21 and 28. Negative values outside the intervals are present at lags 2, 4, 5, 8, 9, 10 and 11. The PACF of the series of coefficients for component 2 shows significant positive values at lags 1, 3, 5, 6, 7, 14, 21 and 28. Negative values are significant at lags 8, 9, 15, 22 and 23. For component 3, the coefficients show positive values in the PACF for lags 1-7, 13, and 14 . Negative values are present for lags 10, 15, 16, 17, 23, 24 and 25. Positive values of the PACF for coefficient 4 are found at lags 1, 3-7, 13, 14 and 21. Negative values are found at lags 8, 9, 11, 15 and 16. For the coefficients of component 5, we have positive PACF values exceeding the confidence intervals for lags 1-7 and 14 and larger negative PACFs for lags 17, 18, 19 and 22.

Collecting the PACF values of all 6 coefficient series and dropping all with an absolute value below 0.2 gives a more clear picture presented as a heatmap in figure A.15. In the upper part, all positive PACF values above 0.2 are shown, whereas the lower part includes all negative ones with a value below $-0.2$.

In this representation, it can be seen that lag 1 is the most important one. However, it will not be used in this thesis, because of the 48h-rule. Furthermore, the first week mainly shows positive PACF values, whereas lag 8 exposes a negative correlation value in coefficient 1. After lag 9 the appearances of values with magnitudes above 0.2 thins out. Only lag 14 is of importance for single components, and lag 21 shows positive autocorrelations in component 1 that also had the strongest positive value at lag 14.

All in all, it can be said that, after the first week, the most important lags are the ones that are connected to a weekly periodicity in the data. As thin models are preferred, lags largely exceeding two weeks will not be used for the creation of features in the models.

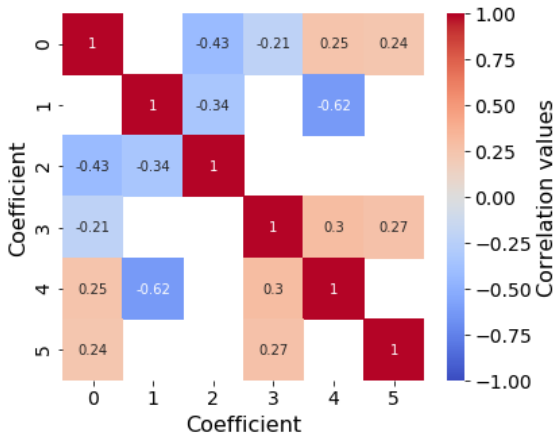



Figure A.16.: Cross-correlations with absolute values above 0.2 among columns of $\mathbf{W}$ for $r = 6$.

A first conclusion can be drawn, stating that it is reasonable to use the lags 2, 3, 4, 5, 6, 7, 8 and 14 as features for the models.

However, as no separate models for each coefficient are fitted, and as the coefficients are not independent of each other, we also look at the cross-correlations of the coefficients. Figure A.16 shows how the series of coefficients are correlated with each other for an inner dimension of $r = 6$. To focus on the higher correlations, only the correlation values with an absolute value of 0.2 or higher are shown. The most prominent cross-correlations are between the coefficients 1 and 4, 1 and 2, 0 and 2.

To receive more information, the cross-correlations with the coefficient series lagged by lags from 2 to 17 were calculated as well. Figure A.17 shows the cross-correlations that have an absolute value above 0.2

First of all, we see that coefficient 0 shows a strong positive correlation with its own lags and a slight one with the lags of coefficients 4 and 5. A negative correlation is present with the lags of coefficient 2. Coefficient 1 hardly correlates at all with the lags. Only a slight correlation with some lags of coefficient 2 is visible. The only bigger contributions are at lags 7 and 14 of itself. For coefficient 2, the bigger positive correlations are with itself for lags 7 and 14 and smaller ones are present for other lags with itself and coefficient 1. But even smaller negative correlation values can be found for all other coefficients. Coefficient 3 mainly correlates positively with its own lags and negatively with lags of coefficient 2. Additionally, some correlations with lags of coefficient 4 can be found. Coefficient 4 shows negative correlations with the lags of coefficient 2 and some positive ones with coefficients 0, 3, 5 and itself. The most positive correlations are present for coefficient 5. Of these, the strongest one is to be found at its own lags, but smaller positive values are present at the lags of coefficients 0, 3 and 4. The lags of coefficient 2 are correlated negatively to coefficient 5 and only coefficient 1 does not have any lags correlating to coefficient 5 with a magnitude above 0.2.

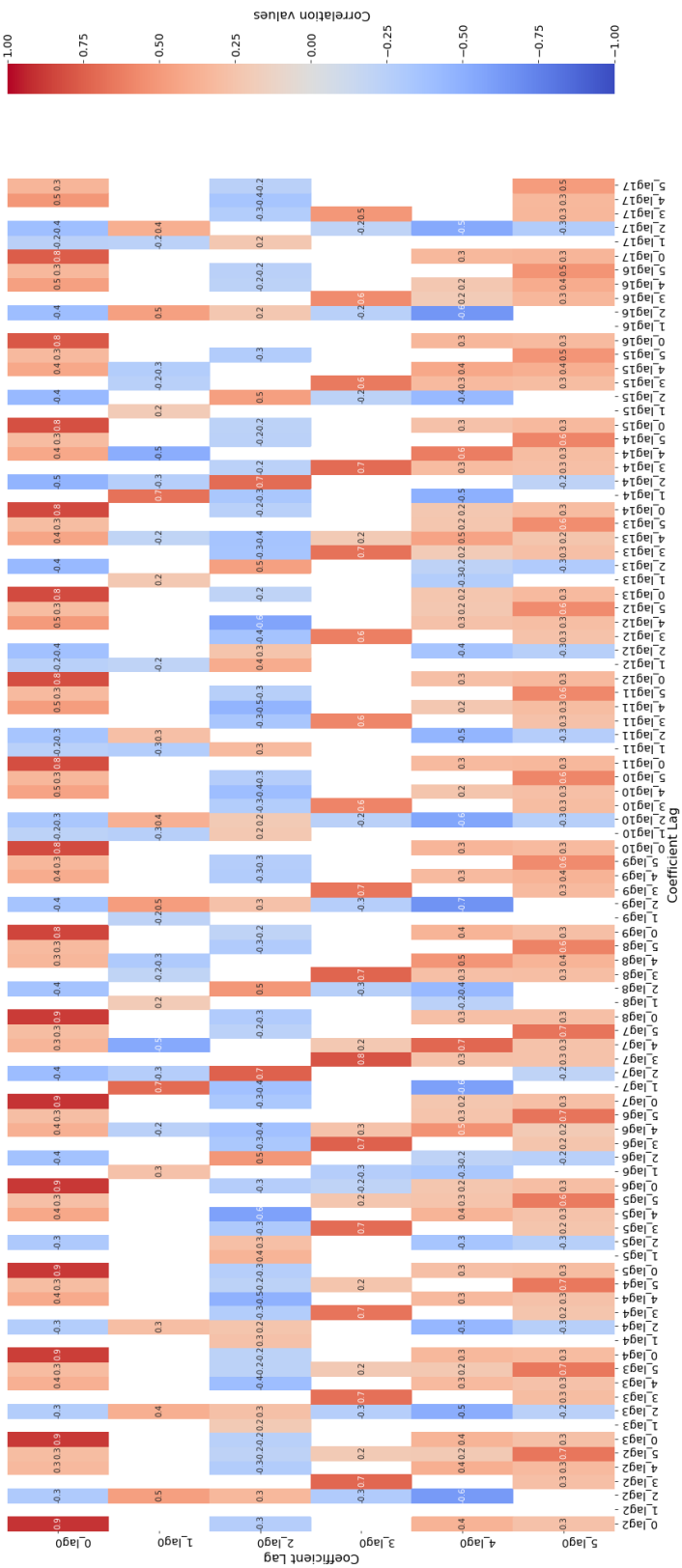



Figure A.17.: Cross-correlations among columns of $\mathbf{W}$ (for $r = 6$) with those columns being shifted with lags 2 to 17. Only magnitudes above 0.2 are shown.

Most correlations of lags 3 to 17, generally, look similar to the ones in lag 2. There are however a few exceptions where the difference at a certain lag towards the prior lag is recognizably large. In lag 6 we see changes in coefficients 1, 2 and 4, compared to lag 5. Lag 7 shows another change in the same direction as we saw in lag 6 in coefficients 1, 2 and 4. At lag 8 the changes of lag 6 seem to be turned back and we also see some more change in the correlations of coefficient 4. Lag 9 then seems to turn back the changes of lag 7. The changes from lag 13 to lag 12 are nearly the same as the changes from lag 6 to lag 5. The same applies to the changes from lag 14 to 13 compared to the differences among lags 7 and 6. The changes from lag 14 to 15 and from 15 to 16 then turn back these changes from 12 to 13 and from 13 to 14 respectively and can be compared to the changes between lags 7 and 8 and between 8 and 9.

Additionally, we can look at the differences, not to the previous lag's correlations, but at the difference of the lags's correlations to the ones present at lag 2: Lag 4 shows a slight increase in the correlations of coefficient 3. Lag 5 shows a slight increase in the correlation magnitudes of coefficients 1, 2, 3 and 4, as does lag 6 in 1, 2 and 3. Lag 7 shows a large increase in the correlation magnitudes of coefficients 1 and 2 and another one for the coefficient of 4. Lag 8 is generally similar to lag 6 for the magnitude differences in the correlations of coefficients 1 and 2. Lag 9 and 10 nearly show no significant difference to lag 2, lag 11 has a slight difference in the correlation magnitudes of coefficients 1, 3 and 4 and lag 12 has a difference for coefficients 1-4. Lags 13 and 14 are similar to each other for coefficients 1 and 2 with the higher differences in lag 14 and additional differences in coefficient 4, whereas lag 13 also shows a slight difference for coefficient 3. In lag 15 we then again see a decrease in the differences to lag 2, with only the ones for coefficients 1 and 2 being mentionable.

Concluding the analysis about which lags to use as features for the inner regression model of the NMF models, one can say: It makes sense to use the lags 2, 6, 7, 8 and 14 as they have relatively high PACF values and show a significant correlation towards the non-lagged series. Additionally, the cross-correlations of lags 6, 7, 8 and 14 differ from the ones in lag 2. These lags can therefore be expected to carry information about the target day that is to be predicted.

### A.2.3. Iterating shift-based models

Looking at the performance of the shift-based models, at first, the errors of different regression models, i.e. a dense NN, a Random Forest and a Ridge Regression will be analyzed. For this comparison, all three regression models operate on the same basic features: The shifts of 2, 6, 7, 8 and 14 days, the temperature of the target timestamp and the temperature shifts of 2, 6, 7, 8 and 14 days.

The first thing we see from figure A.18 is that using a Dense Neural Network as the regression model is very unstable in terms of the forecast errors, as it shows very high error peaks at irregular intervals. However, in those cases where no peak is present

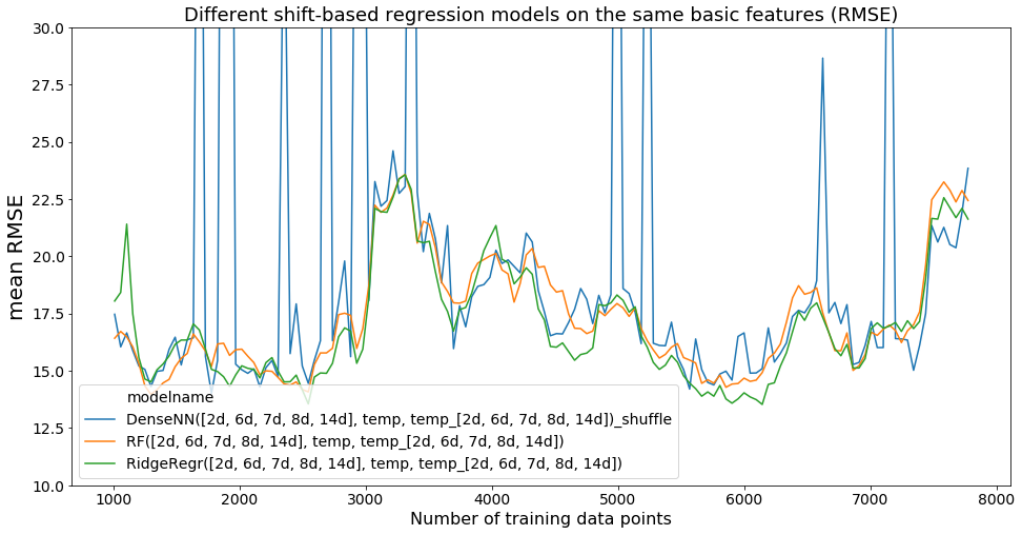



Figure A.18.: RMSE of shift-based models with different regression models, averaged over all four starting date setups (full plot in appendix figure A.66). 48 data points correspond to one day.

the Dense NN model can keep up with the Random Forest (RF) and Ridge Regression models. We also see that the RF model gives slightly higher errors than the model using a Ridge Regression.

To get an idea of how additional features affect the forecast accuracy of the shift-based models, the regression model will now be kept constant and the effects of adding different features is looked at in figures A.19 and A.20 for Ridge Regression and RF respectively. Multiple things can be recognized in these two figures: First of all, no matter the features, all models show the main peaks after about 3000 training data points and at the end of the plotted time after about 7500 training data points. We also see that adding more features affects on the models' performance. Adding the statistical features leads to a worse performance when using a Ridge Regression. However, with the RF Regression, adding these features slightly increases performance in some low error areas, but overall no large improvement is shown.

Looking at the influence of the index features, the difference in errors is clearer. Especially for the RF models, the RMSE was lowered for nearly all training data lengths. Also with a RF, adding the index features on top of the statistical features increases the performance, compared to only using the statistical features. For the Ridge Regression models, the opposite is the case and adding the index features decreased the accuracy consistently.

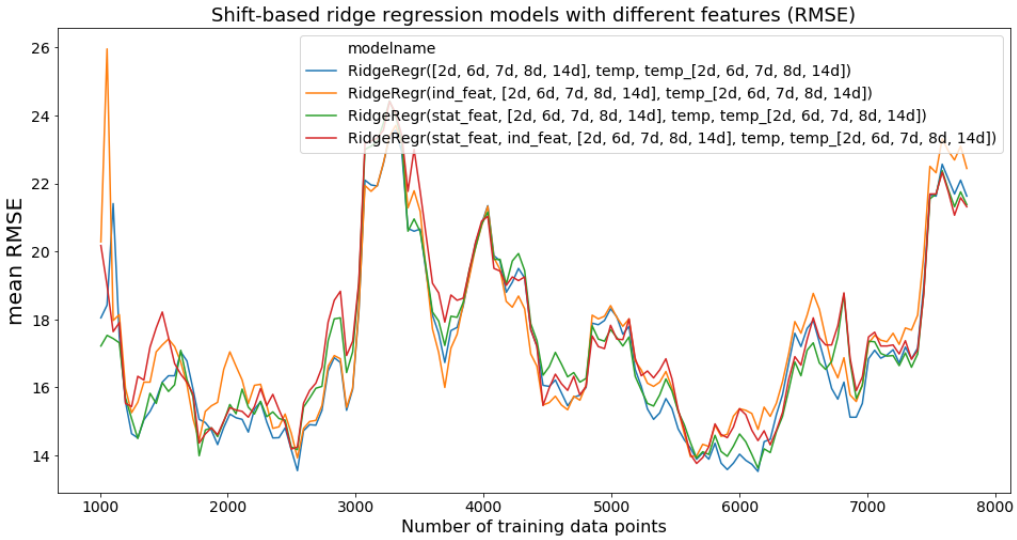



Figure A.19.: RMSE of shift-based Ridge Regression models with different features, averaged over all four starting date setups. 48 data points correspond to one day.

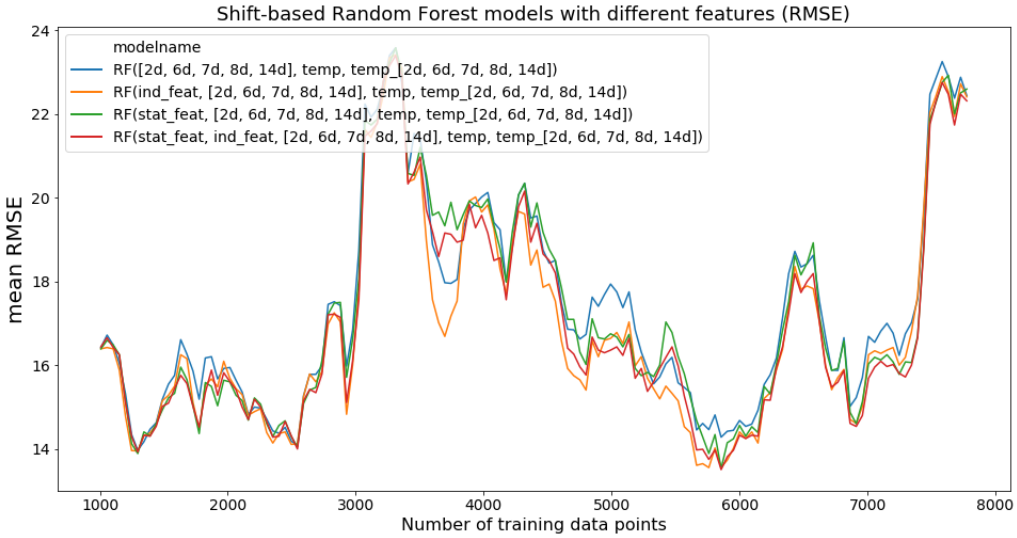



Figure A.20.: RMSE of shift-based Random Forest models with different features, averaged over all four starting date setups. 48 data points correspond to one day.

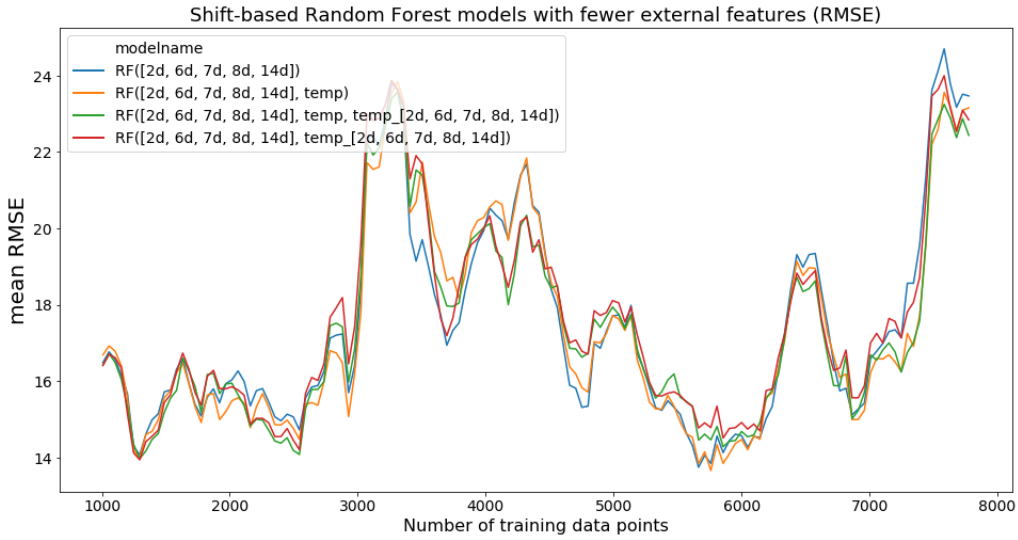



Figure A.21.: RMSE of shift-based RF models with fewer external features, averaged over all four starting date setups. 48 data points correspond to one day.

After having added more features, next the effect on the prediction errors is analyzed when fewer features are used. For this analysis, all models again rely on the same shifts, but different combinations of using the index features, target temperature, and temperature shifts are compared.

First, looking at the shift-based models using a RF Regression, figure A.21 shows that for the RF models, apart from the Christmas and New Years peak, using no temperature based features is a disadvantage. Among the models using temperature features, the ones with the lower errors are the ones leveraging the temperature of the target day. Using the temperature shifts alone seems to help less. However, the model using both the temperature of the target day and the temperature shifts (what we used as the basic feature setup) gives similar results to the model only using the target temperature after having seen about 6000 data points. From figure A.22 we see that, when additionally using the index features, the temperature lags are no longer needed. Compared to the basic feature combinations, all RF models using the index features achieve lower errors. Especially the model only using the shifts and the index features and the one using shifts, target temperature, and index features provide low forecast errors.

Now looking at the shift-based models using a Ridge Regression instead of a RF in figure A.23, we can see that, for these models, not leveraging the temperature features results in higher errors. Additionally, including the target temperature in the features led to a bigger improvement than using the temperature shifts. For the Ridge Regression models, additionally using the index features does not lead to lower errors, compared to the models using the basic feature setup (see figure A.24).

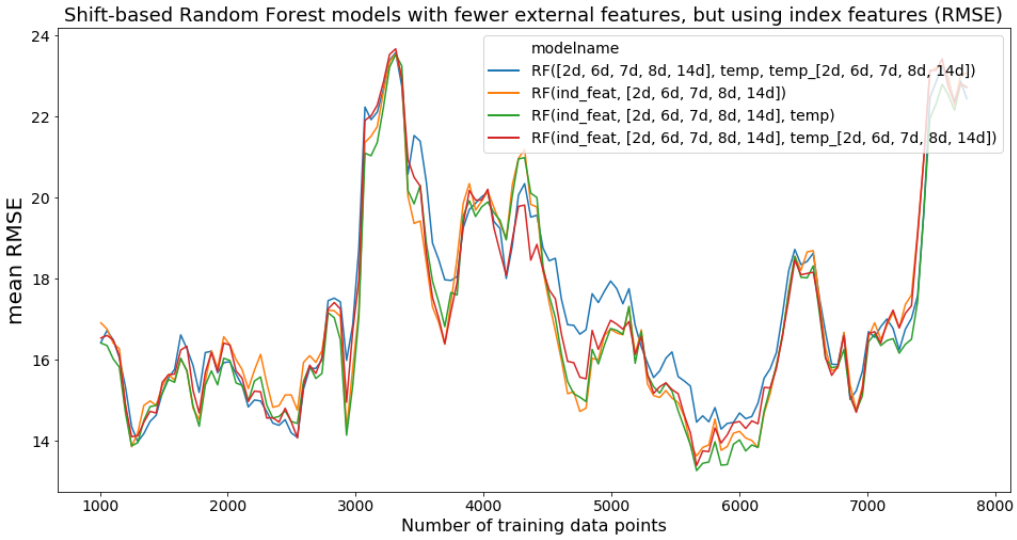



Figure A.22.: RMSE of shift-based RF models with fewer external features, but using index features, averaged over all four starting date setups. 48 data points correspond to one day.

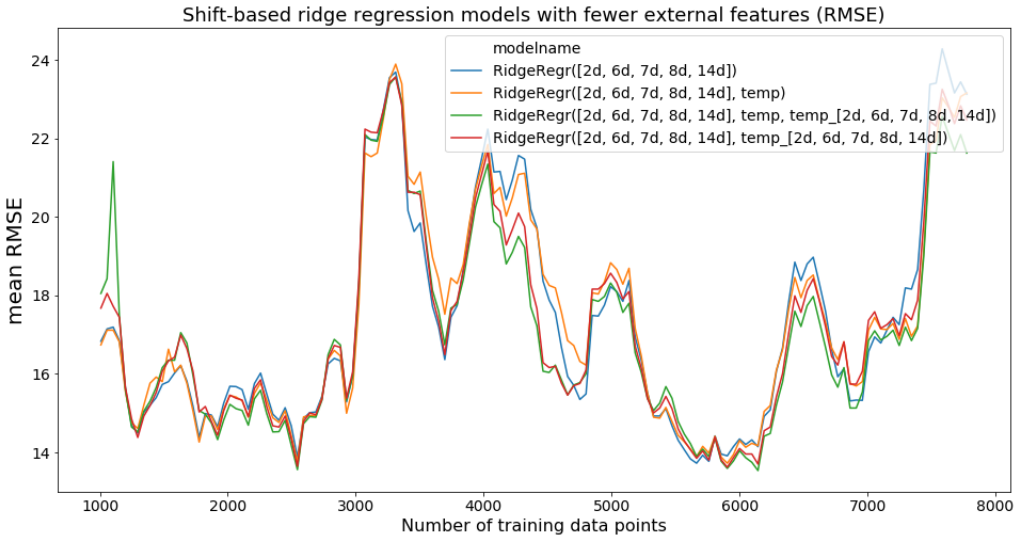



Figure A.23.: RMSE of shift-based Ridge Regression models with fewer external features, averaged over all four starting date setups. 48 data points correspond to one day.

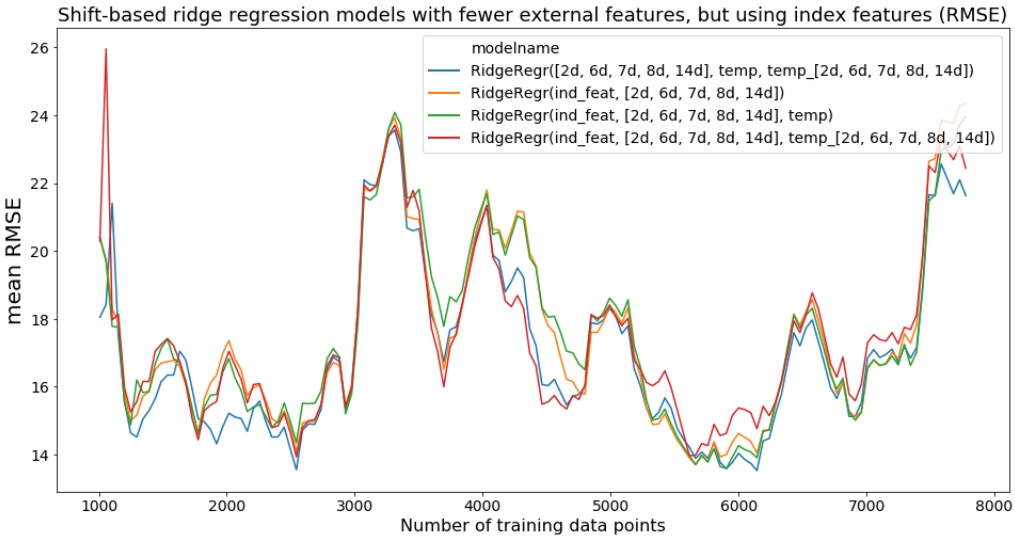



Figure A.24.: RMSE of shift-based Ridge Regression models with fewer external features, but using index features, averaged over all four starting date setups. 48 data points correspond to one day.

To go into a little more detail about the effect of using the index features with the RF models, the errors of models using the target temperature and index features, with additional options, i.e. the temperature shifts and statistical features are also compared.

Looking at figures A.25 and A.26 we see that all presented models give similar errors. The worst performance is achieved with the model not using the target temperature, having slightly more error peaks. It can be concluded that the RF model only using the shifts, index features, and target temperature is the most efficient and thus best one.


Figure A.25.: RMSE of shift-based RF models using index features, averaged over all four starting date setups. 48 data points correspond to one day.

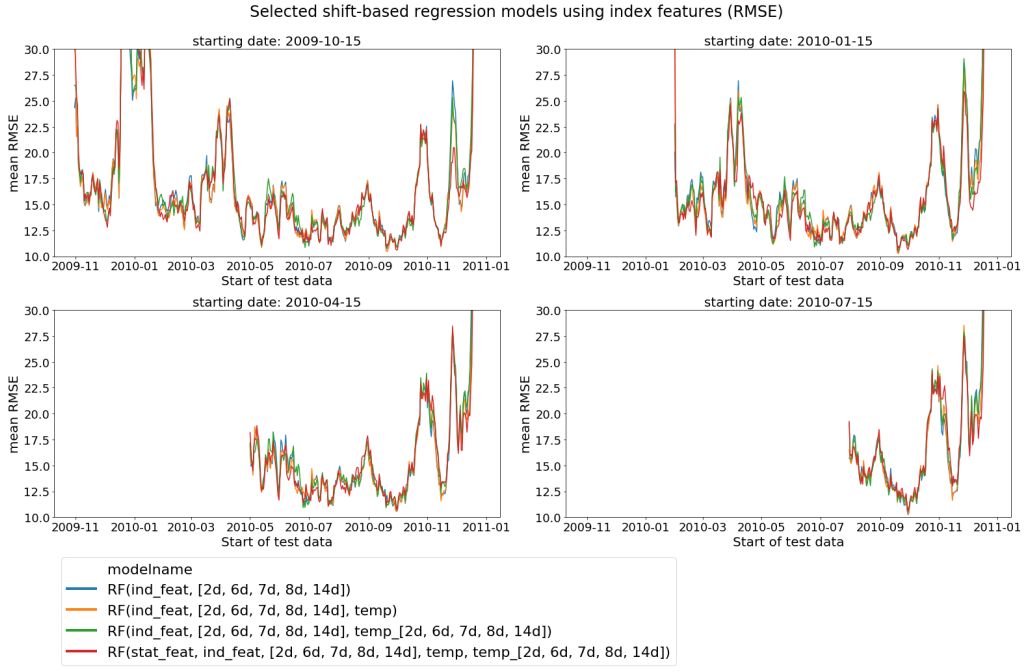



Figure A.26.: RMSE of shift-based RF models using index features for all four starting date setups (full plots in appendix figure A.68).

### A.2.4. Iterating NMF-based models

We already saw for the reconstruction mock models, that putting an $L1$ regularization with a value of $\alpha = 400$ on the NMF part of the model does not help to decrease the errors. This is also true for the NMF-based regression models, as is shown in figures A.27, A.28, A.29 and A.30 for $r = 4$ and $r = 6$, both with RF and Ridge Regression.

Next, it is analyzed if the use of additional features can increase the performance of the NMF-based models.

From figures A.31, A.32, A.33 and A.34 we can see multiple things: First, we see that, when adding the statistical daytime features of the four time ranges (morning, midday, afternoon and evening), the average RMSE values are similar or slightly higher for all training lengths. Including index features about the target day increases the performance of the NMF-based models, for both RF and Ridge Regression. It should be noted, though, that the Ridge Regression models need more training data than the RF Regression to profit from the index features. With only little data available, the performance of models not relying on them is better. But after approximately 5000 training data points, the index features help to decrease the average RMSE.

Comparing the two regression types for the NMF-based models, the effect of using the index features is higher for the models using a RF Regression. Here the use of index features consistently decreases the RMSE for all training lengths.

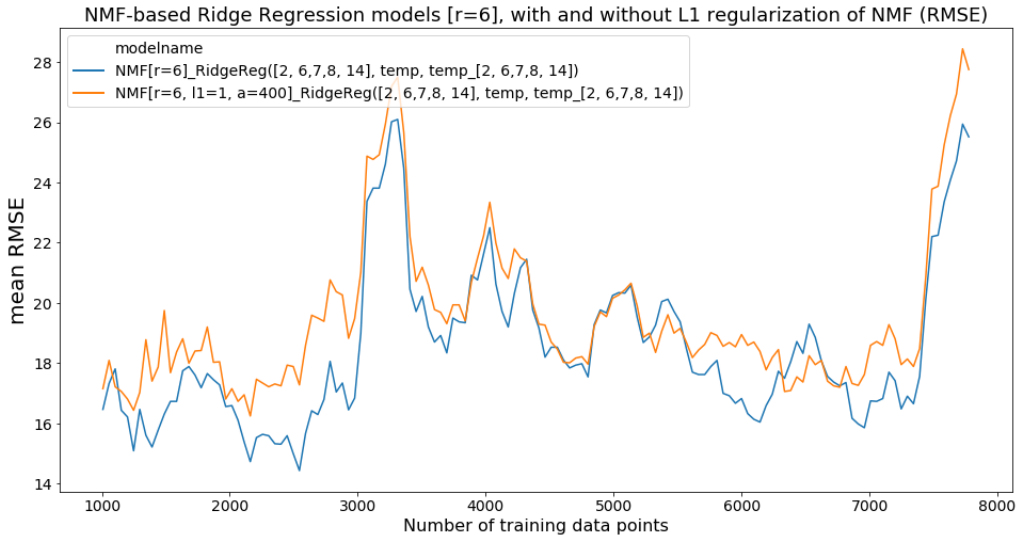



Figure A.27.: RMSE of NMF-based Ridge Regression models on basic features for $r = 6$, with and without $L1$ regularization on the NMF, averaged over all four starting date setups. 48 data points correspond to one day.

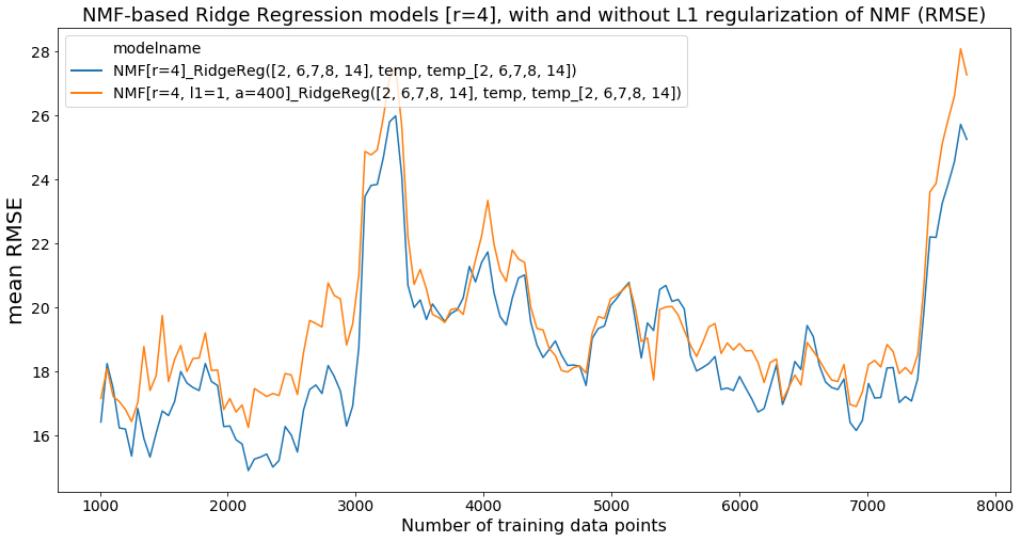



Figure A.28.: RMSE of NMF-based Ridge Regression models on basic features for $r = 4$, with and without $L1$ regularization on the NMF, averaged over all four starting date setups. 48 data points correspond to one day.

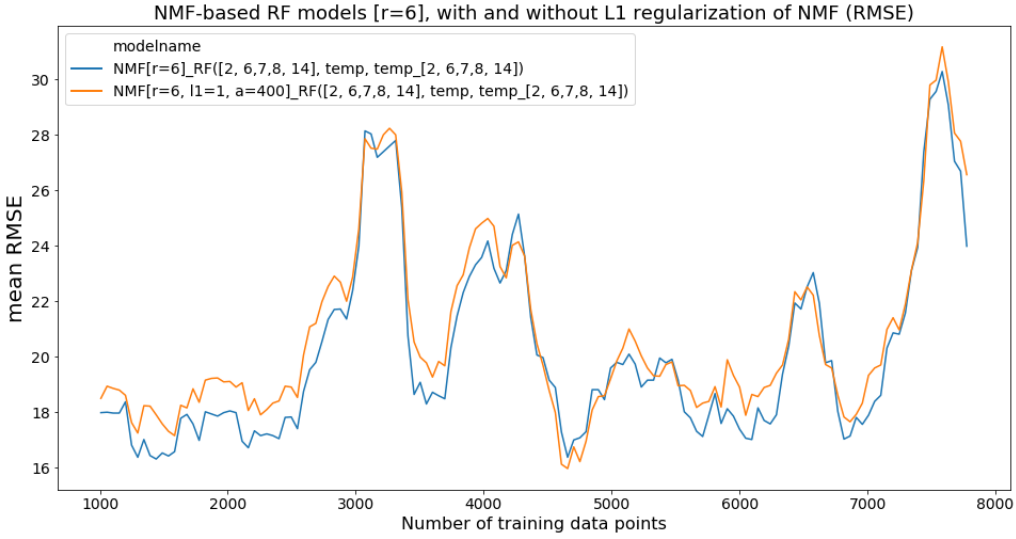



Figure A.29.: RMSE of NMF-based RF models on basic features for $r = 6$, with and without $L1$ regularization on the NMF, averaged over all four starting date setups. 48 data points correspond to one day.

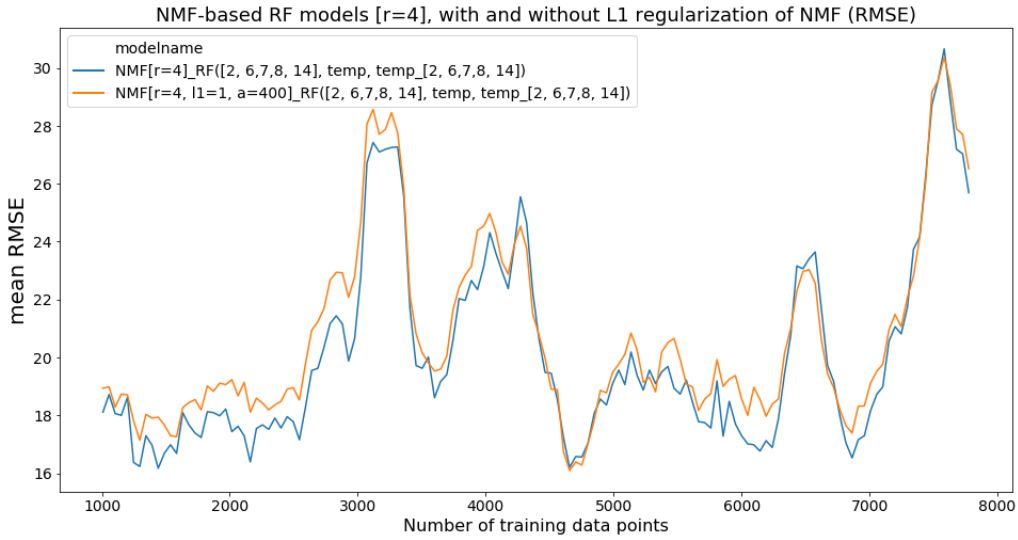



Figure A.30.: RMSE of NMF-based RF models on basic features for $r = 4$, with and without $L1$ regularization on the NMF, averaged over all four starting date setups. 48 data points correspond to one day.

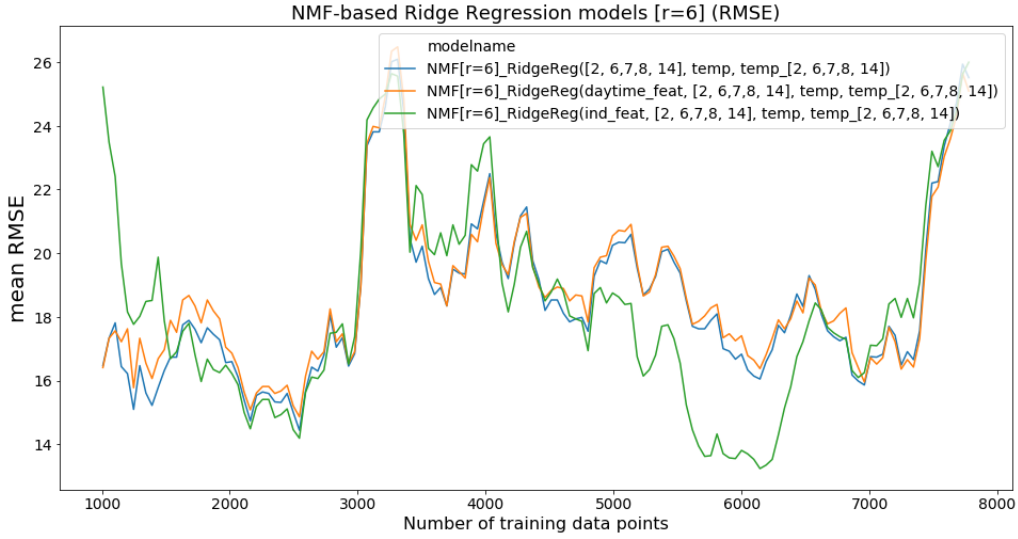



Figure A.31.: RMSE of NMF-based Ridge Regression models for $r = 6$, with different features, averaged over all four starting date setups. 48 data points correspond to one day.

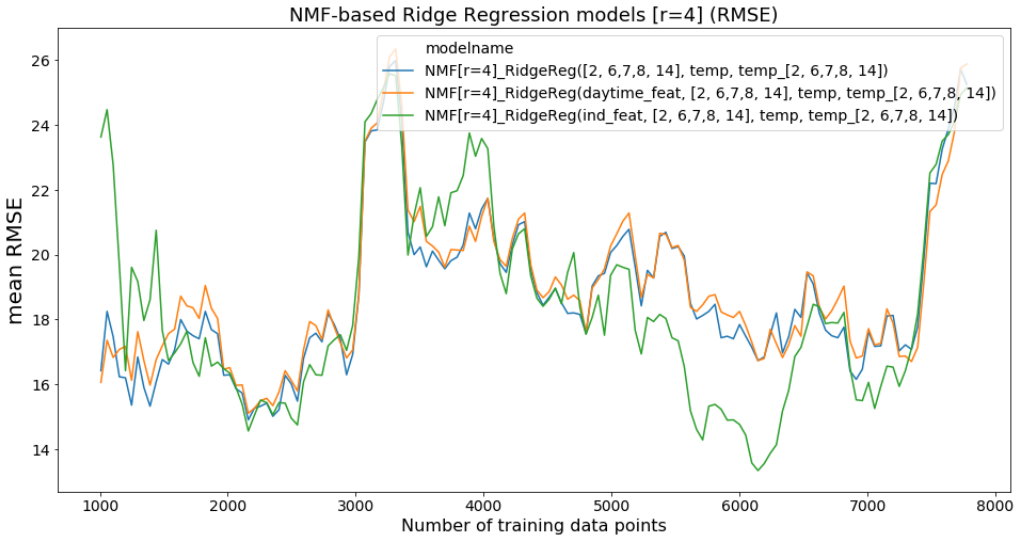



Figure A.32.: RMSE of NMF-based Ridge Regression models for $r = 4$, with different features, averaged over all four starting date setups. 48 data points correspond to one day.

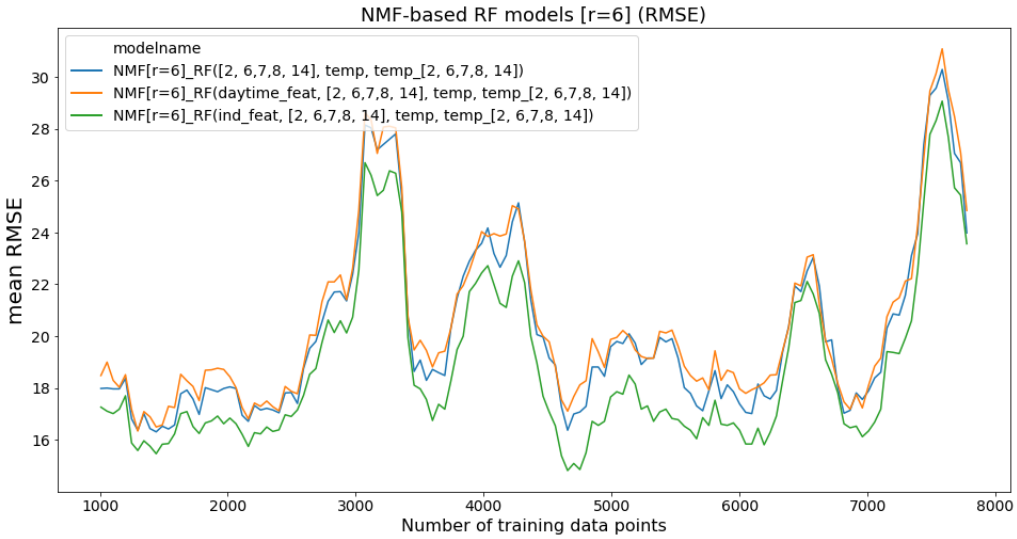



Figure A.33.: RMSE of NMF-based RF models for $r = 6$, with different features, averaged over all four starting date setups. 48 data points correspond to one day.

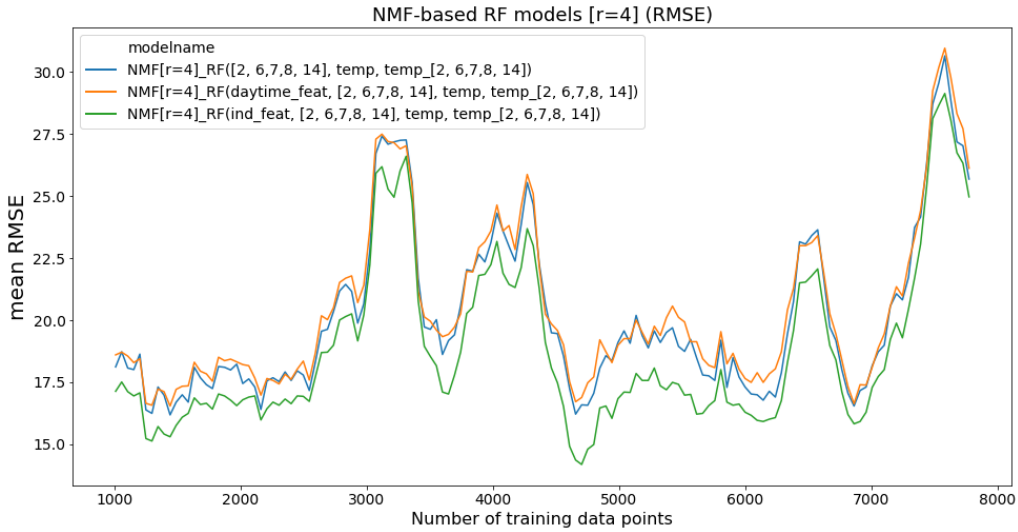



Figure A.34.: RMSE of NMF-based RF models for $r = 4$, with different features, averaged over all four starting date setups. 48 data points correspond to one day.

Next, the effect of adding the index features is analyzed in more detail. Figures A.35 and A.36 show the RMSE values of the NMF-based models using a Ridge Regression, operating on the basic features, with and without the added index features, for each of the four starting dates.

We can see some larger errors for little amounts of training in two starting dates 2009-10-15 and 2010-07-15. For most areas of the plots, the two forecasting errors are similar with a slight advantage towards the models using the index features. However, with increasing amounts of training data, the models using the index features show significantly lower errors, as we can see for the first two starting dates.

The differences between the NMF-based models using the same feature setups with a RF Regression are smaller, as is shown in figures A.37 and A.38. Adding the index features, here, only slightly increases the performance, no matter the training data lengths.

Overall, adding the index features does help to increase the performance of the NMF-based models.

Similar to the analysis of the shift-based models, it is looked into the situation of the NMF-based models using fewer external features.

Focusing on the Ridge Regression NMF models, figures A.39 and A.40 show that when using the temperature lags, the error is higher compared to the models not using them. In contrast, using the target temperature does help to decrease the averaged RMSE. The most successful feature combination in these plots is the one including the lags and target temperature, but not the temperature lags. While dropping the temperature lags also increases the performance when using a Random Forest Regression with the NMF-based models, the error curves of the model only using the lags and the model additionally using the target temperature are very similar (see figures A.41 and A.42).

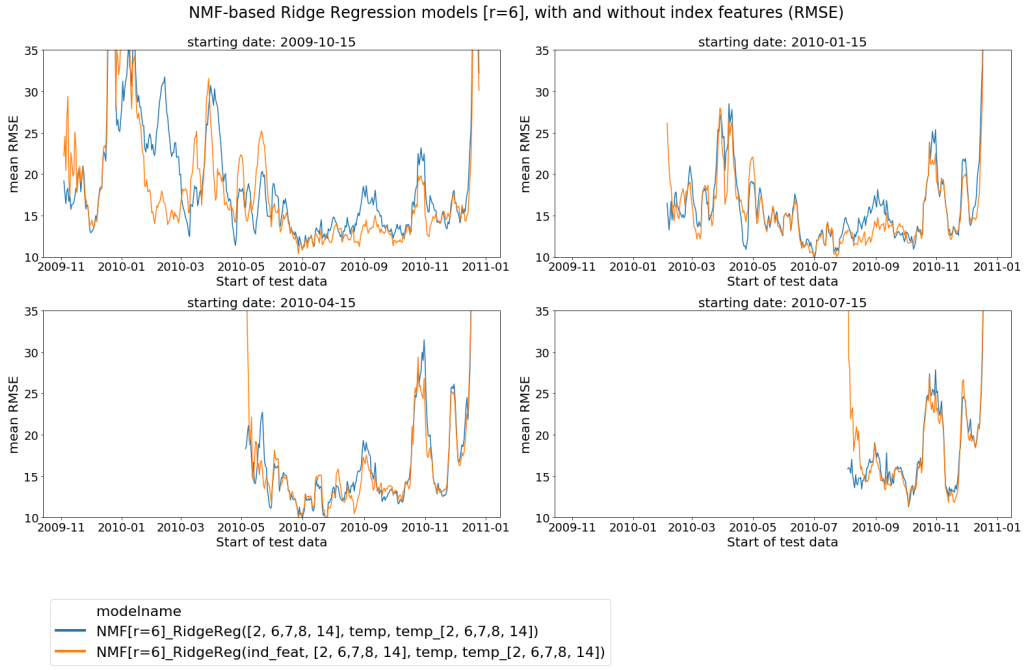



Figure A.35.: RMSE of NMF-based Ridge Regression models for $r = 6$, with and without index features, for all four starting date setups (full plots in appendix figure A.69).

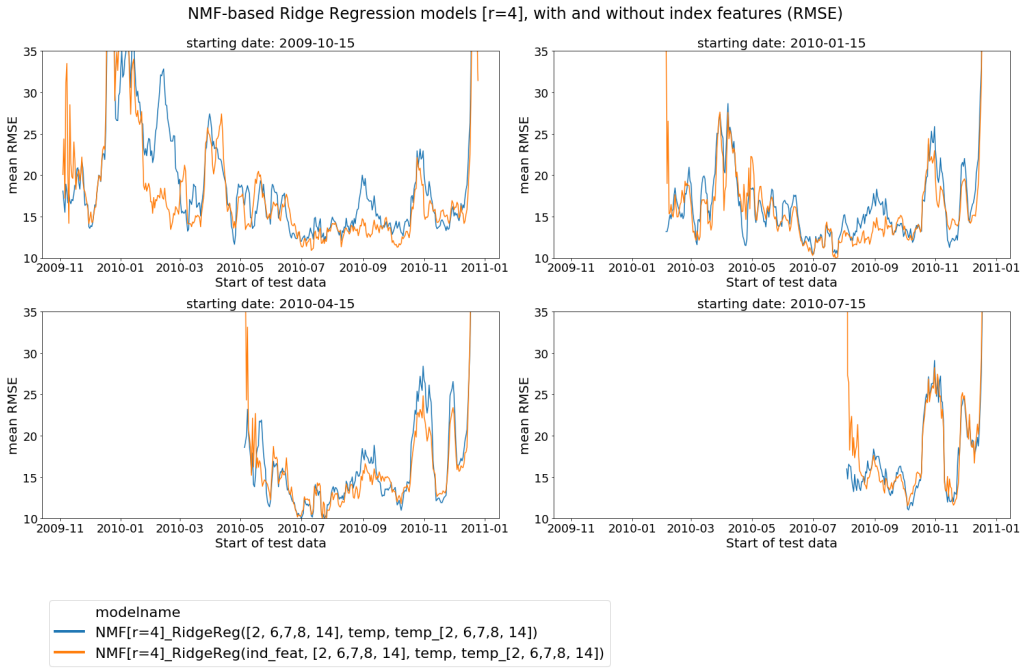



Figure A.36.: RMSE of NMF-based Ridge Regression models for $r = 4$, with and without index features, for all four starting date setups (full plots in appendix figure A.70).

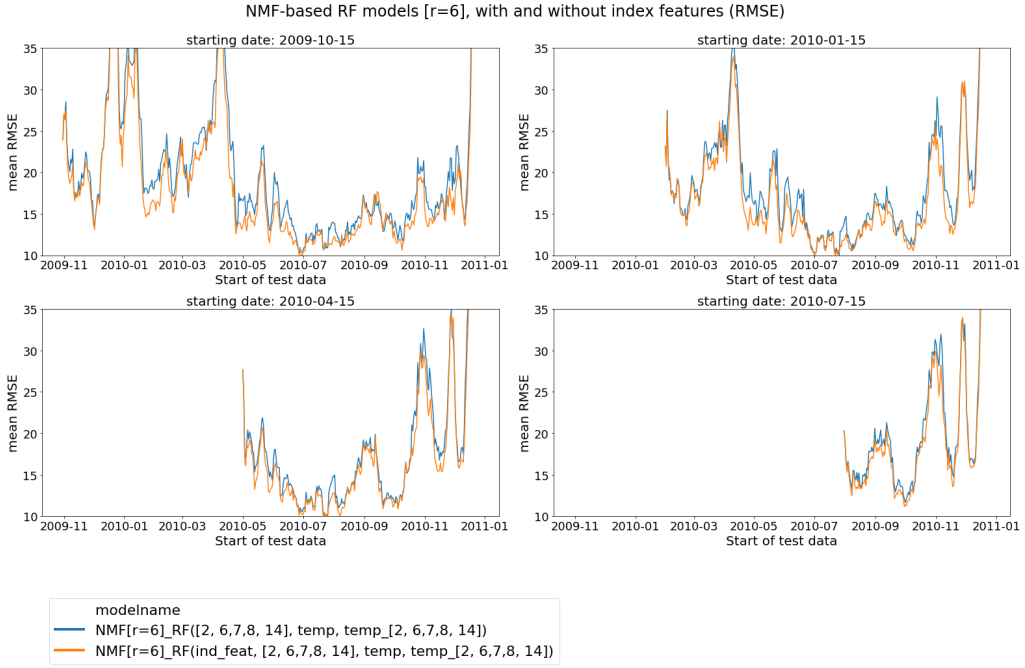



Figure A.37.: RMSE of NMF-based RF models for $r = 6$, with and without index features, for all four starting date setups (full plots in appendix figure A.71).

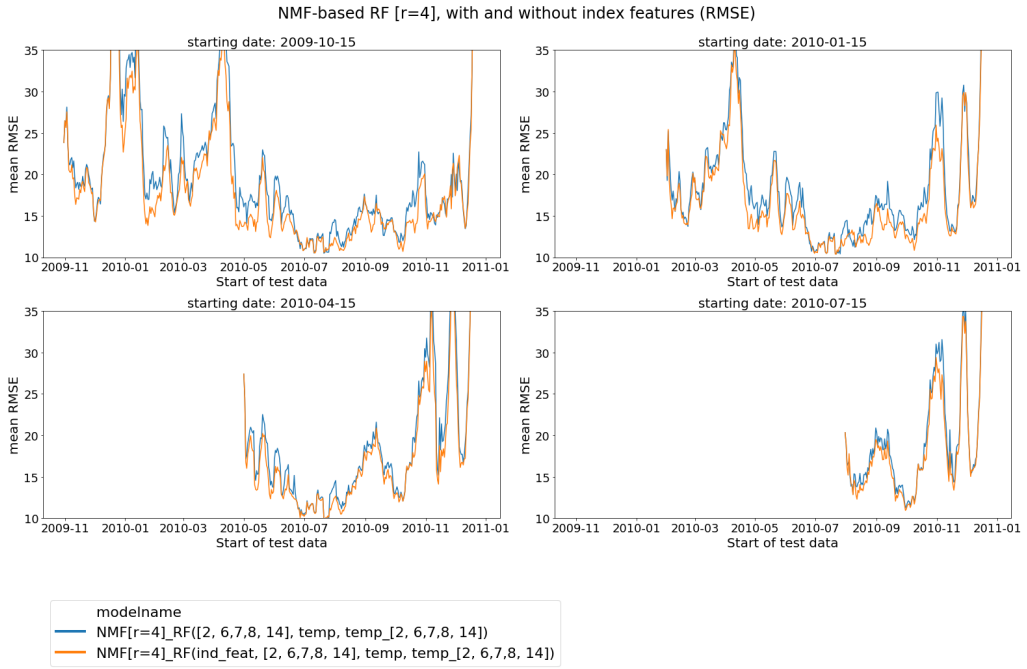



Figure A.38.: RMSE of NMF-based RF models for $r = 4$, with and without index features, for all four starting date setups (full plots in appendix figure A.72).

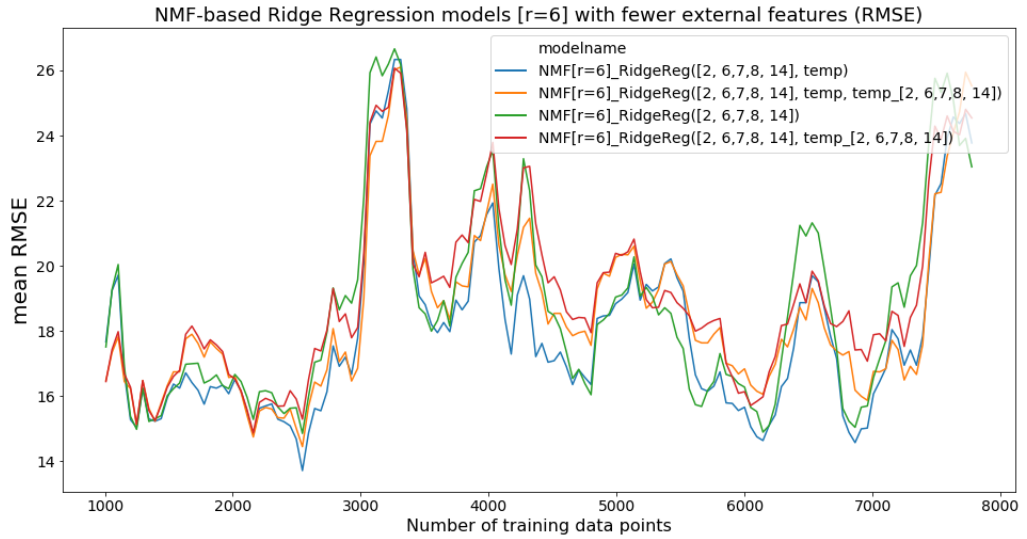



Figure A.39.: RMSE of NMF-based Ridge Regression models for $r = 6$, with fewer external features, averaged over all four starting date setups. 48 data points correspond to one day.

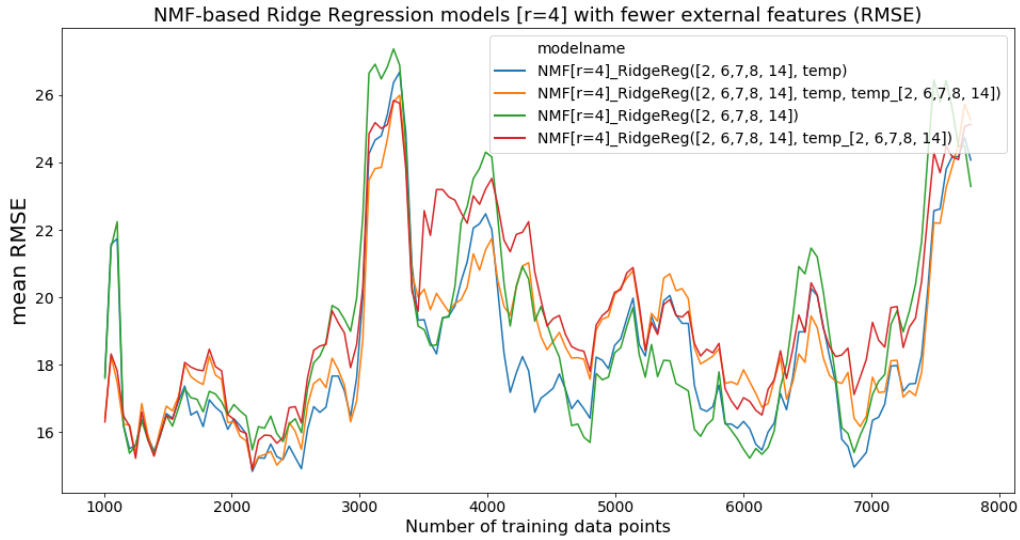



Figure A.40.: RMSE of NMF-based Ridge Regression models for $r = 4$, with fewer external features, averaged over all four starting date setups. 48 data points correspond to one day.

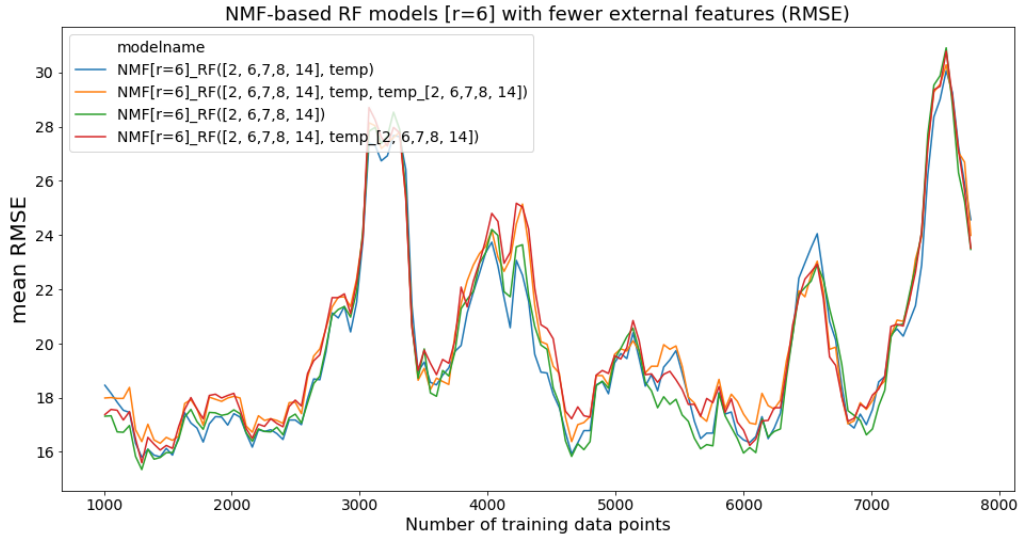



Figure A.41.: RMSE of NMF-based RF models for $r = 6$, with fewer external features, averaged over all four starting date setups. 48 data points correspond to one day.

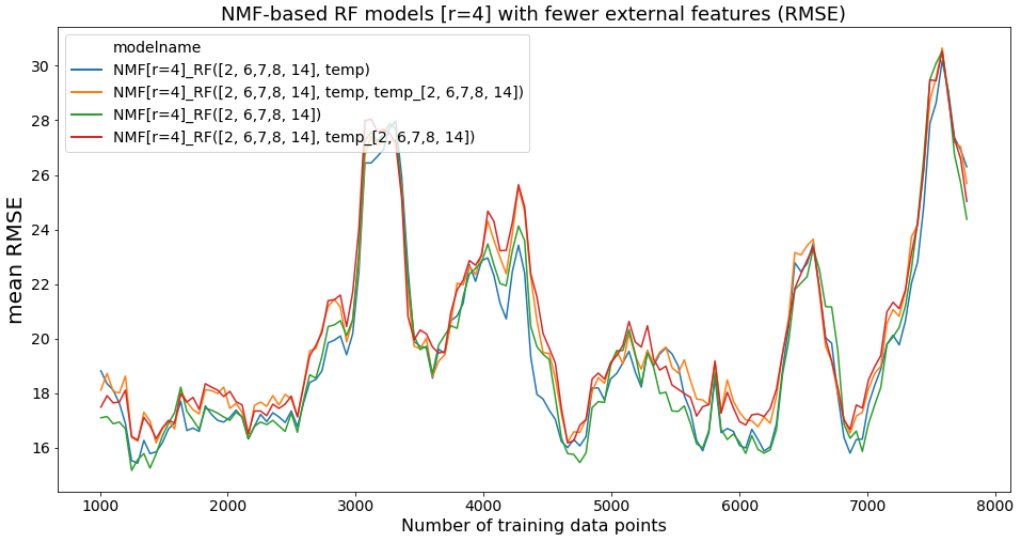



Figure A.42.: RMSE of NMF-based RF models for $r = 4$, with fewer external features, averaged over all four starting date setups. 48 data points correspond to one day.

Looking at the individual starting dates in figures A.43 and A.44, we see that the plots of $r = 4$ and $r = 6$ are nearly the same for most of the time. Additionally, while in some areas the models leveraging the temperature features perform better, there are peaks where not relying on the target temperature helps to decrease the errors.

Going towards finding the ideal combination of available features, the temperature lags are dropped and the index features are included. Comparing feature combinations of lags, target temperature and index features in figures A.45 and A.46, we see that, for the models using RF, including the index features brings down the RMSE values for nearly all times. The model using both, index features and target temperature, on top of lags, in some regions achieves slightly lower values than the one only using index features and lags. When using a Ridge Regression (figures A.47 and A.48) with NMF-based models, the model with the lowest error is the one using the target temperature and index features on top of the lags. Especially for $r = 4$, combining target day temperature features and index features helps to reduce the Root Mean Squared Error (RMSE) values after a sufficient amount of training data has been shown to the model.

Thus, overall, we can say that both the target temperature and index features are useful, though, combined they result in larger improvements than each one of them alone.

Lastly, we want to get an impression of how the actual NMF-based models compare to their reconstruction baselines represented by the mock model errors. For this comparison the models using index features, target temperature, and lags as features are chosen. In figure A.49 we see three things. Firstly, the errors of the models are far from their lower boundaries, given by the reconstruction mock models. Secondly, the peaks in the performance of NMF-based models appear where the reconstruction models also show higher errors. Thirdly, we see that increasing the internal NMF dimension from $r = 4$ to $r = 6$ does not necessarily help to decrease the error in all situations.

Looking closer at the errors for each start date in figure A.50, we see that the peaks that appear in the reconstruction baselines mainly occur around Christmas and at Daylight Saving Time (DST) changes on 2009-10-25, 2010-03-28 and 2010-10-31. We also see that, compared to the Ridge Regression models, the RF models are a little more prone to peaks in the reconstructions.

Another interesting observation from this analysis is that, when comparing the plots of the reconstruction mock models for the later three starting dates, the error curves of the $r = 4$ and $r = 6$ NMFs stay close to each other in the beginning and only split up after more data is used.

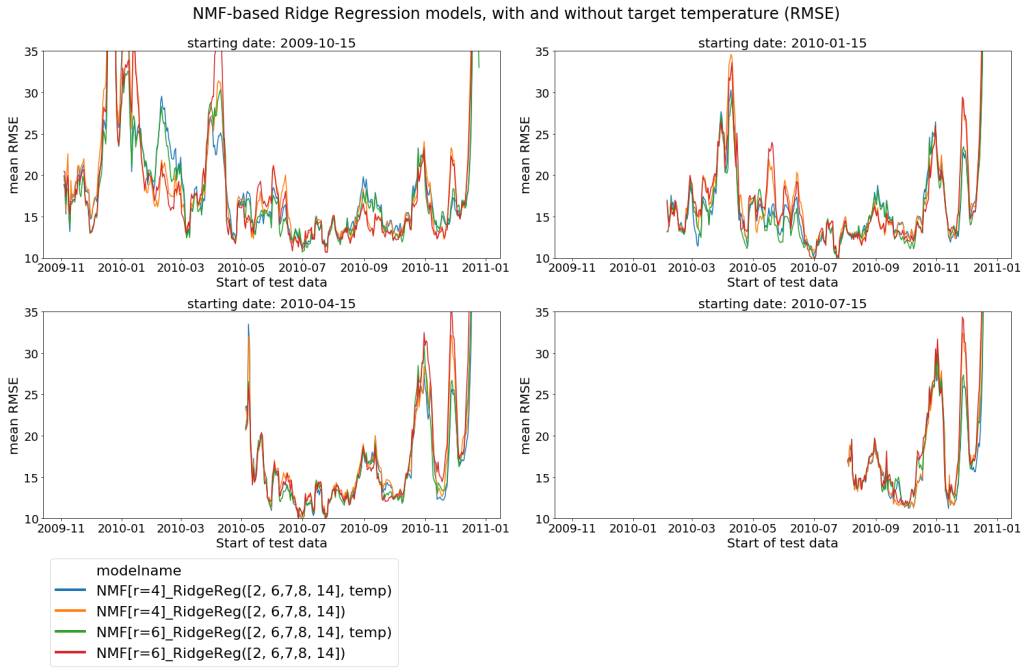



Figure A.43.: RMSE of NMF-based Ridge Regression models, with and without target temperature, for all four starting date setups (full plots in appendix figure A.73).

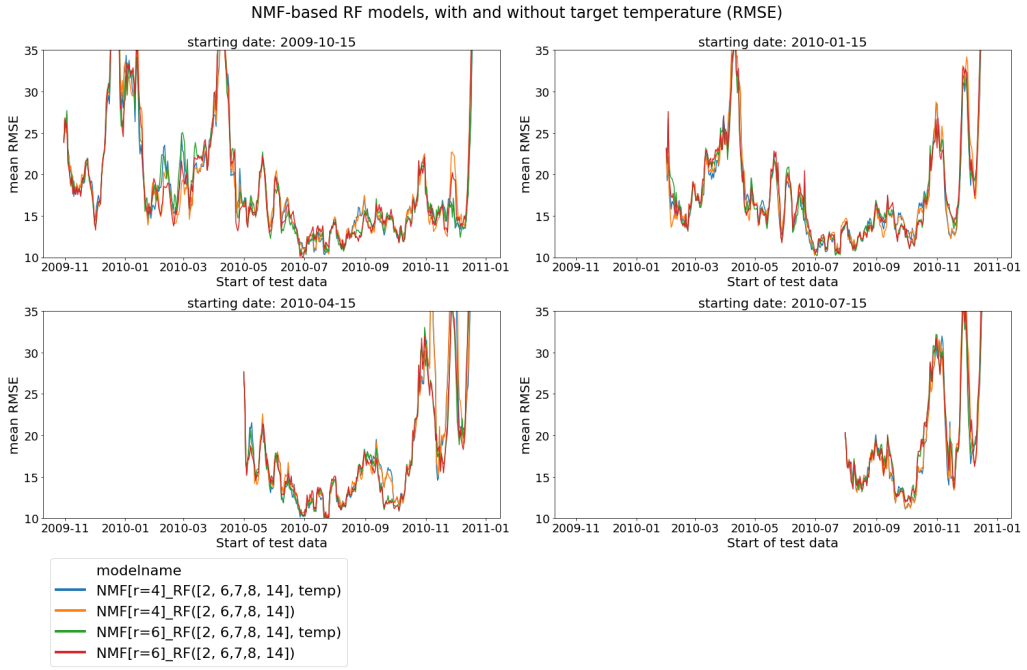



Figure A.44.: RMSE of NMF-based RF models, with and without target temperature, for all four starting date setups (full plots in appendix figure A.74).

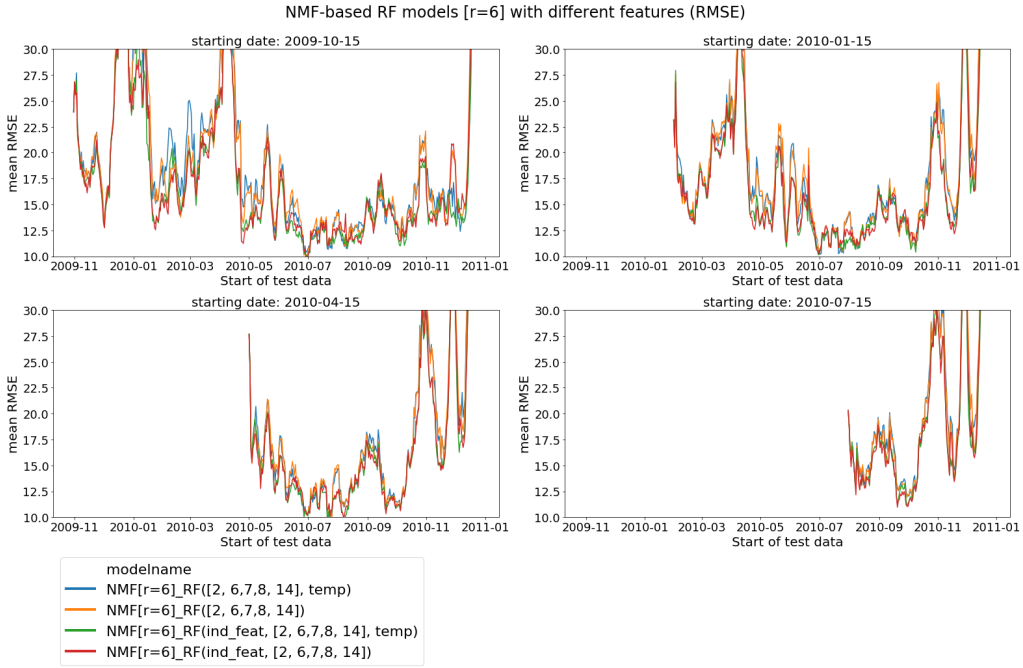



Figure A.45.: RMSE of NMF-based RF models for $r = 6$, with different features, for all four starting date setups (full plots in appendix figure A.77).

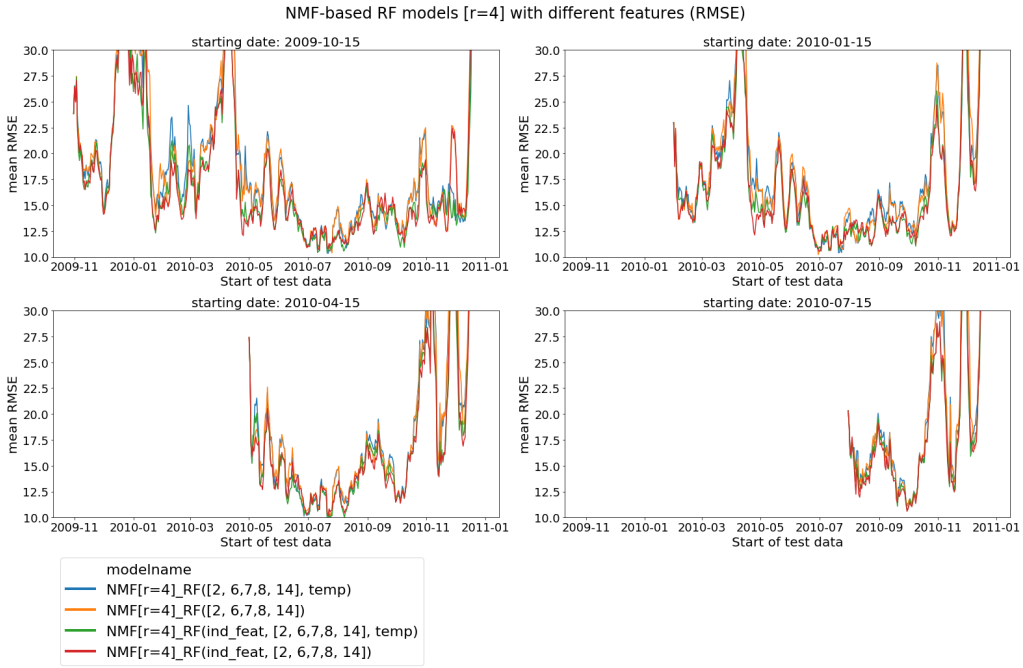



Figure A.46.: RMSE of NMF-based RF models for $r = 4$, with different features, for all four starting date setups (full plots in appendix figure A.78).

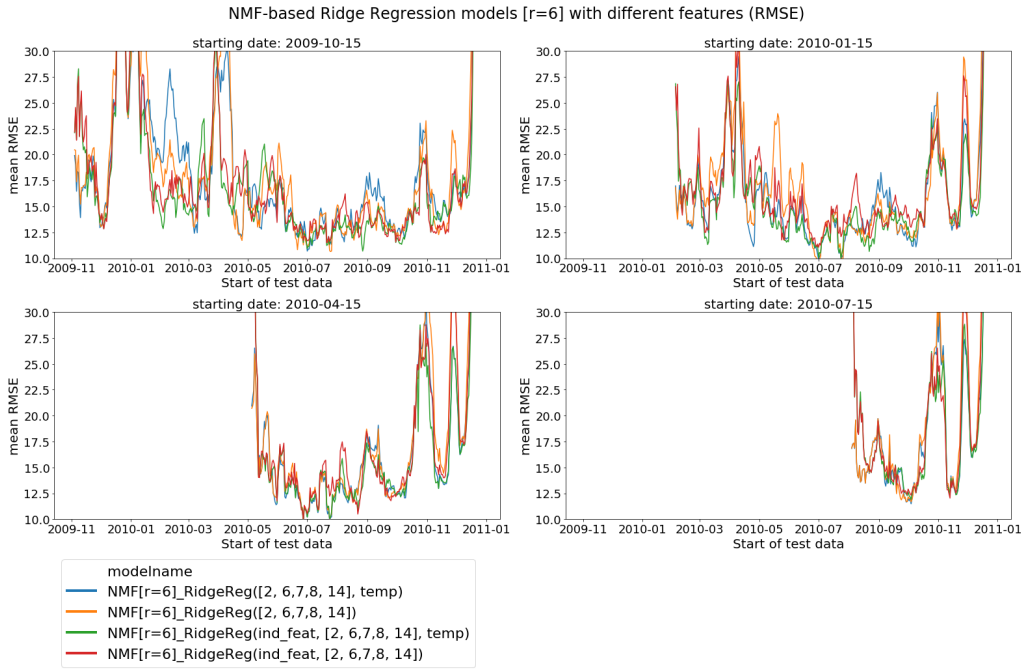



Figure A.47.: RMSE of NMF-based Ridge Regression models for $r = 6$, with different features, for all four starting date setups (full plots in appendix figure A.75).

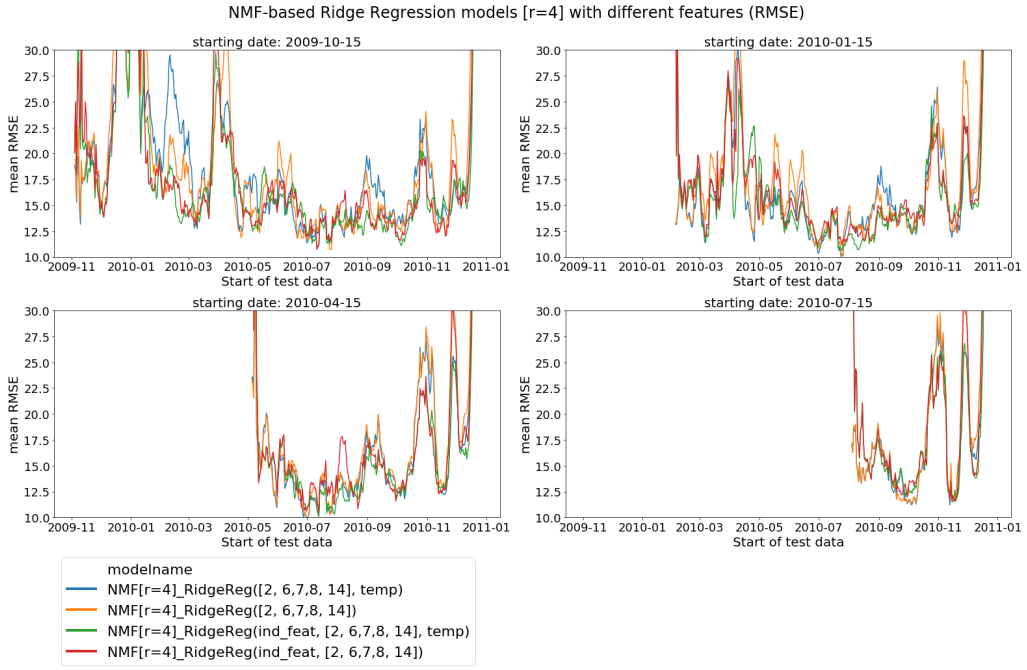



Figure A.48.: RMSE of NMF-based Ridge Regression models for $r = 4$, with different features, for all four starting date setups (full plots in appendix figure A.76).

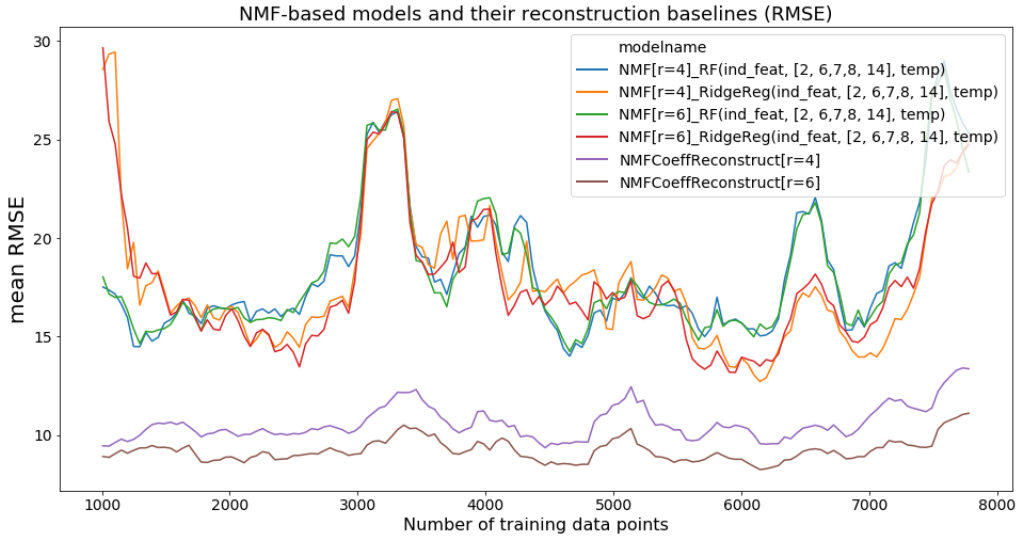



Figure A.49.: RMSE of selected NMF-based models and their reconstruction baselines, averaged over all four starting date setups. 48 data points correspond to one day.

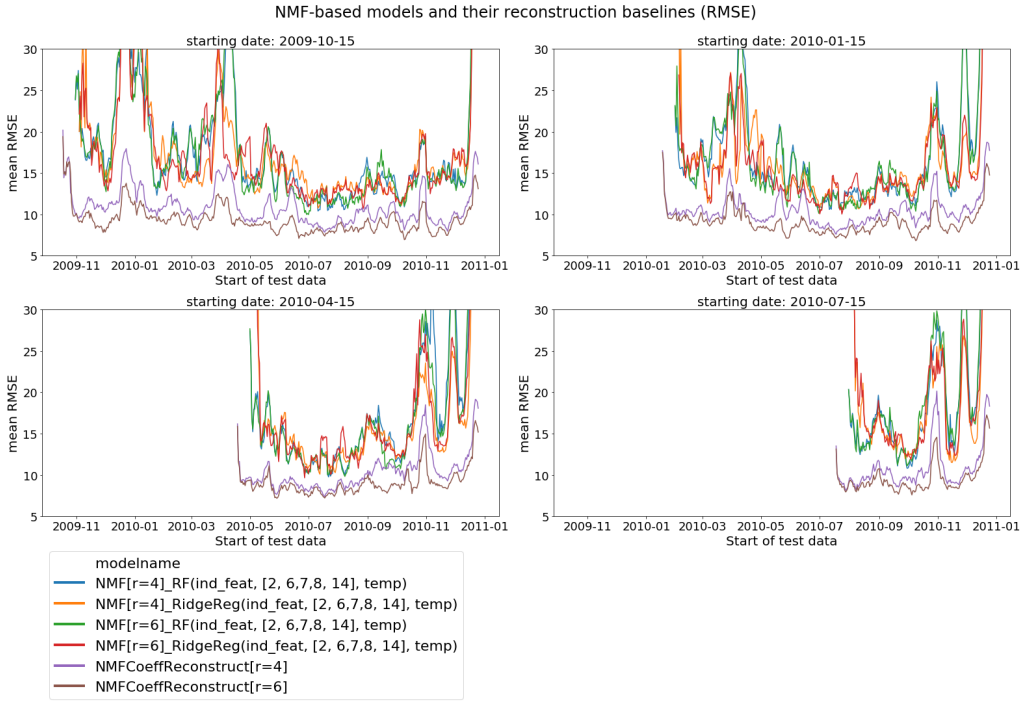



Figure A.50.: RMSE of selected NMF-based models and their reconstruction baselines for all four starting date setups (full plots in appendix figure A.79).

### A.2.5. Comparing model iterations from all model categories

#### Mean metrics over entire test range

We now go on and compare the shift-based and NMF-based models to two baselines. Having seen that for the shift-based models using a RF regression, the use of index features is the most useful tool to reduce forecasting errors, the feature combination of shift features, index features, and target temperature is used for the RF model. For the shift-based Ridge Regression the shifts, corresponding temperature shifts, and the target temperature are leveraged.

In figure A.51 we see that the shift-based models clearly outperform the Repeat Day baseline. The two shift-based models give comparable results. In the beginning, the performance of N-BEATS is worse than the one of the Repeat Day model but steadily improves. Looking at the first two starting dates in figure A.52, we can see that, by receiving more training data, the errors of N-BEATS keep going down and its performance reaches a level similar to the shift-based models. In this representation, we can also see that with more training data, the RF model using the described feature combination outperforms the Ridge Regression model with its feature setup.

Next, the performance of NMF-based models are added to the picture. The focus is first set on the models operating on the lags, index features and target day temperature. The analysis is started by comparing the NMF-based models to only the baseline models in figure A.53. Overall, both NMF-based models stay mostly on par or below the Repeat Day baseline and also below the errors of N-BEATS. The model using a Ridge Regression shows lower errors, where the NMF model relying on a RF Regression shows RMSE values, at peaks similar to the ones of the Repeat Day baseline.

In figure A.54 gives more details, by showing the errors of each starting date setup individually. Apart from the first part of the first starting date with high errors, there are hardly any cases where the NMF-based models show higher errors than the Repeat Day baseline. Especially in the summer and fall months, the NMF-based models show far lower errors than both baseline models, staying below the error curves of RepeatLastDayOfWeek and N-BEATS. Additionally, in contrast to the shift-based model, even in winter the NMF-based models consistently show lower RMSE values than the N-BEATS model.

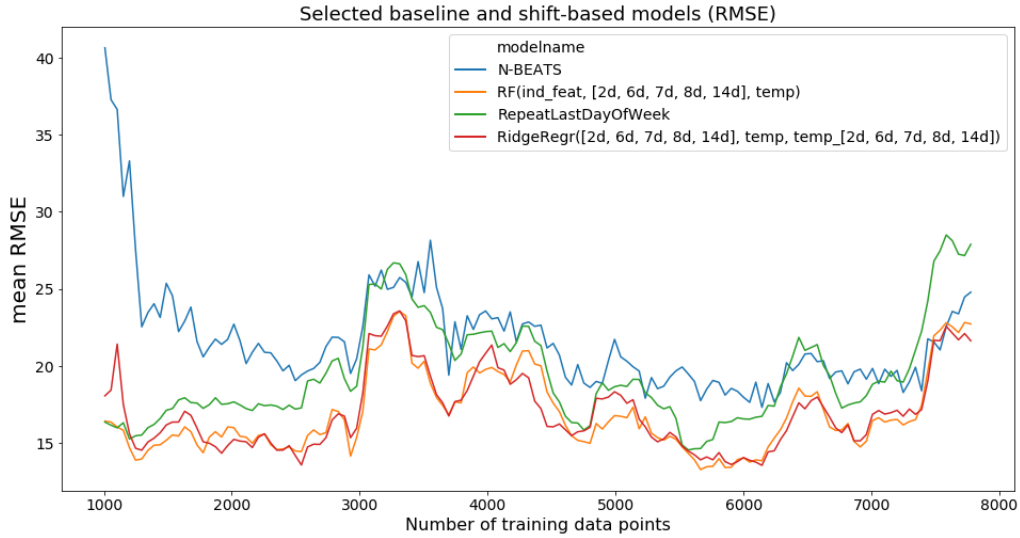



Figure A.51.: RMSE of selected baseline and shift-based models, averaged over all four starting date setups. 48 data points correspond to one day.

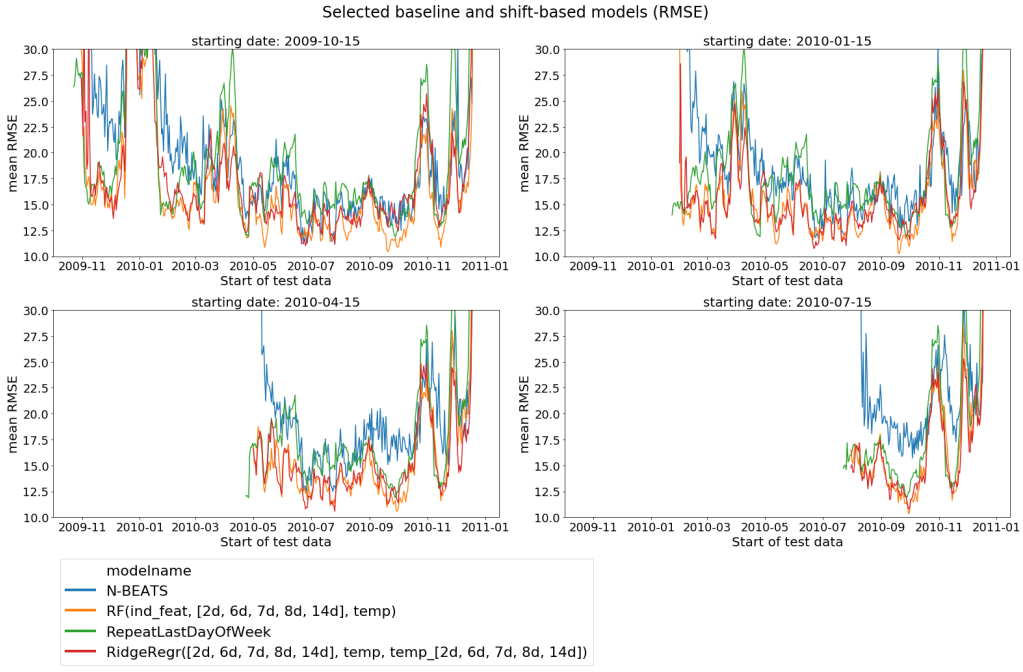



Figure A.52.: RMSE of selected baseline and shift-based models for all four starting date setups (full plots in appendix figure A.82).

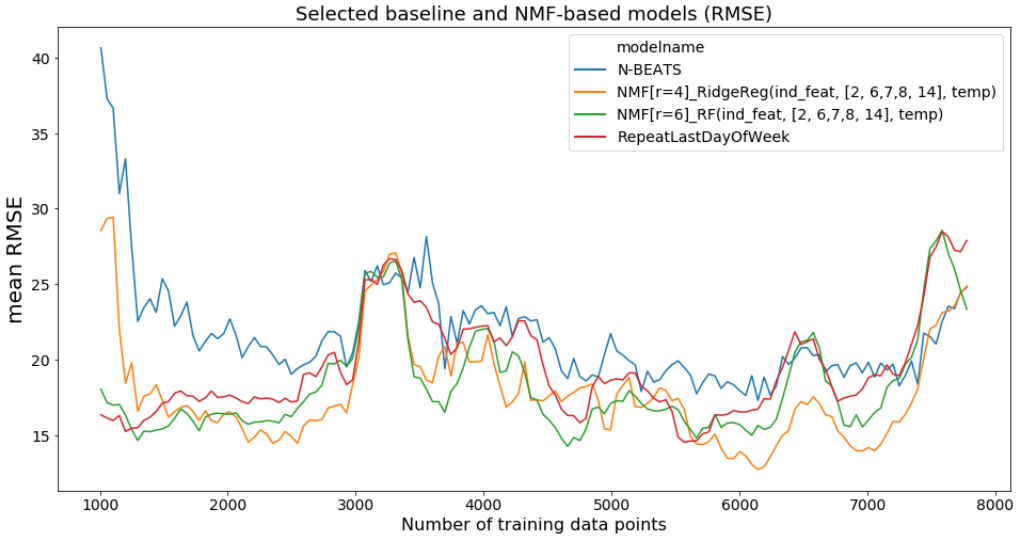



Figure A.53.: RMSE of selected baseline and NMF-based models, averaged over all four starting date setups. 48 data points correspond to one day.

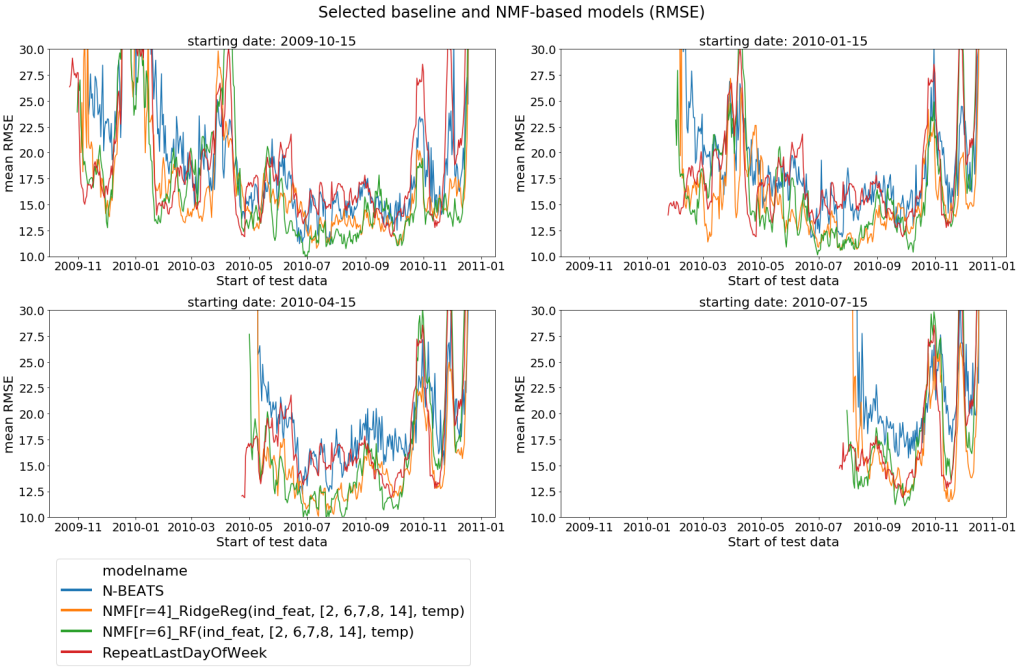



Figure A.54.: RMSE of selected baseline and NMF-based models for all four starting date setups (full plots in appendix figure A.83).

After getting a first impression of the performance of the NMF-based models, by comparing them to the baselines, now, NMF-based and shift-based models are compared.

For this, first, a comparison of shift-based models and NMF-based models operating on the conceptionally same features, i.e. the same shifts/lags without any additional features, is done. For an easier visualization only the $r = 6$ case is used for the NMF models. Looking at the mean errors, averaged over all four starting dates in A.55, we can see that the NMF-based models generally have higher errors than the shift-based models. Going into more detail and looking into the error plots for each starting date individually in figure A.56, we see that the NMF models mostly show slightly higher errors than the corresponding shift models, but the differences are not as large as in spring 2010, captured by the first starting date. However, from the starting dates 2009-10-15 and 2010-01-15 we can see that after September 2010, the NMF-based Ridge Regression model manages to achieve errors comparable to the ones of the shift-based models and even shows the lowest errors in early 2010 winter. Also when looking at the peaks at the beginning of December 2010 and at the end of October 2010 in all four starting dates, we see that the more data the NMF models got, the lower the peaks are. For the 2010-07-15 starting date, the peaks are still clearly higher than for the shift-based models. For the 2010-01-15 starting date, they are already of similar height and for the earliest starting date 2009-10-15, the NMF models achieve lower errors for the December peak.

However, the models would not only use the shifts/lags as features. Therefore, the analysis is repeated with the models operating on the shifts/lags, target temperature and index features. Including these features has a greater effect on the NMF-based models, bringing their error values closer to the ones of the shift-based models. In figure A.57 the NMF-based model using the Ridge Regression shows errors that are in many regions on par with the shift model errors. Looking at the individual starting dates in figure A.58 we see that especially with more training data the NMF models are at least on par with the shift models and even outperform them in some regions. However, with less data, they often show higher RMSE values than the shift-based counterparts.

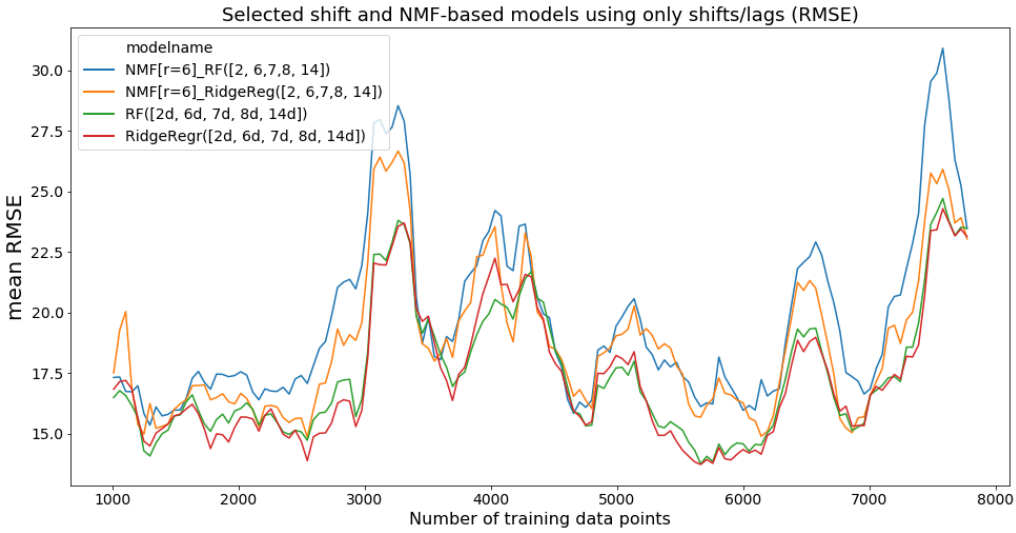



Figure A.55.: RMSE of shift-based and NMF-based models using only shifts/lags, averaged over all four starting date setups. 48 data points correspond to one day.

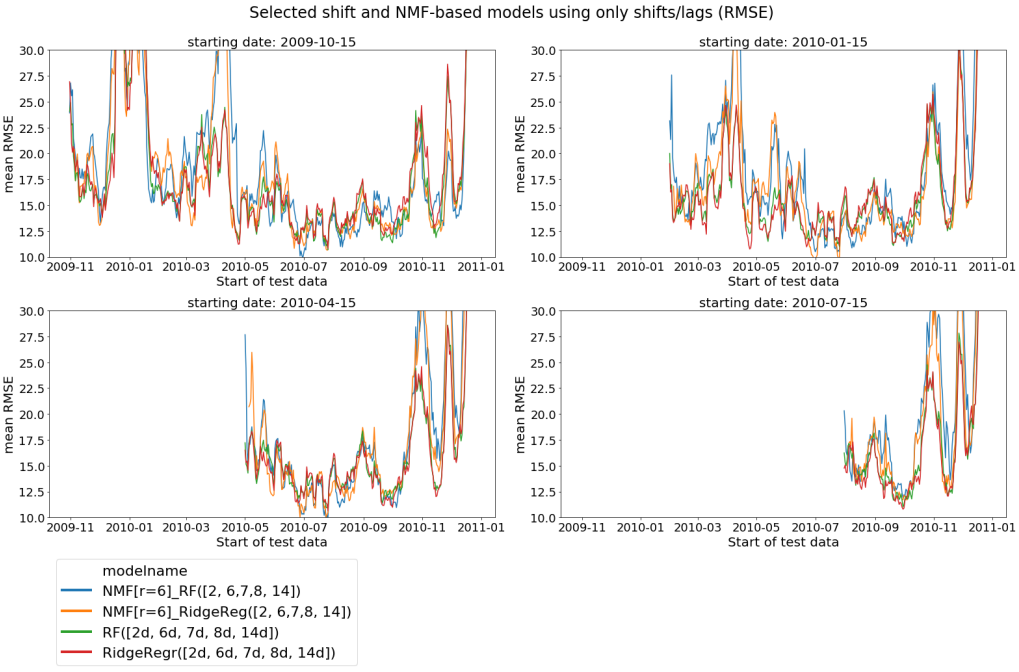



Figure A.56.: RMSE of shift-based and NMF-based models using only shifts/lags, for all four starting date setups (full plots in appendix figure A.84).

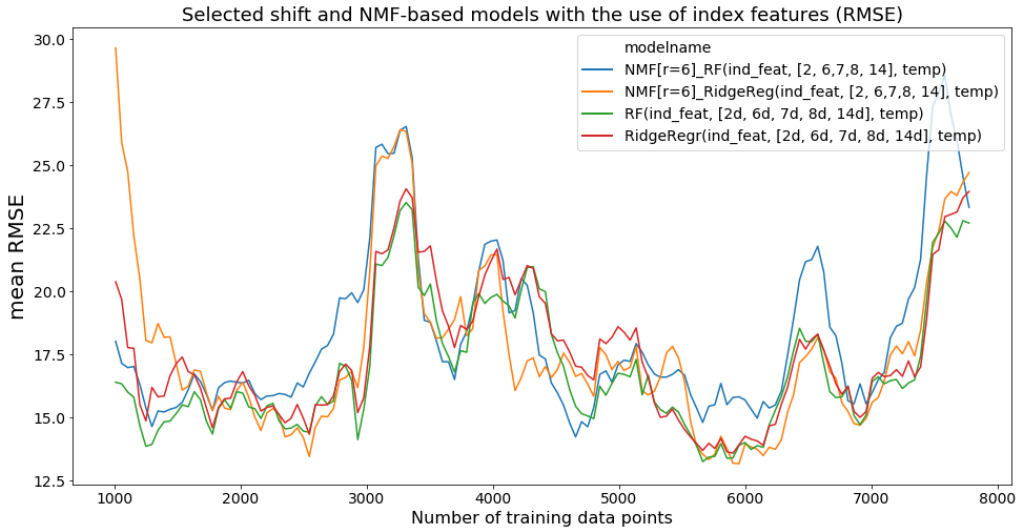



Figure A.57.: RMSE of shift-based and NMF-based models using target temperature and index features, averaged over all four starting date setups. 48 data points correspond to one day.

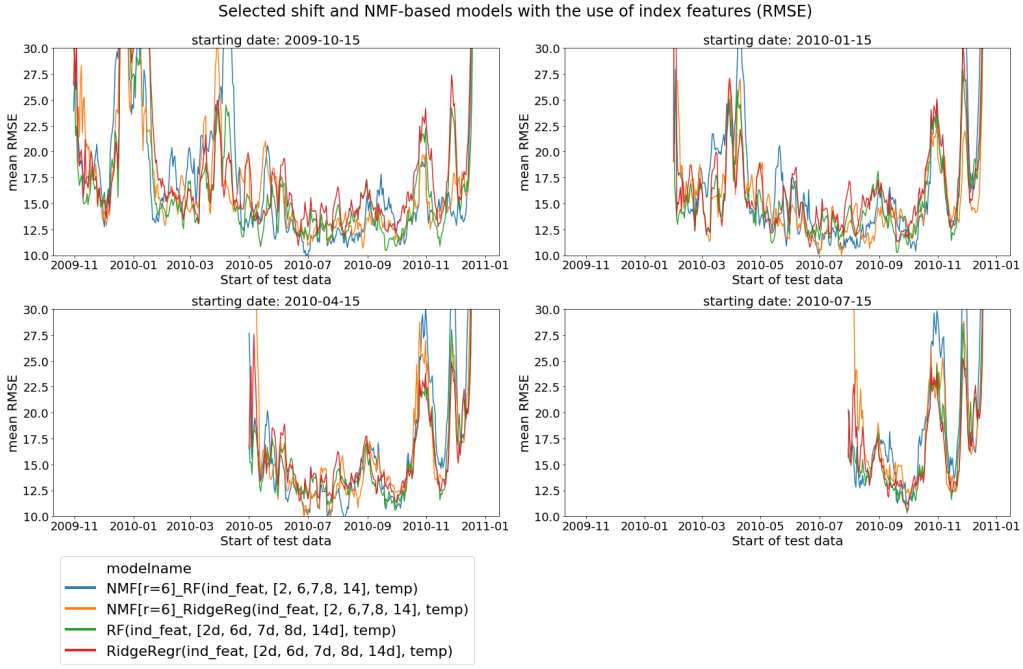



Figure A.58.: RMSE of shift-based and NMF-based models using target temperature and index features, for all four starting date setups (full plots in appendix figure A.85).

One more thing to analyze is how the errors of models are formed. Are models constantly overestimating or underestimating the load demand or does this change? To answer this question, one may look at the models' Sum Of Differences (SOD) (see section 5.6), averaged over the seven days of test data. For this analysis, only the Repeat Day baseline as well as one NMF-based and one shift-based model are chosen, as most configurations show a similar error behavior.

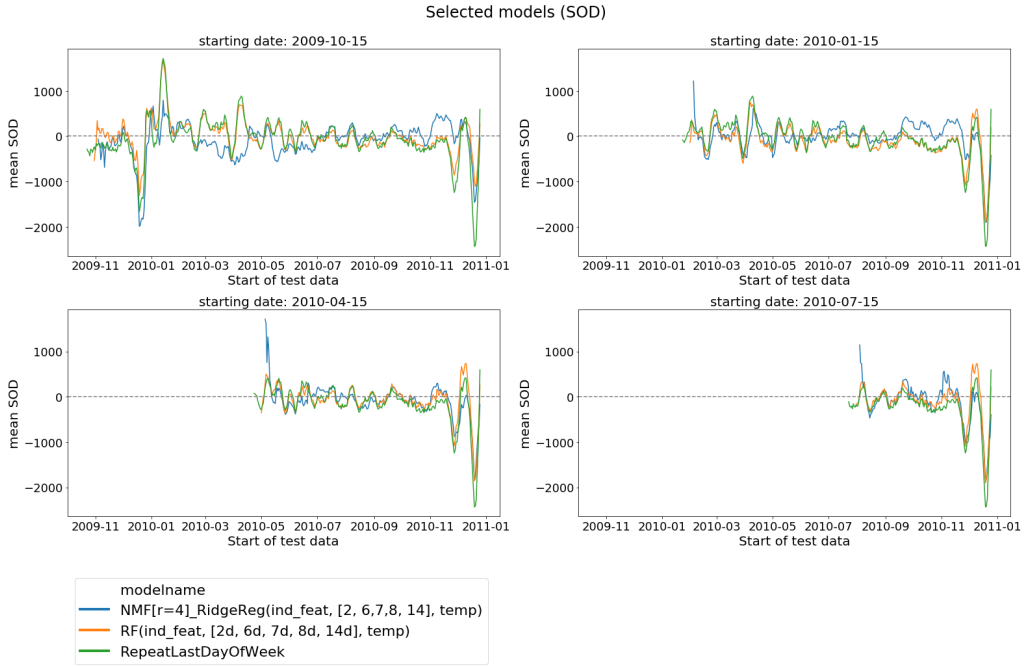



Figure A.59.: Sum Of Differences of selected models, for all four starting date setups.

In figure A.59 we can see that all three models included in the plot do neither only overestimate nor underestimate the targets days' power demands. Overall, all models tend to underestimate the target more often than they overestimate.

Lastly, to be able to compare the models' performances to the ones of other works not using the same data set, the MAPE values of selected models are plotted in figure A.60. Additionally, even though this is not the idea of the iterations, for the sake of comparing the errors, the average MAPE values were also calculated and are presented in table A.1.

| Model | Starting date | Average MAPE value |
|---|---|---|
| N-BEATS | 2009-10-15 | 9.76 |
| | 2010-01-15 | 9.62 |
| | 2010-04-15 | 10.34 |
| | 2010-07-15 | 11.81 |
| | Mean | 10.38 |
| NMF[n=4]_RidgeReg(ind_feat, [2, 6,7,8, 14], temp) | 2009-10-15 | 7.91 |
| | 2010-01-15 | 7.52 |
| | 2010-04-15 | 7.57 |
| | 2010-07-15 | 8.31 |
| | Mean | 7.83 |
| RF(ind_feat, [2d, 6d, 7d, 8d, 14d], temp) | 2009-10-15 | 7.43 |
| | 2010-01-15 | 7.20 |
| | 2010-04-15 | 7.23 |
| | 2010-07-15 | 7.59 |
| | Mean | 7.36 |
| RepeatLastDayOfWeek | 2009-10-15 | 8.64 |
| | 2010-01-15 | 8.20 |
| | 2010-04-15 | 8.25 |
| | 2010-07-15 | 8.37 |
| | Mean | 8.37 |

Table A.1.: Average MAPE values of selected models for all starting dates and their mean value (not weighted).

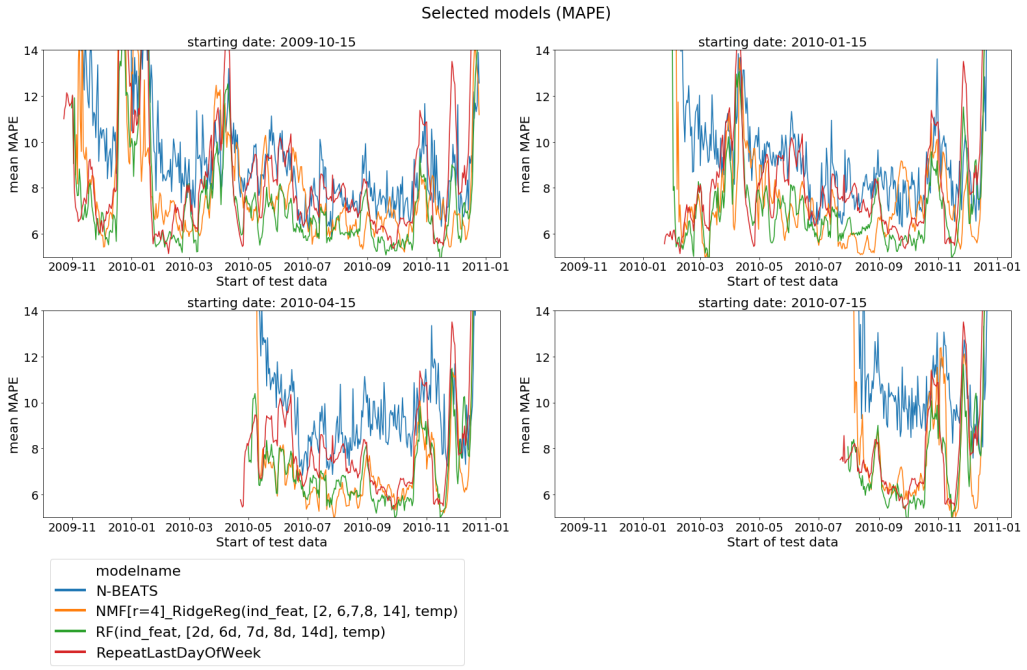



Figure A.60.: MAPE of selected models for all four starting date setups (full plots in appendix figure A.86).

### Metrics for each of the seven days of test range

To analyze the errors across all seven days of the test range, the following plots in figures A.61, A.62, A.63 and A.64 show the RMSE errors for selected models with one line for each test day. Here only models not already shown in the results chapter 9 are presented.

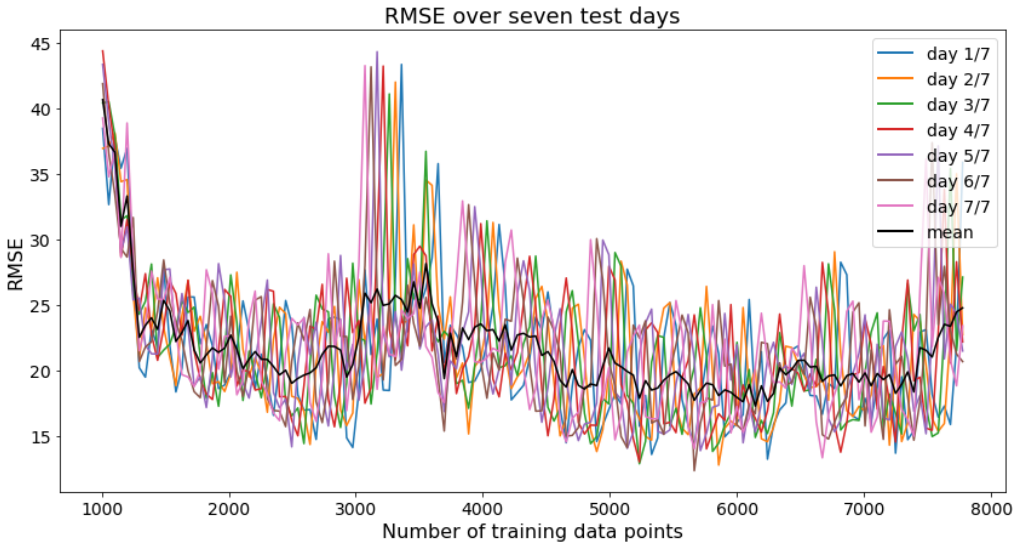



Figure A.61.: RMSE of N-BEATS model, averaged over all four starting date setups, for each of the seven test days. 48 data points correspond to one day.

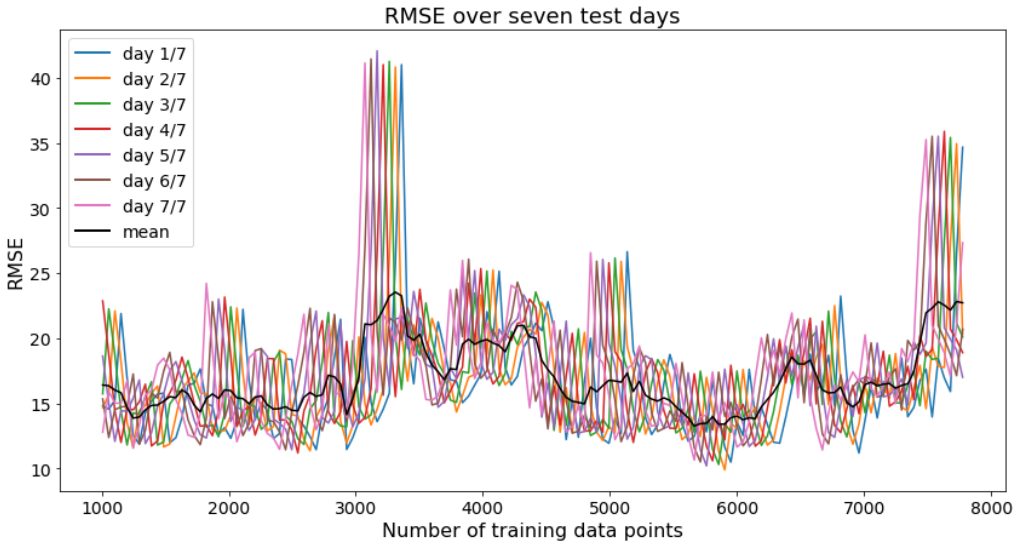



Figure A.62.: RMSE of RF model using shifts, target temperature and index features, averaged over all four starting date setups, for each of the seven test days. 48 data points correspond to one day.

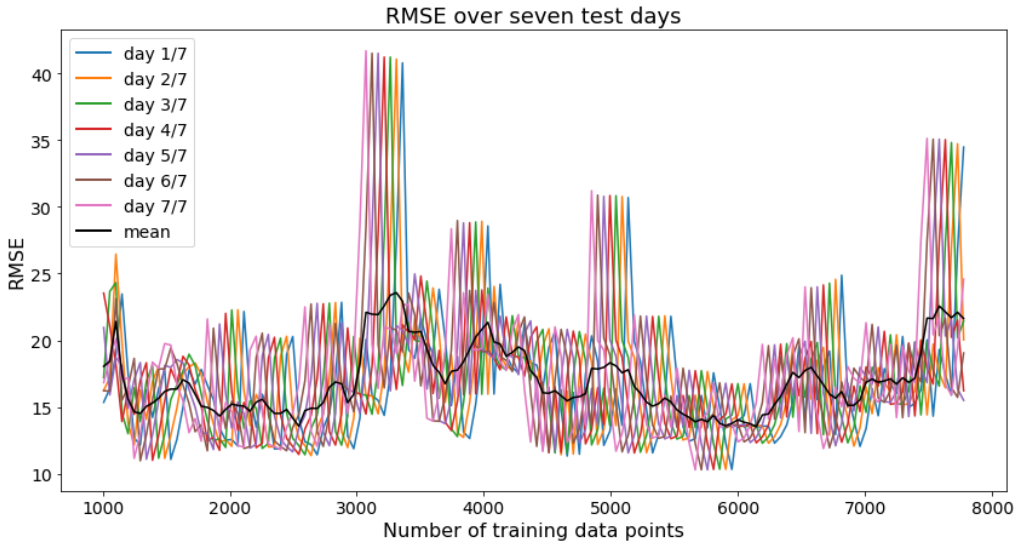



Figure A.63.: RMSE of Ridge Regression model using shifts, target temperature and temperature shifts, averaged over all four starting date setups, for each of the seven test days. 48 data points correspond to one day.

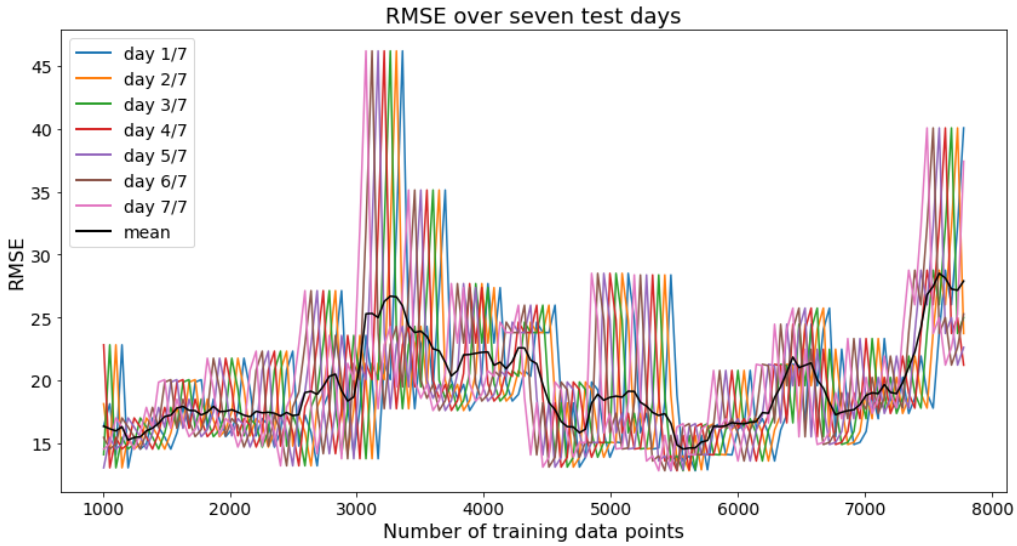



Figure A.64.: RMSE of RepeatLastDayOfWeekForecaster, averaged over all four starting date setups, for each of the seven test days. 48 data points correspond to one day.

## A.3. Non-zoomed plots for results

In the thesis, some plots were presented zoomed in on a certain range. In this section, the full non-zoomed plots are presented.

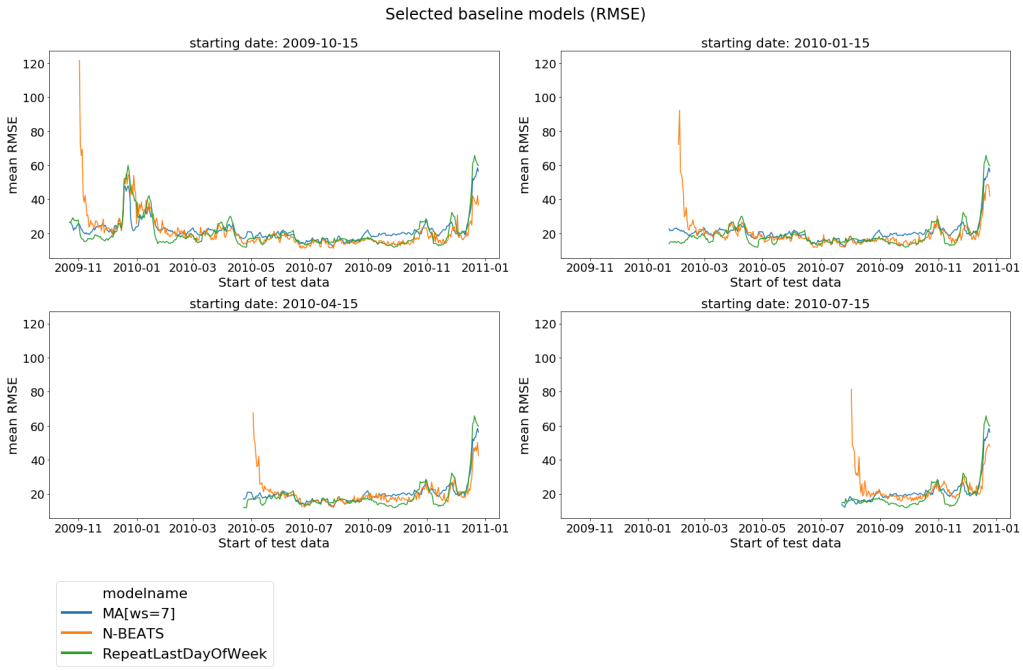



Figure A.65.: RMSE of selected baseline models for all four starting date setups.

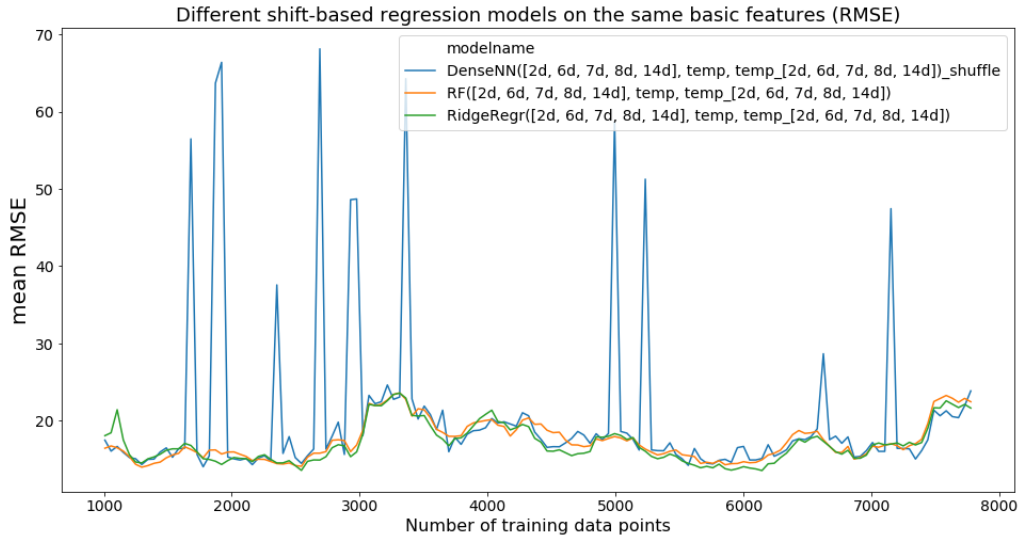



Figure A.66.: RMSE of shift-based models with different regression models, averaged over all four starting date setups. 48 data points correspond to one day.

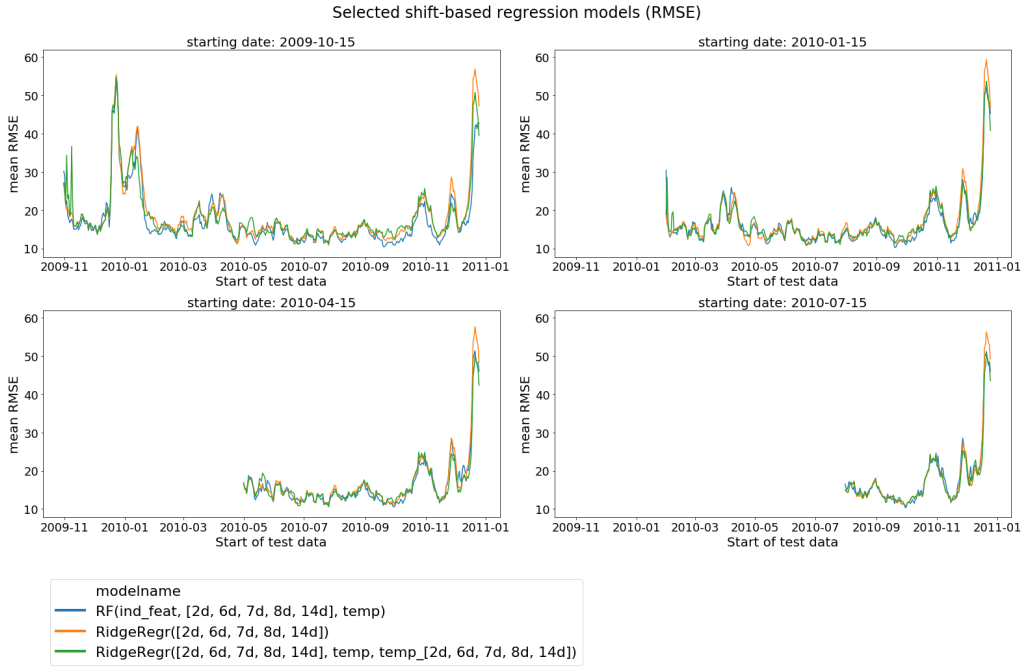



Figure A.67.: RMSE of selected shift-based models for all four starting date setups.

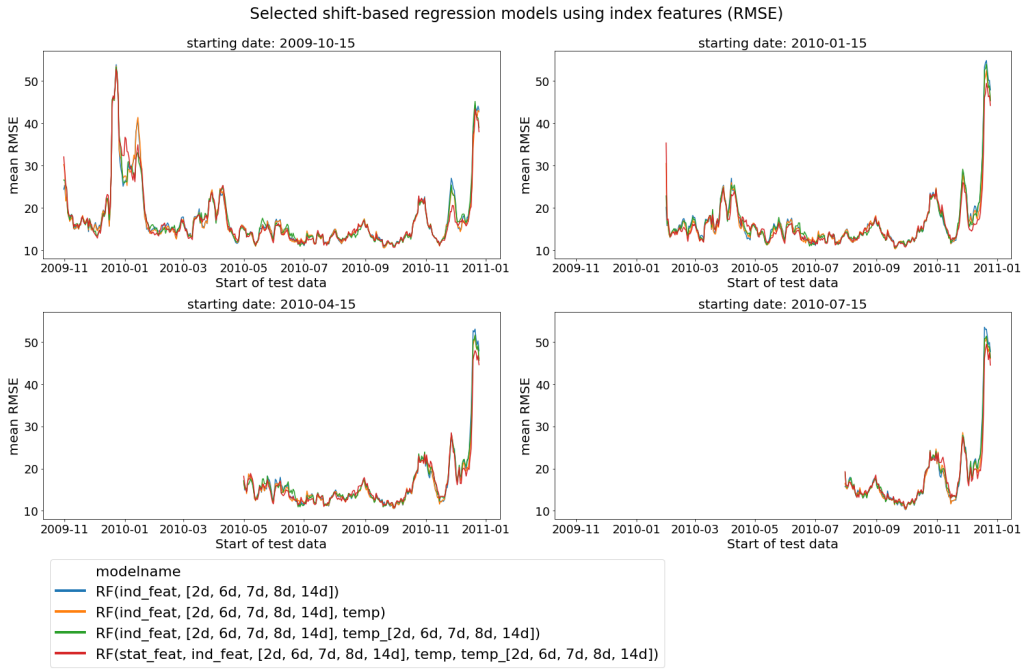



Figure A.68.: RMSE of shift-based RF models using index features, for all four starting date setups.

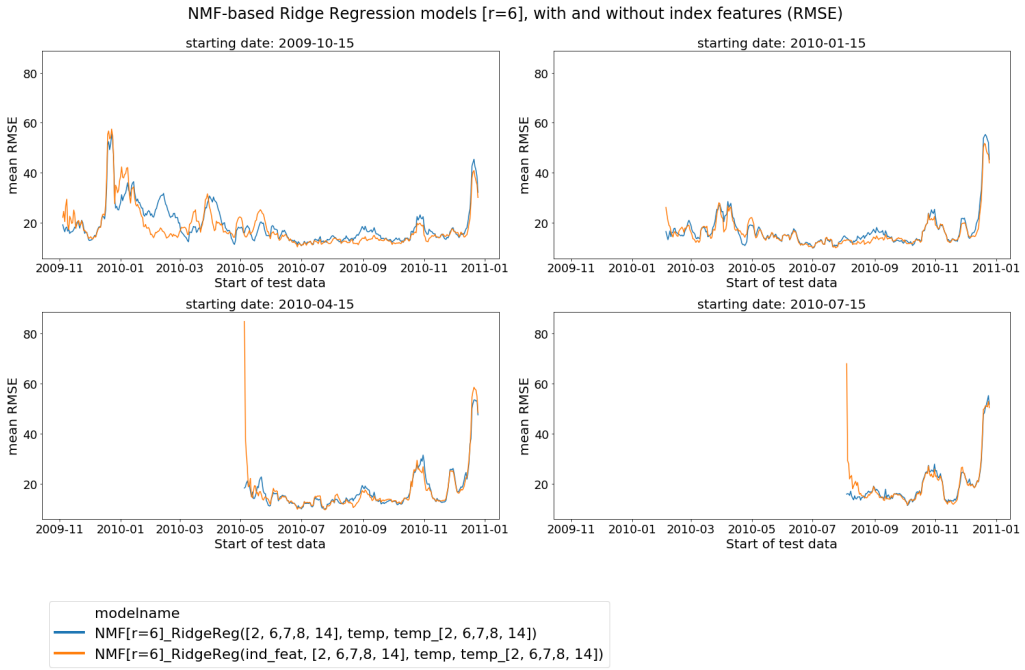



Figure A.69.: RMSE of NMF-based Ridge Regression models for $r = 6$, with and without index features, for all four starting date setups.

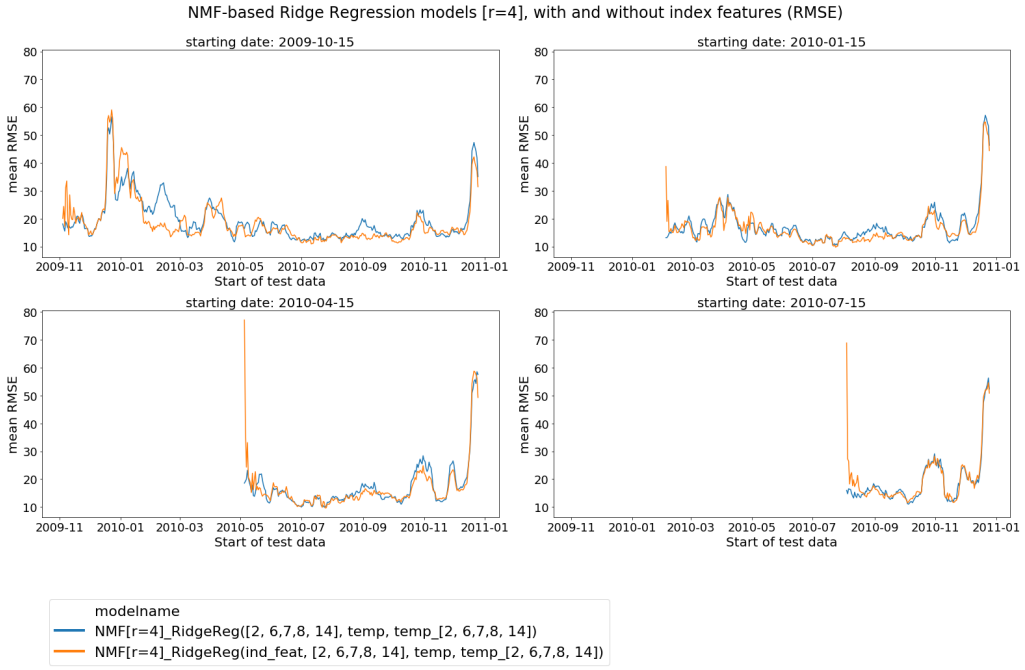



Figure A.70.: RMSE of NMF-based Ridge Regression models for $r = 4$, with and without index features, for all four starting date setups.

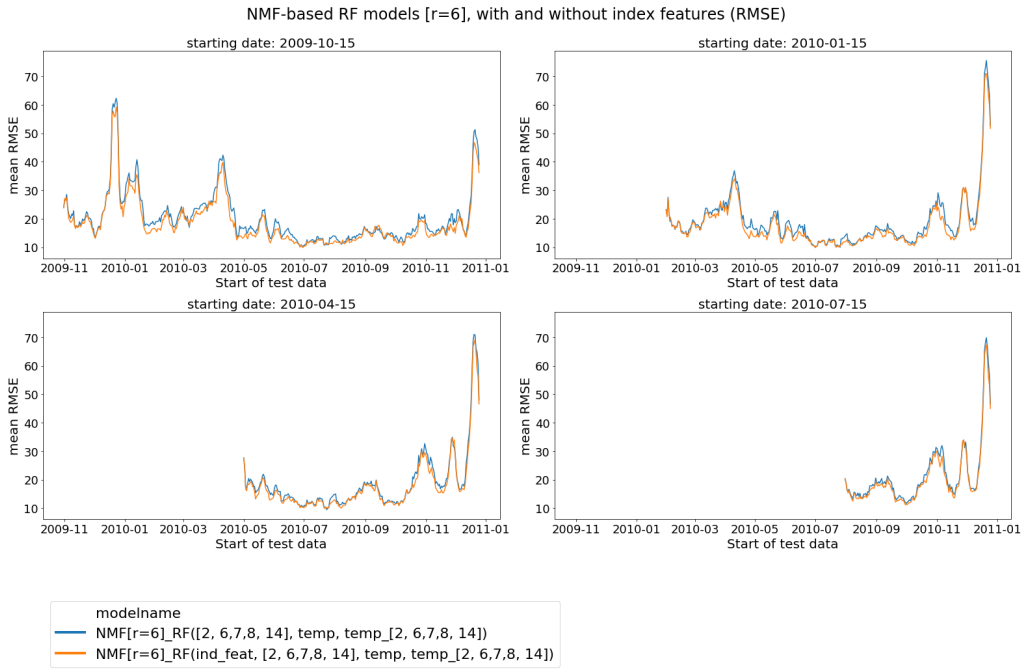



Figure A.71.: RMSE of NMF-based RF models for $r = 6$, with and without index features, for all four starting date setups.

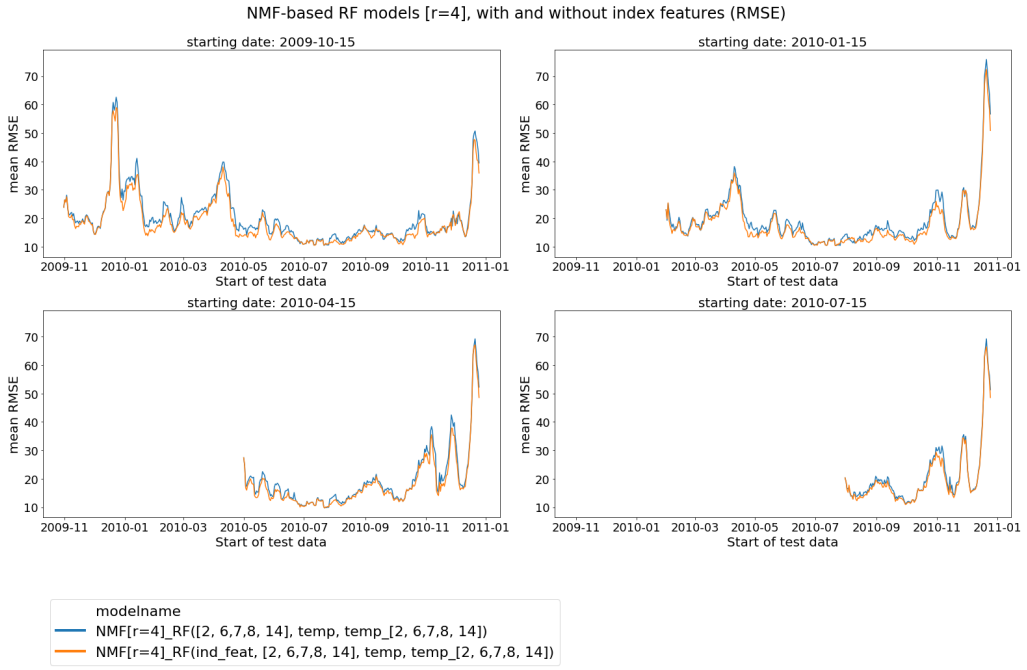



Figure A.72.: RMSE of NMF-based RF models for $r = 4$, with and without index features, for all four starting date setups.

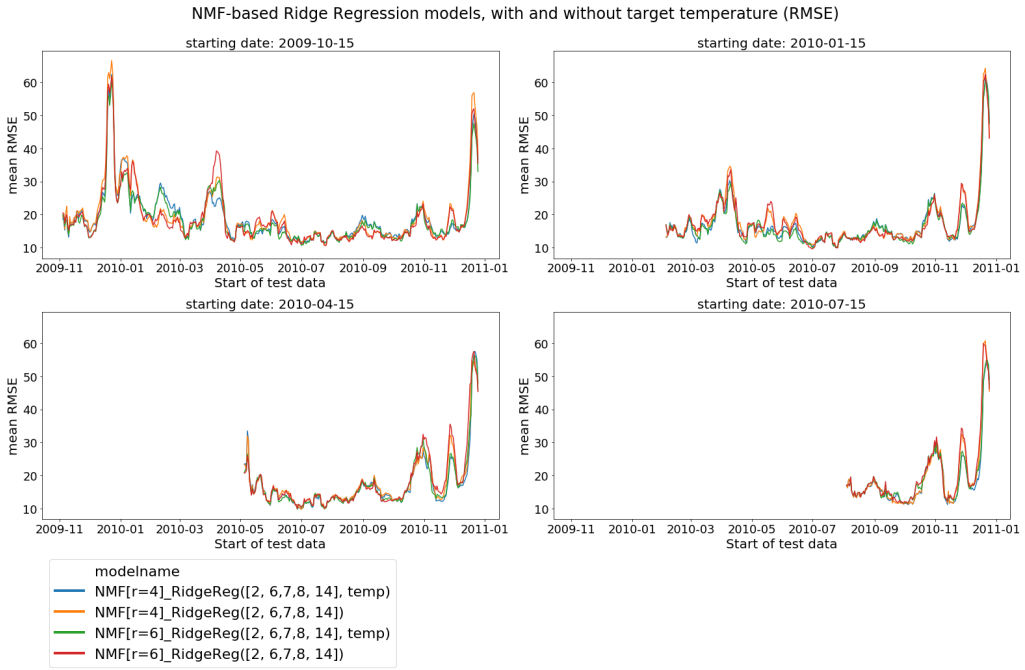



Figure A.73.: RMSE of NMF-based Ridge Regression models, with and without target temperature, for all four starting date setups.

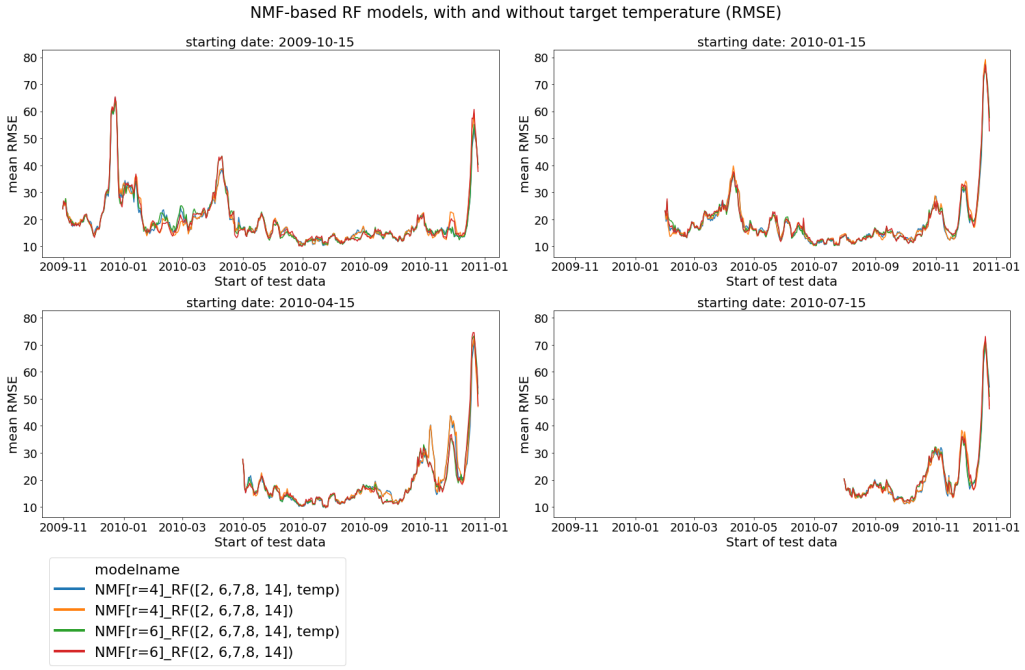



Figure A.74.: RMSE of NMF-based RF models, with and without target temperature, for all four starting date setups.

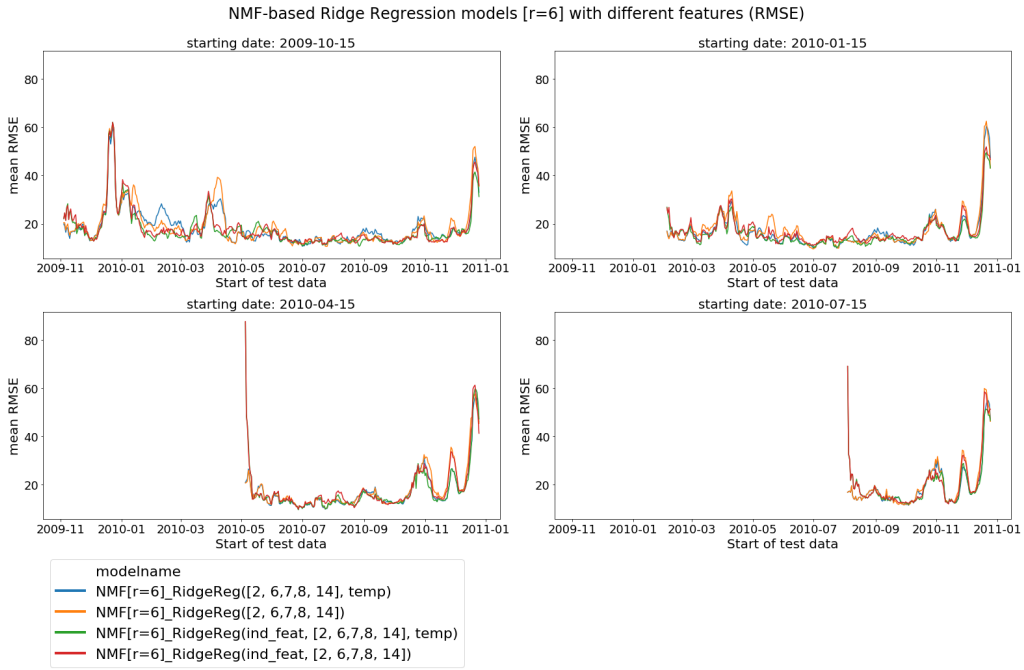



Figure A.75.: RMSE of NMF-based Ridge Regression models for $r = 6$, with different features, for all four starting date setups.

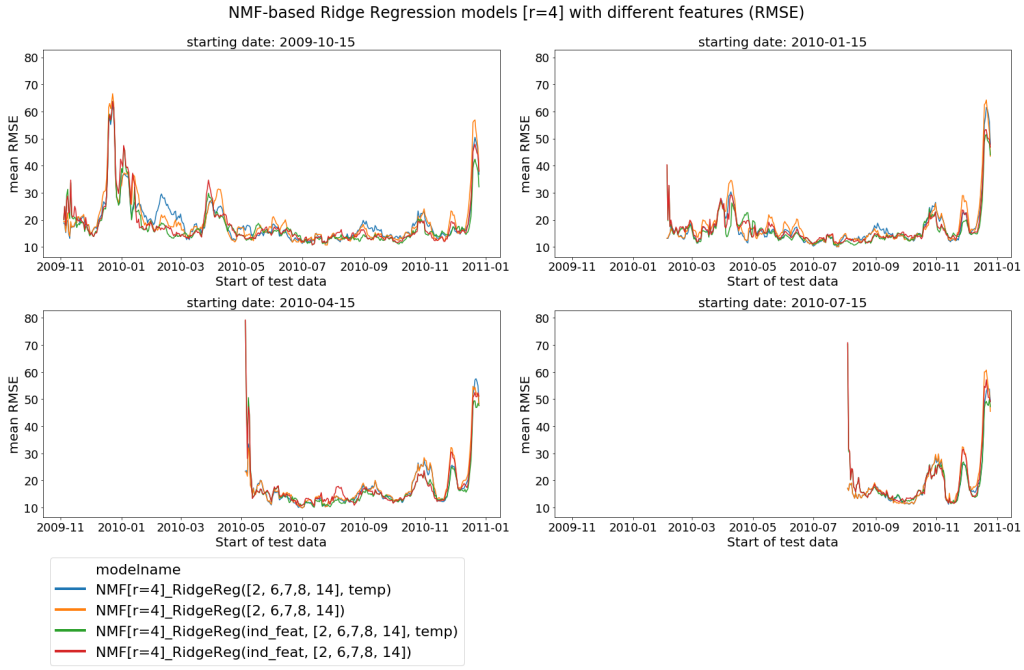



Figure A.76.: RMSE of NMF-based Ridge Regression models for $r = 4$, with different features, for all four starting date setups.

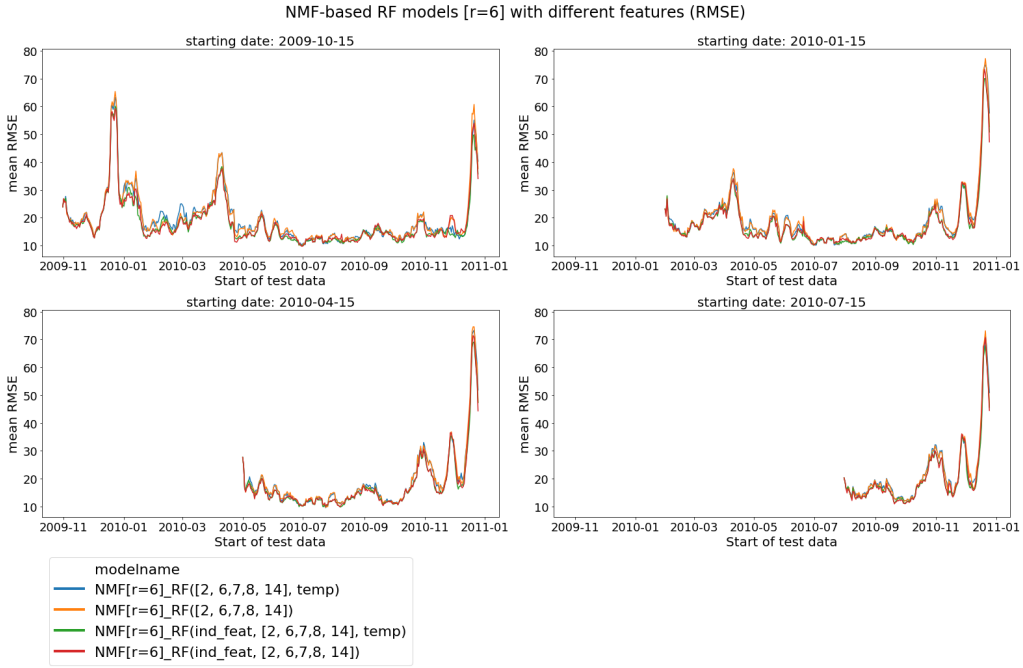



Figure A.77.: RMSE of NMF-based RF models for $r = 6$, with different features, for all four starting date setups.

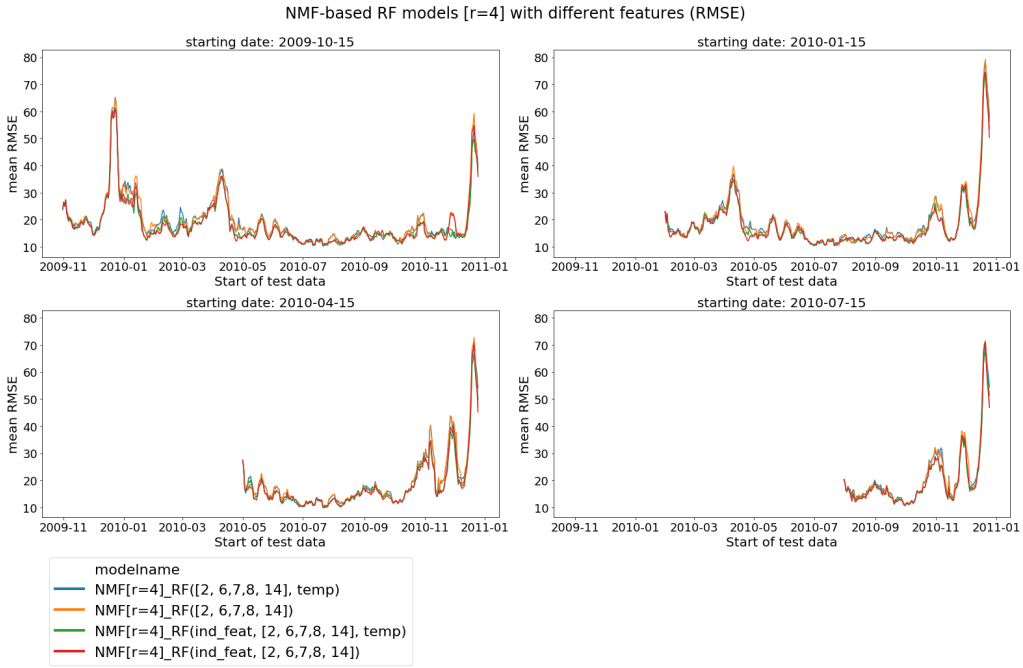



Figure A.78.: RMSE of NMF-based RF models for $r = 4$, with different features, for all four starting date setups.

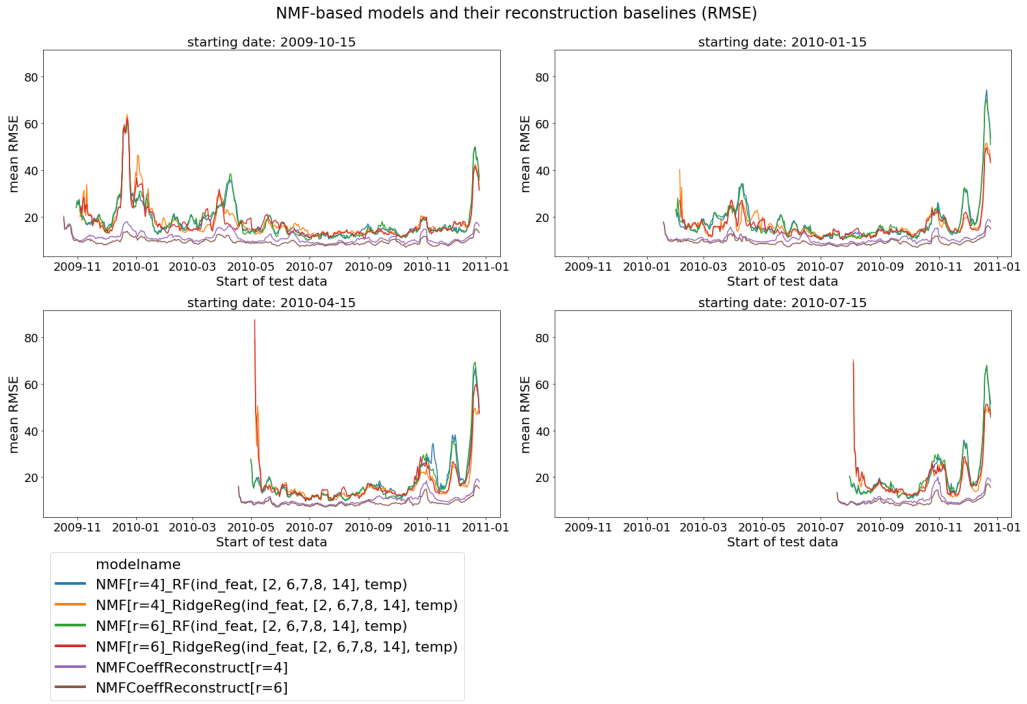



Figure A.79.: RMSE of NMF-based models and their reconstruction baselines for all four starting date setups.

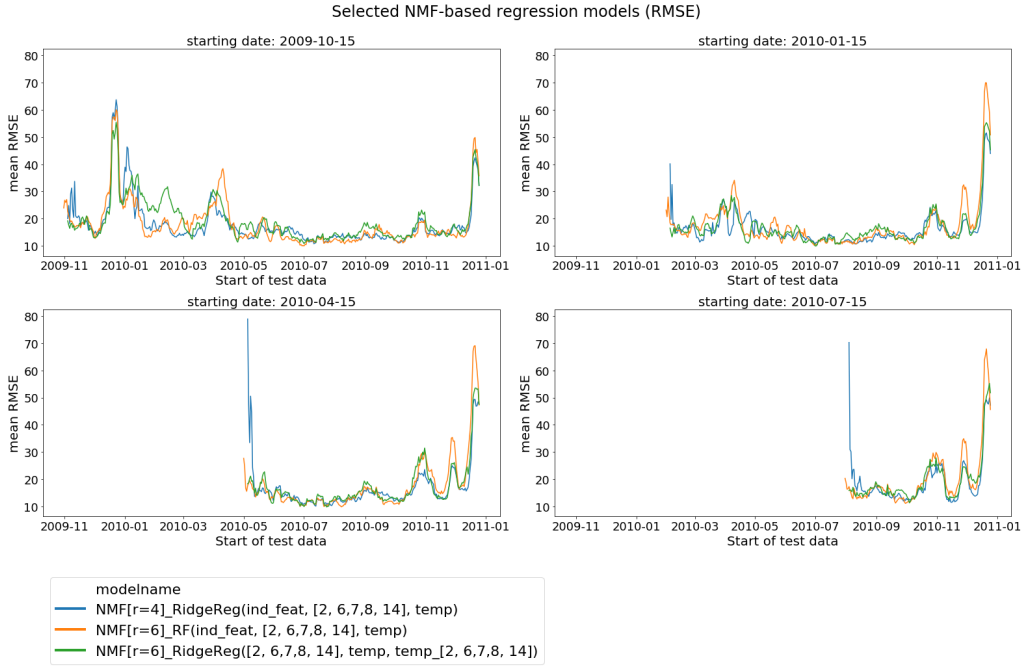



Figure A.80.: RMSE of selected NMF-based models for all four starting date setups.

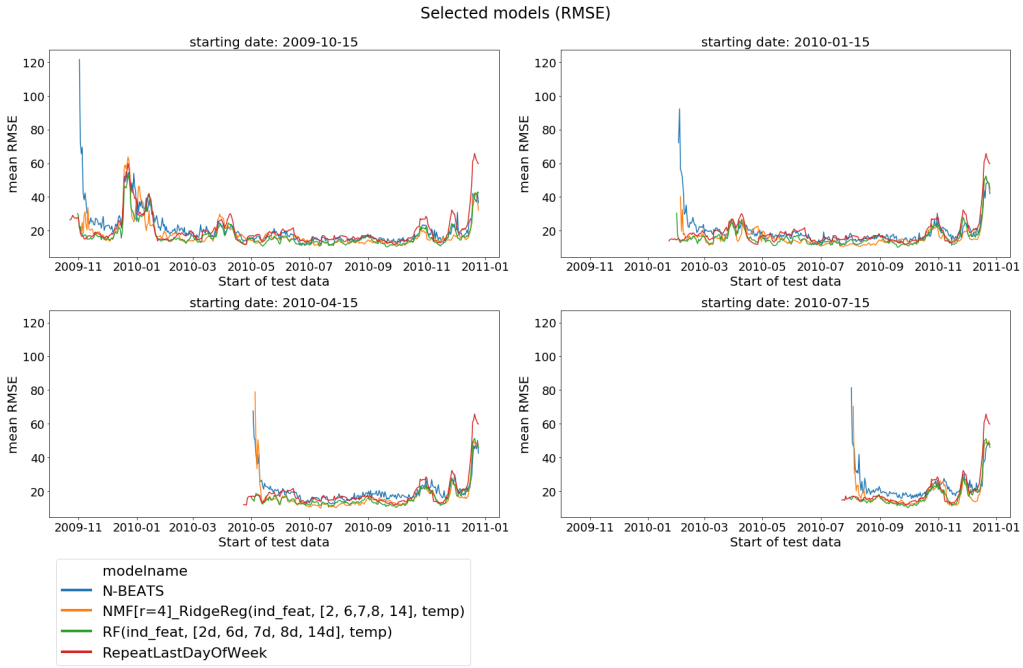



Figure A.81.: RMSE of selected models for all four starting date setups.

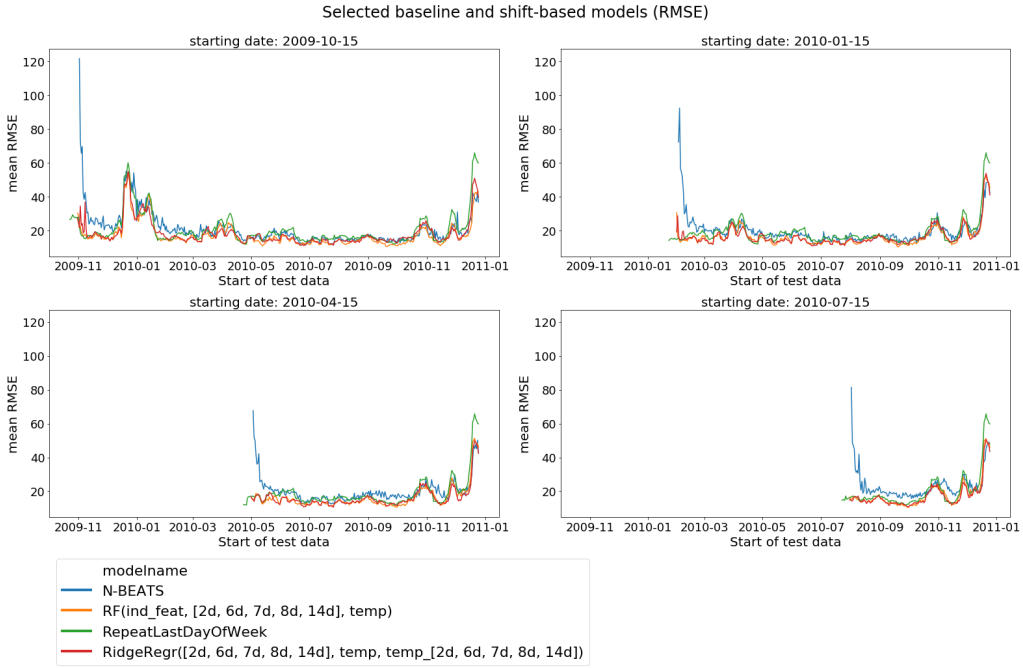



Figure A.82.: RMSE of selected baseline and shift-based models for all four starting date setups.

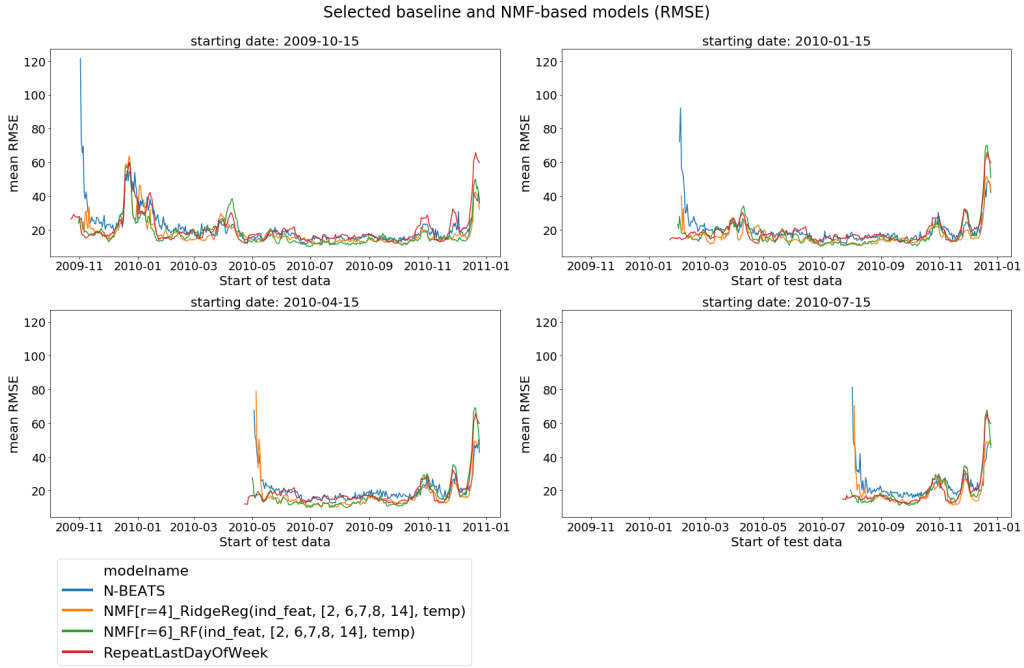



Figure A.83.: RMSE of selected baseline and NMF-based models for all four starting date setups.

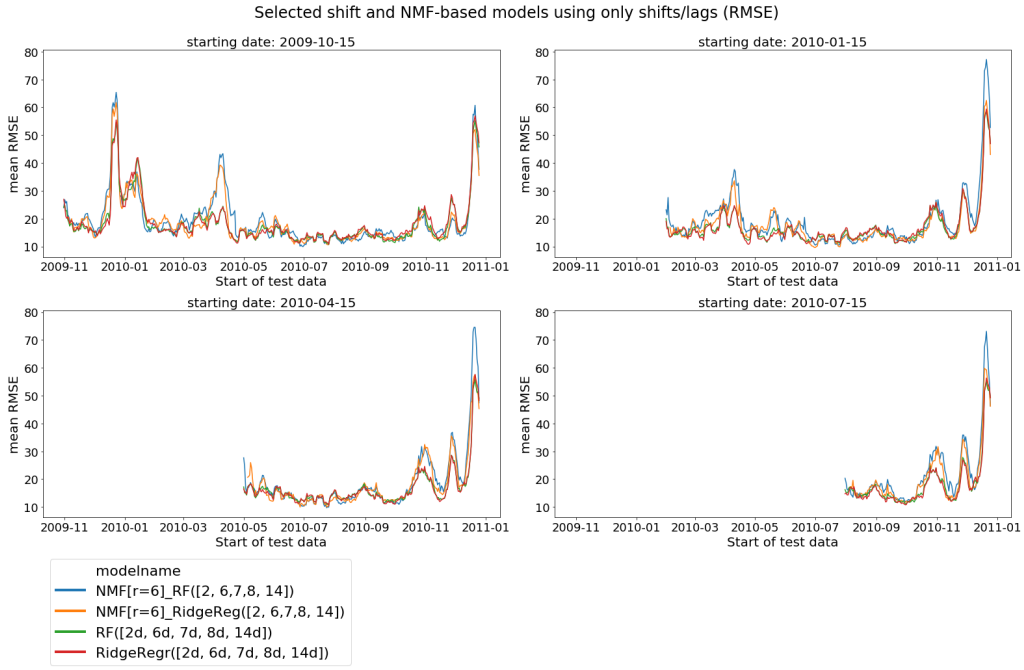



Figure A.84.: RMSE of shift-based and NMF-based models, using only shifts/lags, for all four starting date setups.

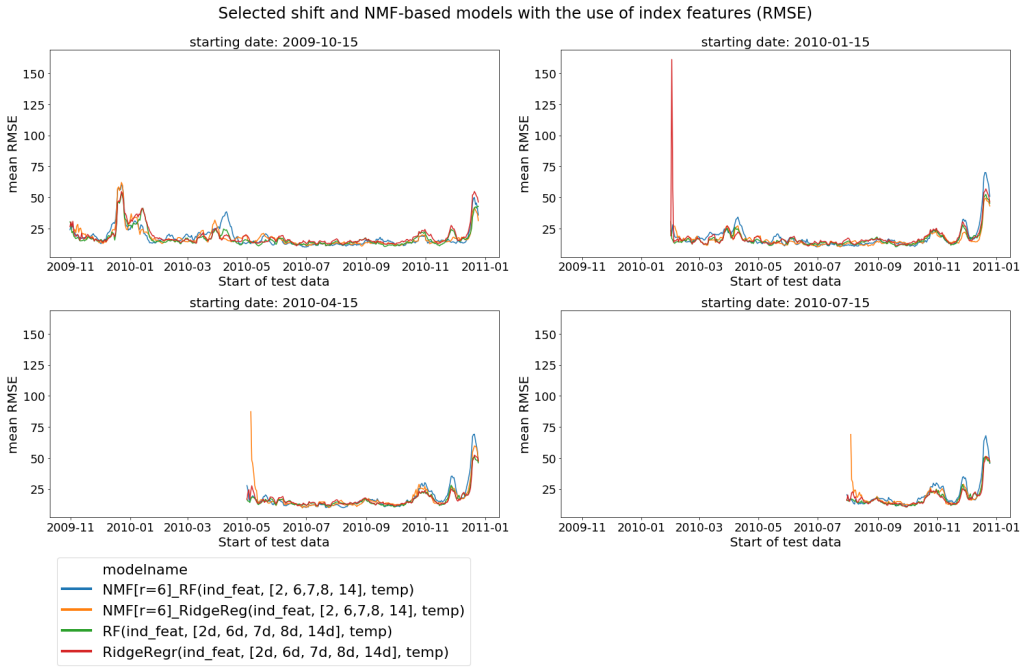



Figure A.85.: RMSE of shift-based and NMF-based models, using target temperature and index features, for all four starting date setups.

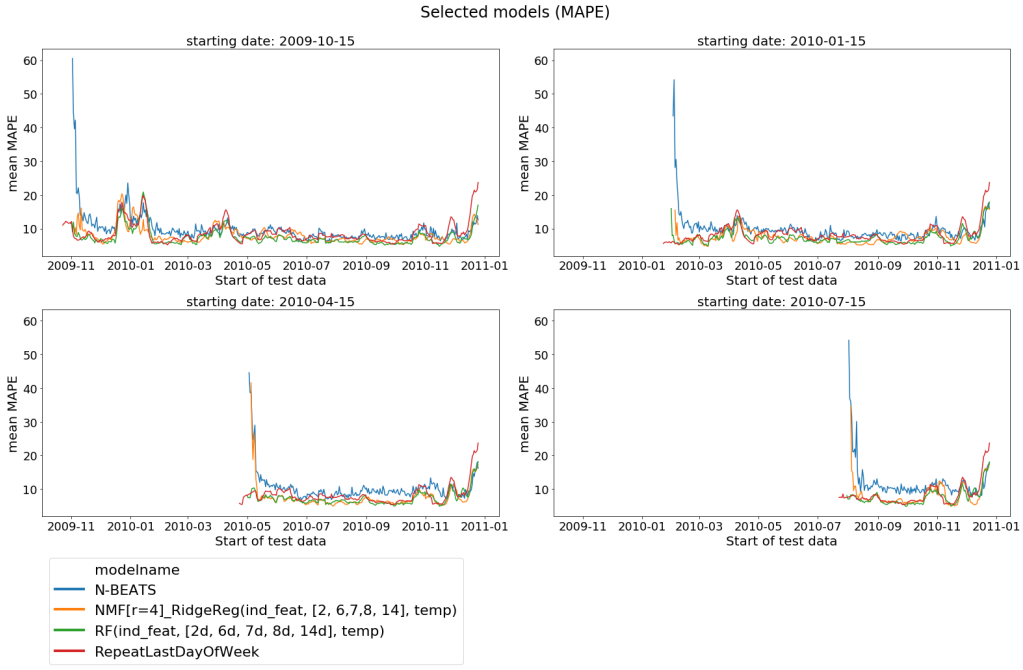



Figure A.86.: MAPE of selected models for all four starting date setups.

## A.4. Used resident-IDs

350 time series of individual residential households were summed up to model the power usage of a neighborhood or a small city district. For this sum, out of the 4225 residential households, 350 with the following IDs were used:

1002, 1014, 1018, 1022, 1027, 1030, 1035, 1036, 1058, 1059, 1060, 1073, 1074, 1076, 1107, 1119, 1120, 1130, 1141, 1144, 1157, 1164, 1171, 1176, 1179, 1207, 1212, 1217, 1225, 1229, 1239, 1240, 1246, 1255, 1257, 1262, 1268, 1274, 1277, 1281, 1286, 1294, 1301, 1312, 1316, 1317, 1331, 1336, 1354, 1364, 1365, 1371, 1376, 1379, 1388, 1425, 1428, 1435, 1440, 1463, 1466, 1480, 1492, 1494, 1498, 1502, 1505, 1507, 1517, 1530, 1532, 1537, 1539, 1546, 1548, 1551, 1561, 1564, 1567, 1573, 1575, 1592, 1599, 1604, 1607, 1609, 1624, 1635, 1639, 1650, 1657, 1660, 1661, 1664, 1665, 1666, 1667, 1670, 1673, 1675, 1677, 1679, 1682, 1695, 1700, 1711, 1716, 1720, 1722, 1724, 1727, 1736, 1742, 1752, 1754, 1760, 1763, 1765, 1768, 1789, 1797, 1804, 1807, 1808, 1809, 1831, 1840, 1842, 1843, 1844, 1854, 1859, 1861, 1874, 1877, 1883, 1884, 1885, 1887, 1889, 1890, 1894, 1903, 1905, 1906, 1917, 1922, 1942, 1948, 1953, 1954, 1964, 1969, 1975, 1984, 1989, 2003, 2018, 2022, 2023, 2029, 2033, 2036, 2041, 2046, 2049, 2055, 2065, 2080, 2095, 2099, 2103, 2104, 2105, 2106, 2109, 2110, 2121, 2140, 2144, 2149, 2160, 2185, 2194, 2196, 2231, 2235, 2238, 2259, 2260, 2263, 2264, 2265, 2268, 2278, 2289, 2293, 2312, 2314, 2331, 2345, 2347, 2349, 2354, 2370, 2377, 2381, 2387, 2404, 2405, 2406, 2407, 2409, 2423, 2424, 2436, 2438, 2463, 2464, 2485, 2489, 2490, 2512, 2519, 2522, 2526, 2529, 2532, 2541, 2547, 2553, 2562, 2574, 2591, 2594, 2597, 2605, 2607, 2621, 2623, 2635, 2638, 2640, 2646, 2654, 2669, 2678, 2684, 2698, 2700, 2701, 2702, 2707, 2708, 2710, 2725, 2730, 2732, 2733, 2742, 2744, 2749, 2757, 2774, 2789, 2793, 2803, 2804, 2811, 2832, 2834, 2835, 2838, 2857, 2872, 2873, 2878, 2892, 2905, 2918, 2926, 2945, 2956, 2958, 2967, 2968, 2971, 2986, 2987, 2988, 3003, 3009, 3010, 3019, 3033, 3043, 3050, 3052, 3055, 3059, 3061, 3077, 3080, 3087, 3091, 3098, 3105, 3110, 3111, 3120, 3124, 3144, 3152, 3165, 3168, 3177, 3178, 3180, 3184, 3211, 3217, 3221, 3232, 3234, 3245, 3246, 3262, 3281, 3293, 3295, 3302, 3306, 3314, 3326, 3333, 3335, 3337, 3339, 3346, 3355, 3358, 3367, 3371, 3387, 3393, 3397, 3403, 3435, 3436, 3437

## A.5. Weather stations

List of the used Met Éireann weather stations:

| Station Name | Station Height | Latitude | Longitude |
|---|---|---|---|
| PHOENIX PARK | 48m | 53.364 | -6.350 |
| MACE HEAD | 21m | 53.326 | -9.901 |
| OAK PARK | 62m | 52.861 | -6.915 |
| SHANNON AIRPORT | 15m | 52.690 | -8.918 |
| DUBLIN AIRPORT | 71m | 53.428 | -6.241 |
| MOORE PARK | 46m | 52.164 | -8.264 |
| BALLYHAISE | 78m | 54.051 | -7.310 |
| SherkinIsland | 21m | 51.476 | -9.428 |
| MULLINGAR | 101m | 53.537 | -7.362 |
| ROCHES POINT | 40m | 51.793 | -8.244 |
| NEWPORT | 22m | 53.922 | -9.572 |
| MARKREE | 34m | 54.175 | -8.456 |
| DUNSANY | 83m | 53.516 | -6.660 |
| GURTEEN | 75m | 53.053 | -8.009 |
| JOHNSTOWNII | 62m | 52.298 | -6.497 |
| MT DILLON | 39m | 53.727 | -7.981 |
| CASEMENT | 91m | 53.306 | -6.439 |
| CORK AIRPORT | 155m | 51.847 | -8.486 |
| KNOCK AIRPORT | 201m | 53.906 | -8.817 |

# List of Figures

# List of Tables

# Glossary

**ACF** autocorrelation function.

**AI** Artificial Intelligence.

**AIC** Akaike's Information Criterion.

**ALS** alternating least squares.

**API** application programming interface.

**ARIMA** Autoregressive Integrated Moving Average.

**ARMA** Autoregressive Moving Average.

**BIC** Bayesian Information Criterion.

**CART** Classification and Regression Tree.

**CH4** methane.

**CHP** combined heat and power plant.

**CNN** Convolutional Neural Network.

**CO2** carbon dioxide.

**Deutsche Energie-Agentur GmbH** German Energy Agency, federally owned company providing services to design and implement the goals the energy transition.

**DFT** Discrete Fourier Transform.

**DL** Deep Learning.

**DNN** Deep Neural Network.

**DSM** Demand Side Management.

**DST** Daylight Saving Time.

**EMD** Empirical Mode Decomposition.

**EPEX** European Power Exchange.

**FCN** Fully Connected Network.

**FFT** Fast Fourier Transform.

**ICA** Independent Component Analysis.

**IEA** International Energy Agency.

**IMF** Intrinsic Mode Function.

**IPCC** Intergovernmental Panel on Climate Change.

**Irish CER** Irish Commission for Energy Regulation.

**ISSDA** Irish Social Science Data Archive.

**KL** Kullback-Leibler.

**KL divergence** Kullback-Leibler divergence.

**kNN** k-Nearest-Neighbours.

**LF** Load Forecasting.

**LSTM** Long Short Term Memory.

**LSTNet** Long- and Short-term Time-series network.

**MAE** Mean Absolute Error.

**MAPE** Mean Absolute Percentage Error.

**ML** Machine Learning.

**MLP** Multilayer Perceptron.

**MSE** Mean Squared Error.

**MSR** measuring, control and regulation unit.

**N-BEATS** Neural Basis Expansion Analysis for Interpretable Time Series Forecasting.

**NLP** Natural Language Processing.

**NMF** Nonnegative Matrix Factorization.

**NN** Neural Network.

**NNDSVD** Non-negative Double Singular Value Decomposition.

**PACF** partial autocorrelation function.

**PCA** Principal Component Analysis.

**RBM** Restricted Boltzmann Machine.

**RF** Random Forest.

**RMSE** Root Mean Squared Error.

**RNN** Recurrent Neural Network.

**SLP** Standard Load Profile.

**SOD** Sum Of Differences.

**SOTA** state-of-the-art.

**STLF** Short Term Load Forecasting.

**SVD** Singular Value Decomposition.

**SVM** Support Vector Machine.

**SVR** Support Vector Regression.

**TCN** Temporal Convolution Network.

**TSC** Time Series Classification.

**TSF** Time Series Forecasting.

**Umweltbundesamt** German Federal Environment Agency.

**UTC** Coordinated Universal Time.

**VAR** Vector Autoregression.

**VPP** virtual power plant.

**VQ** Vector Quantization.

**XGB** XGBoost.

# Bibliography


[1] Masson-Delmotte and others (eds.) *Summary for Policymakers*. Tech. rep. Geneva, Switzerland: IPCC, 2018.

[2] E. Council. *Climate change: what the EU is doing*. URL: https://www.consilium.europa.eu/en/policies/climate-change/ (visited on 02/19/2021).

[3] Umweltbundesamt. *Energiebedingte Emissionen*. Tech. rep. Umweltbundesamt, 2019. URL: https://www.umweltbundesamt.de/daten/energie/energiebedingte-emissionen/ (visited on 07/30/2019).

[4] Umweltbundesamt. *Übersicht zur Entwicklung der energiebedingten Emissionen und Brennstoffeinsätze in Deutschland 1990 - 2017*. Tech. rep. Dessau-Roßlau: Umweltbundesamt, 2019.

[5] P. Icha. *Entwicklung der spezifischen Kohlendioxid-Emissionen des deutschen Strommix in den Jahren 1990 – 2017*. Tech. rep. Dessau-Roßlau: Umweltbundesamt, 2019.

[6] E. Council. *Euorepean Council Meeting 10-11 December 2020 Main results*. URL: https://www.consilium.europa.eu/en/meetings/european-council/2020/12/10-11/ (visited on 02/18/2021).

[7] O. Edenhofer, R. Pichs-Madruga, Y. Sokona, K. Seyboth, P. Matschoss, S. Kadner, T. Zwickel, P. Eickemeier, G. Hansen, S. Schlömer, et al. *IPCC Special Report on Renewable Energy Sources and Climate Change Mitigation*. Tech. rep. IPCC, 2011.

[8] T. Klaus, C. Vollmer, K. Werner, H. Lehmann, and K. Müschen. *Energy target 2050: 100 % renewable electricity supply*. Tech. rep. Dessau-Roßlau: Umweltbundesamt, 2020.

[9] *dena-Netzstudie II Integration erneuerbarer Energien in die deutsche Stromversorgung im Zeitraum 2015 2020 mit Ausblick 2025*. Tech. rep. Berlin: Deutsche Energieagentur GmbH, 2010.

[10] S. D. Ramchurn, P. Vytelingum, A. Rogers, and N. R. Jennings. "Putting the'smarts' into the smart grid: a grand challenge for artificial intelligence". In: *Communications of the ACM* 55.4 (2012), pp. 86–97.

[11] J. P. Lopes, N. Hatziargyriou, J. Mutale, P. Djapic, and N. Jenkins. "Integrating distributed generation into electric power systems: A review of drivers, challenges and opportunities". In: *Electric power systems research* 77.9 (2007), pp. 1189–1203.

[12] I. I. Pyc and S. E. Sector. *VDE-Studie: Erneuerbare Energie braucht flexible Kraftwerke - Szenarien bis 2020*. Tech. rep. Frankfurt: Energietechnische Gesellschaft im VDE (ETG), 2013.

[13] B. für Wirtschaft und Energie. *Stromnetze*. 2019. URL: https://www.bmwi.de/Redaktion/DE/Artikel/Energie/Energieforschung/energieforschung-netze.html (visited on 07/18/2019).

[14] C. Hinrichs. "Selbstorganisierte Einsatzplanung dezentraler Akteure im smart grid". PhD thesis. Carl von Ossietzky Universität Oldenburg, 2014.

[15] S. Awerbuch and A. Preston. *The Virtual Utility: Accounting, technology & competitive aspects of the emerging industry*. 1st ed. Vol. 26. New York, NY, USA: Springer Science & Business Media, 1997. ISBN: 978-0-7923-9902-5. DOI: 10.1007/978-1-4615-6167-5.

[16] A. Nieße, S. Beer, J. Bremer, C. Hinrichs, O. Lünsdorf, and M. Sonnenschein. "Conjoint dynamic aggregation and scheduling methods for dynamic virtual power plants". In: *2014 federated conference on computer science and information systems*. Vol. 2. IEEE. 2014, pp. 1505–1514.

[17] M. Tröschel. "Aktive Einsatzplanung in holonischen Virtuellen Kraftwerken". PhD thesis. Carl von Ossietzky Universität Oldenburg, 2010.

[18] International Energy Agency. *Distributed Generation in Liberalised Electricity Markets*. Tech. rep. Paris, France: International Energy Agency, 2002.

[19] M. Sterner and I. Stadler. *Energiespeicher - Bedarf, Technologien, Integration*. 2nd ed. Berlin Heidelberg: Springer Vieweg, 2017. ISBN: 978-3-662-48892-8. DOI: 10.1007/978-3-662-48893-5.

[20] N. P. Padhy. "Unit commitment-a bibliographical survey". In: *IEEE Transactions on power systems* 19.2 (2004), pp. 1196–1205. DOI: 10.1109/TPWRS.2003.821611.

[21] D. Pisinger. "Linear time algorithms for knapsack problems with bounded weights". In: *Journal of Algorithms* 33.1 (1999), pp. 1–14. DOI: 10.1006/jagm.1999.1034.

[22] G. Anders, J.-P. Steghöfer, F. Siefert, and W. Reif. "A trust-and cooperation-based solution of a dynamic resource allocation problem". In: *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE. 2013, pp. 1–10.

[23] M. Sonnenschein, M. Tröschel, and O. Lünsdorf. "Smart Grids for Optimised Utilisation of Renewable Energy Supply". In: *Environmental Informatics and Renewable Energies - 27th International Conference on Informatics for Environmental Protection*. Vol. 137. 1. 2013, pp. 178–187.

[24] J. Bremer. "Constraint-Handling mit Supportvektor-Dekodern in der verteilten Optimierung". PhD thesis. Carl von Ossietzky Universität Oldenburg, 2015.

[25] A. Ohsenbrügge. "Dynamische Regel-und Reserveleistungsvorhaltung in zukünftigen Smart Grids". PhD thesis. Carl von Ossietzky Universität Oldenburg, 2015.

[26] M. Scott. *Delphi: A history of the center of the ancient world*. Princeton University Press, 2014.

[27] S. Makridakis, R. J. Hyndman, and F. Petropoulos. "Forecasting in social settings: The state of the art". In: *International Journal of Forecasting* 36.1 (2020), pp. 15–28.

[28] 021 E.ON Energie Deutschland GmbH. *Strombörse: So funktioniert der Strommarkt in Deutschland*. URL: https://www.eon.de/de/gk/energiewissen/stromboerse.html (visited on 03/04/2021).

[29] K. Amasyali and N. M. El-Gohary. "A review of data-driven building energy consumption prediction studies". In: *Renewable and Sustainable Energy Reviews* 81 (2018), pp. 1192–1205.

[30] T. Hong. "Short Term Electric Load Forecasting". PhD thesis. North Carolina State University, 2010.

[31] S. Haben, G. Giasemidis, F. Ziel, and S. Arora. "Short term load forecasting and the effect of temperature at the low voltage level". In: *International Journal of Forecasting* 35.4 (2019), pp. 1469–1484.

[32] D. Bugden and R. Stedman. "A synthetic view of acceptance and engagement with smart meters in the United States". In: *Energy Research & Social Science* 47 (2019), pp. 137–145.

[33] M. Alrizq and E. de Doncker. "A novel fuzzy based human behavior model for residential electricity consumption forecasting". In: *2018 IEEE Power and Energy Conference at Illinois (PECI)*. IEEE. 2018, pp. 1–7.

[34] M. Jacob, C. Neves, and D. Vukadinović Greetham. *Forecasting and Assessing Risk of Individual Electricity Peaks*. Springer Nature, 2020. DOI: https://doi.org/10.1007/978-3-030-28669-9.

[35] B. Hayes, J. Gruber, and M. Prodanovic. "Short-term load forecasting at the local level using smart meter data". In: *2015 IEEE Eindhoven PowerTech*. IEEE. 2015, pp. 1–6.

[36] T. Zufferey, A. Ulbig, S. Koch, and G. Hug. "Forecasting of smart meter time series based on neural networks". In: *International workshop on data analytics for renewable energy integration*. Springer. 2016, pp. 10–21.

[37] S. B. Taieb, J. W. Taylor, and R. J. Hyndman. "Hierarchical probabilistic forecasting of electricity demand with smart meter data". In: *Journal of the American Statistical Association* (2020), pp. 1–17.

[38] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. 2nd ed. OTexts: Melbourne, Australia, 2018. URL: http://OTexts.com/fpp2 (visited on 03/10/2021).

[39] Wikipedia, the free encyclopedia. *Natural language processing*. URL: https://en.wikipedia.org/wiki/Natural_language_processing (visited on 03/03/2021).

[40] Y. Goldberg. "A primer on neural network models for natural language processing". In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 345–420.

[41] D. Bahdanau, K. Cho, and Y. Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[42] A. Karpathy, J. Johnson, and L. Fei-Fei. "Visualizing and understanding recurrent networks". In: *arXiv preprint arXiv:1506.02078* (2015).

[43] C. Deb, F. Zhang, J. Yang, S. E. Lee, and K. W. Shah. "A review on time series forecasting techniques for building energy consumption". In: *Renewable and Sustainable Energy Reviews* 74 (2017), pp. 902–924.

[44] H. Wang, Z. Lei, X. Zhang, B. Zhou, and J. Peng. "A review of deep learning for renewable energy forecasting". In: *Energy Conversion and Management* 198 (2019), p. 111799.

[45] S. Smyl, J. Ranganathan, and A. Pasqua. *M4 forecasting competition: Introducing a new hybrid ES-RNN mode*. URL: `https://eng.uber.com/m4-forecasting-competition/` (visited on 02/18/2021).

[46] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. "The M4 Competition: Results, findings, conclusion and way forward". In: *International Journal of Forecasting* 34.4 (2018), pp. 802–808.

[47] R. G. Brown. *Statistical forecasting for inventory control*. McGraw/Hill, 1959.

[48] C. C. Holt. "Forecasting seasonals and trends by exponentially weighted moving averages". In: *International journal of forecasting* 20.1 (2004), pp. 5–10.

[49] P. R. Winters. "Forecasting sales by exponentially weighted moving averages". In: *Management science* 6.3 (1960), pp. 324–342.

[50] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[51] S.-Y. Shih, F.-K. Sun, and H.-y. Lee. "Temporal pattern attention for multivariate time series forecasting". In: *Machine Learning* 108.8 (2019), pp. 1421–1441.

[52] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL: `http://www.deeplearningbook.org`.

[53] E. Alpaydin. *Maschinelles Lernen*. Walter de Gruyter GmbH & Co KG, 2019.

[54] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[55] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[56] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[57] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[58] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[59] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[60] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *IEEE Signal processing magazine* 29.6 (2012), pp. 82–97.

[61] S. Jean, K. Cho, R. Memisevic, and Y. Bengio. "On using very large target vocabulary for neural machine translation". In: *arXiv preprint arXiv:1412.2007* (2014).

[62] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to sequence learning with neural networks". In: *arXiv preprint arXiv:1409.3215* (2014).

[63] H. S. Hippert, C. E. Pedreira, and R. C. Souza. "Neural networks for short-term load forecasting: A review and evaluation". In: *IEEE Transactions on power systems* 16.1 (2001), pp. 44–55.

[64] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[65] 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.

[66] S. Bai, J. Z. Kolter, and V. Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv preprint arXiv:1803.01271* (2018).

[67] S. Aryal, D. Nadarajah, P. L. Rupasinghe, C. Jayawardena, and D. Kasthurirathna. "Comparative Analysis of Deep Learning Models for Multi-Step Prediction of Financial Time Series". In: *Journal of Computer Science* 16.10 (2020), pp. 1401–1416.

[68] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting". In: *arXiv preprint arXiv:1905.10437* (2019).

[69] G. Lai, W.-C. Chang, Y. Yang, and H. Liu. "Modeling long-and short-term temporal patterns with deep neural networks". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 2018, pp. 95–104.

[70] J. L. Viegas, S. M. Vieira, J. M. Sousa, R. Melicio, and V. Mendes. "Electricity demand profile prediction based on household characteristics". In: *2015 12th International Conference on the European Energy Market (EEM)*. IEEE. 2015, pp. 1–5.

[71] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang. "Short-term residential load forecasting based on LSTM recurrent neural network". In: *IEEE Transactions on Smart Grid* 10.1 (2017), pp. 841–851.

[72] H. Zheng, J. Yuan, and L. Chen. "Short-term load forecasting using EMD-LSTM neural networks with a Xgboost algorithm for feature importance evaluation". In: *Energies* 10.8 (2017), p. 1168.

[73] S. Bouktif, A. Fiaz, A. Ouni, and M. A. Serhani. "Optimal deep learning lstm model for electric load forecasting using feature selection and genetic algorithm: Comparison with machine learning approaches". In: *Energies* 11.7 (2018), p. 1636.

[74] F. H. Al-Qahtani and S. F. Crone. "Multivariate k-nearest neighbour regression for time series data—A novel algorithm for forecasting UK electricity demand". In: *The 2013 international joint conference on neural networks (IJCNN)*. IEEE. 2013, pp. 1–8.

[75] C. Rudin, D. Waltz, R. N. Anderson, A. Boulanger, A. Salleb-Aouissi, M. Chow, H. Dutta, P. N. Gross, B. Huang, S. Ierome, et al. "Machine learning for the New York City power grid". In: *IEEE transactions on pattern analysis and machine intelligence* 34.2 (2011), pp. 328–345.

[76] L. Ekonomou. "Greek long-term energy consumption prediction using artificial neural networks". In: *Energy* 35.2 (2010), pp. 512–517.

[77] H. Shi, M. Xu, and R. Li. "Deep learning for household load forecasting—A novel pooling deep RNN". In: *IEEE Transactions on Smart Grid* 9.5 (2017), pp. 5271–5280.

[78] C. Lang, F. Steinborn, O. Steffens, and E. W. Lang. "Applying a 1D-CNN network to electricity load forecasting". In: *International Conference on Time Series and Forecasting*. Springer. 2019, pp. 205–218.

[79] H. K. Alfares and M. Nazeeruddin. "Electric load forecasting: literature survey and classification of methods". In: *International journal of systems science* 33.1 (2002), pp. 23–34.

[80] T. Hong and S. Fan. "Probabilistic electric load forecasting: A tutorial review". In: *International Journal of Forecasting* 32.3 (2016), pp. 914–938.

[81] M. T. Hagan and S. M. Behr. "The time series approach to short term load forecasting". In: *IEEE transactions on power systems* 2.3 (1987), pp. 785–791.

[82] J. W. Taylor. "Short-term electricity demand forecasting using double seasonal exponential smoothing". In: *Journal of the Operational Research Society* 54.8 (2003), pp. 799–805.

[83] P. Mandal, T. Senjyu, N. Urasaki, and T. Funabashi. "A neural network based several-hour-ahead electric load forecasting using similar days approach". In: *International Journal of Electrical Power & Energy Systems* 28.6 (2006), pp. 367–373.

[84] R. Zhang, Y. Xu, Z. Y. Dong, W. Kong, and K. P. Wong. "A composite k-nearest neighbor model for day-ahead load forecasting with limited temperature forecasts". In: *2016 IEEE Power and Energy Society General Meeting (PESGM)*. IEEE. 2016, pp. 1–5.

[85] X. Cao, S. Dong, Z. Wu, and Y. Jing. "A data-driven hybrid optimization model for short-term residential load forecasting". In: *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. IEEE. 2015, pp. 283–287.

[86] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[87] T. Chen. "Introduction to boosted trees". In: *University of Washington Computer Science* (2014). URL: `https://web.njit.edu/~usman/courses/cs675_fall16/BoostedTree.pdf` (visited on 04/28/2021).

[88] D. Nielsen. "Tree boosting with xgboost-why does xgboost win "every" machine learning competition?" MA thesis. NTNU, 2016.

[89] J. Moon, Y. Kim, M. Son, and E. Hwang. "Hybrid short-term load forecasting scheme using random forest and multilayer perceptron". In: *Energies* 11.12 (2018), p. 3283.

[90] G. Dudek. "Short-term load forecasting using random forests". In: *Intelligent Systems' 2014*. Springer, 2015, pp. 821–828.

[91] P.-H. Kuo and C.-J. Huang. "A high precision artificial neural networks model for short-term energy load forecasting". In: *Energies* 11.1 (2018), p. 213.

[92] A. Lahouar and J. B. H. Slama. "Day-ahead load forecast using random forest and expert input selection". In: *Energy Conversion and Management* 103 (2015), pp. 1040–1051.

[93] J. Walther, D. Spanier, N. Panten, and E. Abele. "Very short-term load forecasting on factory level–A machine learning approach". In: *Procedia CIRP* 80 (2019), pp. 705–710.

[94] B.-J. Chen, M.-W. Chang, et al. "Load forecasting using support vector machines: A study on EUNITE competition 2001". In: *IEEE transactions on power systems* 19.4 (2004), pp. 1821–1830.

[95] S. Humeau, T. K. Wijaya, M. Vasirani, and K. Aberer. "Electricity load forecasting for residential customers: Exploiting aggregation and correlation between households". In: *2013 Sustainable internet and ICT for sustainability (SustainIT)*. IEEE. 2013, pp. 1–6.

[96] P. Vrablecová, A. B. Ezzeddine, V. Rozinajová, S. Šárik, and A. K. Sangaiah. "Smart grid load forecasting using online support vector regression". In: *Computers & Electrical Engineering* 65 (2018), pp. 102–117.

[97] E. Ceperic, V. Ceperic, and A. Baric. "A strategy for short-term load forecasting by support vector regression machines". In: *IEEE Transactions on Power Systems* 28.4 (2013), pp. 4356–4364.

[98] E. E. Elattar, J. Goulermas, and Q. H. Wu. "Electric load forecasting based on locally weighted support vector regression". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.4 (2010), pp. 438–447.

[99] L. Hernandez, C. Baladron, J. M. Aguiar, B. Carro, A. J. Sanchez-Esguevillas, J. Lloret, and J. Massana. "A survey on electric power demand forecasting: future trends in smart grids, microgrids and smart buildings". In: *IEEE Communications Surveys & Tutorials* 16.3 (2014), pp. 1460–1495.

[100] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[101] K. Gajowniczek and T. Ząbkowski. "Short term electricity forecasting using individual smart meter data". In: *Procedia Computer Science* 35 (2014), pp. 589–597.

[102] Y. Bengio, A. Courville, and P. Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[103] E. Mocanu, P. H. Nguyen, M. Gibescu, E. M. Larsen, and P. Pinson. "Demand forecasting at low aggregation levels using factored conditional restricted boltzmann machine". In: *2016 Power Systems Computation Conference (PSCC)*. IEEE. 2016, pp. 1–7.

[104] D. L. Marino, K. Amarasinghe, and M. Manic. "Building energy load forecasting using deep neural networks". In: *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2016, pp. 7046–7051.

[105] S. Ryu, J. Noh, and H. Kim. "Deep neural network based demand side short term load forecasting". In: *Energies* 10.1 (2017), p. 3.

[106] N. Al Khafaf, M. Jalili, and P. Sokolowski. "Application of deep learning long short-term memory in energy demand forecasting". In: *International Conference on Engineering Applications of Neural Networks*. Springer. 2019, pp. 31–42.

[107] Z. C. Lipton, J. Berkowitz, and C. Elkan. "A critical review of recurrent neural networks for sequence learning". In: *arXiv preprint arXiv:1506.00019* (2015).

[108] K. Chen, K. Chen, Q. Wang, Z. He, J. Hu, and J. He. "Short-term load forecasting with deep residual networks". In: *IEEE Transactions on Smart Grid* 10.4 (2018), pp. 3943–3952.

[109] B. N. Oreshkin, G. Dudek, and P. Pełka. "N-BEATS neural network for mid-term electricity load forecasting". In: *arXiv preprint arXiv:2009.11961* (2020).

[110] S. N. Fallah, R. C. Deo, M. Shojafar, M. Conti, and S. Shamshirband. "Computational intelligence approaches for energy load forecasting in smart energy management grids: state of the art, future challenges, and research directions". In: *Energies* 11.3 (2018), p. 596.

[111] X. Qiu, Y. Ren, P. N. Suganthan, and G. A. Amaratunga. "Empirical mode decomposition based ensemble deep learning for load demand time series forecasting". In: *Applied Soft Computing* 54 (2017), pp. 246–255.

[112] J.-Y. Kim and S.-B. Cho. "Electric energy consumption prediction by deep learning with state explainable autoencoder". In: *Energies* 12.4 (2019), p. 739.

[113] K. Hopf, M. Sodenkamp, I. Kozlovkiy, and T. Staake. "Feature extraction and filtering for household classification based on smart electricity meter data". In: *Computer Science-Research and Development* 31.3 (2016), pp. 141–148.

[114] Z. Wei, X. Li, K. W. Cheung, J. Wu, and S. Huang. "A new short-term load forecasting model based on relevance vector machine". In: *22nd International Conference on Electricity Distribution*. IET, 2013.

[115] P. Paatero and U. Tapper. "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values". In: *Environmetrics* 5.2 (1994), pp. 111–126.

[116] D. D. Lee and H. S. Seung. "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755 (1999), pp. 788–791.

[117] I. Buciu and I. Pitas. "Application of non-negative and local non negative matrix factorization to facial expression recognition". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 1. IEEE. 2004, pp. 288–291.

[118] Z. Tang, X. Zhang, and S. Zhang. "Robust perceptual image hashing based on ring partition and NMF". In: *IEEE transactions on knowledge and data engineering* 26.3 (2013), pp. 711–724.

[119] F. Shahnaz, M. W. Berry, V. P. Pauca, and R. J. Plemmons. "Document clustering using nonnegative matrix factorization". In: *Information Processing & Management* 42.2 (2006), pp. 373–386.

[120] V. P. Pauca, F. Shahnaz, M. W. Berry, and R. J. Plemmons. "Text mining using non-negative matrix factorizations". In: *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM. 2004, pp. 452–456.

[121] R. Schachtner, D. Lutter, P. Knollmüller, A. M. Tomé, F. J. Theis, G. Schmitz, M. Stetter, P. G. Vilda, and E. W. Lang. "Knowledge-based gene expression classification via matrix factorization". In: *Bioinformatics* 24.15 (2008), pp. 1688–1697.

[122] K. Devarajan. "Nonnegative matrix factorization: an analytical and interpretive tool in computational biology". In: *PLoS Comput Biol* 4.7 (2008), e1000029.

[123] R. Schachtner, G. Poppel, and E. W. Lang. "A nonnegative blind source separation model for binary test data". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 57.7 (2010), pp. 1439–1448.

[124] P. Weiderer, A. M. Tomé, and E. W. Lang. "A NMF-based extraction of physically meaningful components from sensory data of metal casting processes". In: *Journal of Manufacturing Systems* 54 (2020), pp. 62–73.

[125] H.-W. Tseng, M. Hong, and Z.-Q. Luo. "Combining sparse NMF with deep neural network: A new classification-based approach for speech enhancement". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 2145–2149.

[126] A. Rahimpour, H. Qi, D. Fugate, and T. Kuruganti. "Non-intrusive energy disaggregation using non-negative matrix factorization with sum-to-k constraint". In: *IEEE Transactions on Power Systems* 32.6 (2017), pp. 4430–4441.

[127] A. K. Zarabie, S. Das, and H. Wu. "A Data-Driven Machine Learning Approach for Consumer Modeling with Load Disaggregation". In: *arXiv preprint arXiv:2011.03519* (2020).

[128] S.-J. Kim and G. B. Giannakis. "Load forecasting via low rank plus sparse matrix factorization". In: *2013 Asilomar Conference on Signals, Systems and Computers*. IEEE. 2013, pp. 1682–1686.

[129] J. Mei, Y. De Castro, Y. Goude, J.-M. Azais, and G. Hébrail. "Nonnegative matrix factorization with side information for time series recovery and prediction". In: *IEEE Transactions on Knowledge and Data Engineering* 31.3 (2018), pp. 493–506.

[130] X. Shao, C.-S. Kim, and P. Sontakke. "Accurate deep model for electricity consumption forecasting using multi-channel and multi-scale feature fusion CNN–LSTM". In: *Energies* 13.8 (2020), p. 1881.

[131] M. V. Shcherbakov, A. Brebels, N. L. Shcherbakova, A. P. Tyukov, T. A. Janovsky, V. A. Kamaev, et al. "A survey of forecast error measures". In: *World Applied Sciences Journal* 24.24 (2013), pp. 171–176.

[132] Commission for Energy Regulation (CER). *CER Smart Metering Project - Electricity Customer Behaviour Trial, 2009-2010 [dataset]. 1st Edition*. Irish Social Science Data Archive. SN: 0012-00. 2012. URL: `https://www.ucd.ie/issda/data/commissionforenergyregulationcer/`.

[133] R. H. Shumway and D. S. Stoffer. *Time series analysis and its applications*. Vol. 4. Springer, 2017.

[134] P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting*. Springer, 2016.

[135] S. L. Brunton and J. N. Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

[136] numpy documentation. *Descrete Fourier Transform*. URL: https://numpy.org/doc/stable/reference/routines.fft.html#background-information (visited on 05/21/2021).

[137] J. W. Cooley and J. W. Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

[138] I. T. Jolliffe. "Principal components in regression analysis". In: *Principal component analysis*. Springer, 1986, pp. 129–155.

[139] A. Gersho and R. M. Gray. *Vector quantization and signal compression*. Vol. 159. Springer Science & Business Media, 2012.

[140] P. Foldiak. "Sparse coding in the primate cortex". In: *The handbook of brain theory and neural networks* (2003).

[141] D. D. Lee and H. S. Seung. "Algorithms for non-negative matrix factorization". In: *Advances in neural information processing systems*. 2001, pp. 556–562.

[142] P. O. Hoyer. "Non-negative matrix factorization with sparseness constraints." In: *Journal of machine learning research* 5.9 (2004).

[143] T. Serre. "Hierarchical Models of the Visual System." In: *Encyclopedia of computational neuroscience* 6 (2014), pp. 1–12.

[144] H. Abdi and L. J. Williams. "Principal component analysis". In: *Wiley interdisciplinary reviews: computational statistics* 2.4 (2010), pp. 433–459.

[145] Wikipedia, the free encyclopedia. *Explained variation*. URL: https://en.wikipedia.org/wiki/Explained_variation (visited on 03/04/2021).

[146] scikit-learn documentation. *Explained variance score*. URL: https://scikit-learn.org/0.24/modules/model_evaluation.html#explained-variance-score (visited on 03/04/2021).

[147] scikit-learn documentation. *Non-negative matrix factorization (NMF or NNMF)*. URL: https://scikit-learn.org/0.24/modules/decomposition.html#nmf (visited on 03/04/2021).

[148] C. Févotte and J. Idier. "Algorithms for nonnegative matrix factorization with the $\beta$-divergence". In: *Neural computation* 23.9 (2011), pp. 2421–2456.

[149] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009.

[150] G. Zhou, A. Cichocki, Q. Zhao, and S. Xie. "Nonnegative matrix and tensor factorizations: An algorithmic perspective". In: *IEEE Signal Processing Magazine* 31.3 (2014), pp. 54–65.

[151] C. Boutsidis and E. Gallopoulos. "SVD based initialization: A head start for nonnegative matrix factorization". In: *Pattern recognition* 41.4 (2008), pp. 1350–1362.

[152] J. Le Roux, J. R. Hershey, and F. Weninger. "Deep NMF for speech separation". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 66–70.

[153] H. Laurberg. "Uniqueness of non-negative matrix factorization". In: *2007 IEEE/SP 14th Workshop on Statistical Signal Processing*. IEEE. 2007, pp. 44–48.

[154] H. Laurberg, M. G. Christensen, M. D. Plumbley, L. K. Hansen, and S. H. Jensen. "Theorems on positive data: On the uniqueness of NMF". In: *Computational intelligence and neuroscience* 2008 (2008).

[155] C. Eckart and G. Young. "The approximation of one matrix by another of lower rank". In: *Psychometrika* 1.3 (1936), pp. 211–218.

[156] B. Huppert. *Angewandte Lineare Algebra*. Walter de Gruyter, 2013.

[157] scikit-learn documentation. *Clustering*. URL: `https://scikit-learn.org/0.24/modules/clustering.html#` (visited on 04/29/2021).

[158] scikit-learn documentation. *Linear Models*. URL: `https://scikit-learn.org/0.24/modules/linear_model.html` (visited on 04/29/2021).

[159] scikit-learn documentation. *Decision Trees*. URL: `https://scikit-learn.org/0.24/modules/tree.html#` (visited on 04/23/2021).

[160] R. Couronné, P. Probst, and A.-L. Boulesteix. "Random forest versus logistic regression: a large-scale benchmark experiment". In: *BMC bioinformatics* 19.1 (2018), pp. 1–14.

[161] L. Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[162] L. Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.

[163] Y. Freund and R. E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[164] J. H. Friedman. "Stochastic gradient boosting". In: *Computational statistics & data analysis* 38.4 (2002), pp. 367–378.

[165] A. Liaw, M. Wiener, et al. "Classification and regression by randomForest". In: *R news* 2.3 (2002), pp. 18–22.

[166] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. 2nd ed. Springer, 2009. URL: `https://web.stanford.edu/~hastie/ElemStatLearn/` (visited on 07/01/2021).

[167] R. Kruse, C. Borgelt, C. Braune, F. Klawonn, C. Moewes, and M. Steinbrecher. *Computational Intelligence - Eine Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. Springer, 2015.

[168] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[169] T. Tieleman and G. Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[170] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[171] C. for Energy Regulation (CER). *Data from the Commission for Energy Regulation*. URL: https://www.ucd.ie/issda/data/commissionforenergyregulationcer/ (visited on 11/10/2020).

[172] Jon Skeet. *Storing UTC is not a silver bullet*. 2019. URL: https://codeblog.jonskeet.uk/2019/03/27/storing-utc-is-not-a-silver-bullet/ (visited on 03/24/2021).

[173] H. Chen, Y. Du, and J. N. Jiang. "Weather sensitive short-term load forecasting using knowledge-based ARX models". In: *IEEE Power Engineering Society General Meeting, 2005*. IEEE. 2005, pp. 190–196.

[174] Met Éireann. *Met Éireann - The Irish Meteorological Service - Historical Data*. URL: https://www.met.ie//climate/available-data/historical-data (visited on 05/27/2020).

[175] Maurizio Montel. *Python Package: Holidays*. https://github.com/dr-prodigy/python-holidays. Used version: 0.10.2.

[176] Wikipedia, the free encyclopedia. *Public holidays in the Republic of Ireland*. URL: https://en.wikipedia.org/wiki/Public_holidays_in_the_Republic_of_Ireland (visited on 01/25/2021).

[177] P. Remy. *N-BEATS: Neural basis expansion analysis for interpretable time series forecasting*. https://github.com/philipperemy/n-beats. 2020.

[178] D. E. Golberg. "Genetic algorithms in search, optimization, and machine learning". In: *Addion wesley* 1989.102 (1989), p. 36.

[179] E. Caro, J. Juan, M. Maña, J. Rupérez, C. Rodríguez, A. Rodríguez, and J. J. Abellán. "The Effect of Daylight Saving Time on Spanish Electrical Consumption". In: *International Conference on Time Series and Forecasting*. Springer. 2019, pp. 177–190.

[180] Simon Kennedy. *Python Package: Astral*. https://astral.readthedocs.io/en/stable/index.html. Used version: 2.2.

[181] Ordnance Survey Ireland. *Irish Ordnance Survey FAQ (archived)*. URL: https://web.archive.org/web/20120228155031/http://www.osi.ie/en/faq/faq3.aspx (visited on 01/26/2021).

# Acknowledgments

At this point, I would like to thank some people, without whom this thesis would not have turned out the way it did and who made the last few years a really special time.

First of all, I would like to express many thanks to Elmar Lang. Thank you for giving me the opportunity to pursue my doctorate. Thanks for the feedback, the always relaxed atmosphere in our meetings and coffee breaks and thank you for continuously supporting my project.

Many thanks also go out to Marlene. Thank you for having my back, for the moral support and for building me up during difficult times. You always know how to keep me motivated and active. Thank you for the critical questioning of my writing and the elaborate corrections. Overall, thanks for a great time together. I'm glad to have you by my side.

I also want to thank my parents for their continuous and unconditional support and love since my childhood. No matter what ideas - like pursuing a doctorate - I come up with, I can always be sure that you have by back and provide me with honest and valuable feedback.

A large thank you also goes out to Marinus and Christian for the many good talks, and fun times - whether in the office or outside. Thanks for many interesting cookie-supported discussions on technical topics and everything else. Besides, the world is simply a better place with colored chalk on your hands. Also thank you Simon for putting in so much work to get our GPUs up and running and enabling remote access for our group.

Additionally, I would like to thank Haley for putting my English writing to another level and helping me to place about a million commas.

Lastly, I want to thank all my friends who made the past few years a truly wonderful experience I will never forget. Thank you Lehmy, Andi, Thea, Carmen, Alex, Jo, Aaron, Nele, Beda, Stric, Pierre, Paula, Nadine, Fritzi, Schorsch, Seba, Jonas, Björn, Wacki, Chrissy, Lena, Sergej, Toni, Sabs, and so many more.

I would also like to thank the CER Smart Metering Project - Electricity Customer Behaviour Trial, 2009-2010 for making the data, this work is based on, available. The data was accessed via the Irish Social Science Data Archive - `www.ucd.ie/issda`.