Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Reliable Data Forecasting for Building Automation Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Jürgen Pannosch, BSc
Matrikelnummer 00725044

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dr. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Daniel Schachinger

Wien, 7. März 2018

Jürgen Pannosch

Wolfgang Kastner

Reliable Data Forecasting for Building Automation Systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Jürgen Pannosch, BSc
Registration Number 00725044

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof.Dr. Wolfgang Kastner
Assistance: Dipl.-Ing. Daniel Schachinger

Vienna, 7th March, 2018

Jürgen Pannosch Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Jürgen Pannosch, BSc
Scheunenweg 9, 2301 Gross-Enzersdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. März 2018

Jürgen Pannosch

Acknowledgements

Special thanks to my adviser Wolfgang Kastner that he gave me the opportunity to write my thesis at the Automation Systems Group and to Daniel Schachinger for his patience and support while writing this thesis.

Further, I want to thank my parents for the never-ending support in my life and the encouragement to keep on doing my studies. Also a big thanks for the support to my family, who had always an open ear for me.

I want to thank everyone who has supported me along my way at TU Wien, especially my closest colleagues: Mathias Burgholzer, Thomas Mayer, Thomas Preindl, Stefan Seifried and Djordje Slijepcevic.

Abstract

Over the years, more and more buildings got equipped with a building automation system (BAS) which provides the ability to automate building services, like heating, cooling, ventilation, air conditioning, shading, lighting, alarming, safety and security systems. Furthermore, an additional aim of a BAS is to optimize such services automatically. Such optimizations can have different purposes, e.g., to make the building more energy efficient, raise the comfort of the users, or just to make them more maintainable. For most optimization tasks, it is essential to know the future behavior of a building. Forecasts of different sensors in the building can generate this knowledge. Therefore, this thesis evaluated different algorithms to forecast such sensor data. The algorithms under investigation are autoregressive integrated moving average (ARIMA), artificial neural network (ANN) and support vector machine (SVM). The evaluation was divided into five categories related to different types of sensors in a building. These types of sensors were electricity, district heating, humidity, temperature, and photovoltaic production. Furthermore, the algorithms were evaluated at different seasons to ensure the accuracy of the algorithms throughout the year. This approach ended in a comprehensive performance evaluation where the ANN algorithm was superior to the others. Finally, the ANN algorithm with its elaborated network structure was implemented in a JAVA library for further development and reuse.

Kurzfassung

Im Laufe der Jahre wurden immer mehr Gebäude mit einem Gebäudeautomationssystem ausgestattet. Dieses automatisiert verschiedenste Gebäudedienste, wie Heizung, Kühlung, Lüftung, Klimatisierung, Beschattung, Beleuchtung, Alarmierung und Sicherheitssysteme. Darüber hinaus ist ein weiteres Ziel des Gebäudeautomationssystems, solche Dienste automatisch zu optimieren. Optimierungen können verschiedene Zwecke haben, z.B. um das Gebäude energieeffizienter zu machen, den Komfort der Nutzer zu erhöhen oder einfach nur um es wartungsfreundlicher zu machen. Für die meisten Optimierungsaufgaben ist es unerlässlich, das zukünftige Verhalten eines Gebäudes zu kennen. Prognosen verschiedener Sensoren im Gebäude können dieses Wissen generieren. Daher hat diese Arbeit verschiedene Algorithmen zur Vorhersage solcher Sensordaten evaluiert. Die untersuchten Algorithmen waren Autoregressive Integrated Moving Average (ARIMA), Artificial Neural Network (ANN) und Support Vector Machine (SVM). Die Bewertung wurde in fünf Kategorien eingeteilt, die sich auf verschiedene Arten von Sensoren im Gebäude beziehen. Diese Arten von Sensoren waren Elektrizität, Fernwärme, Feuchtigkeit, Temperatur und Photovoltaik-Produktion. Darüber hinaus wurden die Algorithmen zu verschiedenen Jahreszeiten ausgewertet, um die Genauigkeit der Algorithmen über das ganze Jahr zu untersuchen. Die Auswertung endete in einer umfassenden Leistungsbewertung, bei der der ANN-Algorithmus den anderen überlegen war. Schließlich wurde der ANN-Algorithmus mit der erarbeiteten Netzwerkstruktur in einer JAVA-Bibliothek zur weiteren Entwicklung und Wiederverwendung implementiert.

Contents

Abstract **ix**

Kurzfassung **xi**

Contents **xiii**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem statement . . . 2
- 1.3 Aim of this work . . . 3
- 1.4 Methodological approach . . . 3
- 1.5 Related work . . . 4
- 1.6 Structure of this work . . . 6

2 Definitions for time series analysis **7**

3 Linear models **13**
- 3.1 Linear filter model . . . 13
- 3.2 Autoregressive model . . . 14
- 3.3 Moving average model . . . 15
- 3.4 Autoregressive moving average model . . . 17
- 3.5 Autoregressive integrated moving average model . . . 18

4 Nonlinear models **19**
- 4.1 Artificial neural network . . . 20
- 4.2 Support vector machine . . . 28

5 Data preparation and preprocessing **31**
- 5.1 Data selection and preparation . . . 31
- 5.2 Correlation analysis . . . 37
- 5.3 Feature selection . . . 43

6 Implementation and evaluation **47**
- 6.1 Performance measures . . . 47

6.2 Model architecture 49
6.3 Evaluation method 58
6.4 Results 60
6.5 Implementation 66

7 Conclusion 73
7.1 Summary 73
7.2 Future work 74

A Detailed forecast results 77

List of Figures 87

List of Tables 89

List of Acronyms 91

Bibliography 93

CHAPTER 1

Introduction

1.1 Motivation

The number of buildings equipped with automation systems increased in the past few years. Apart from the positive aspects of this development, e.g., higher user comfort or easier monitoring, there is an inherent problem with the growing amount of generated sensor data. Due to the complexity of such systems, most sensor data are not evaluated or even used, neither are the connections and dependencies analyzed and used to optimize building efficiency. However, many statistical or economic evaluations can be performed on these data, like energy performance evaluation, cost estimation, or failure analysis. This results in an unused potential, where historical data can be used for trend estimation, to predict future failures of devices, or validation of taken actions. Also, the prediction of sensor data, so-called forecasting, is an important issue in many decision-making applications or planning processes. Thereby, forecasts of sensor data can be used by an automation and management system to react to upcoming events. For example, the prior knowledge of the temperature behavior can be used to optimize the control loop of a heating or cooling system to achieve less temperature overshooting or undershooting compared to the setpoint. Furthermore, forecasts can be used to generate schedules for task execution. Prior knowledge of building process behavior is essential for demand-side management as cost-intensive tasks can be scheduled and optimized to reduce energy costs or to minimize the risk of an overload in the grid. In general, data forecasts can be used in a building automation system (BAS) to better satisfy comfort needs of the users and inhabitants while optimizing the building efficiency. However, such forecasts are very complex and not always straightforward. The complexity is a result of the different configurations of buildings. For each building, a prediction model is generated and actively maintained in order to react to changes in the process behavior. Experts are needed to generate and maintain such a model, which is expensive and error-prone. In this context, the focus of this thesis is the use of historical data to generate reliable forecasts automatically, to limit the manual effort.

1.2 Problem statement

Modern buildings, especially in the commercial sector, are more and more equipped with a BAS. Such a system provides the ability to control and monitor building services like heating, cooling, ventilation, air conditioning, shading, lighting, alarming, safety, and security systems [1]. With the growing amount of photovoltaic power, battery storage, and electrical powered transport vehicles, new tasks arise for the BAS.

One major task is named energy demand management or demand-side management. The concept of demand-side management is used by the electricity industry for years to shift the time of energy consumption [2]. There are several techniques to achieve this like, e.g., Peak Clipping, Valley Filling, or Load Shifting [2]. Possible implementations of these techniques are the *Market Demand Response* and *Physical Demand Response* [3]. *Market Demand Response* tries to control the energy consumption by variable prices while *Physical Demand Response* provides direct emergency signals. Therefore, the energy market is changing from a rigid price model to variable prices [4]. This means periods with high demand are getting more expensive than periods with less or no demand. An example for high demand is, in the afternoon when many people come home and would like to charge their electric cars at the same time. Combined with renewable energies like sun or wind, there are also situations where a lot of sun or wind power is available, but the demand is stable, and thereby the energy gets cheaper. In the same way, as the market is changing, also a BAS has to change. This means, an efficient BAS will take future events like sunshine, wind or changing prices into account. Therefore a BAS can store data, so that any command, event, or state transition of a device can be recorded in a temporary or permanent way [1]. Sensor data like energy consumption, temperature, humidity, or CO_2 can easily be stored. After a period the recorded data forms a database. Such a database can be used by experts to improve or evaluate the efficiency of a building by discovering an unwanted or unintended behavior of the BAS. Depending on the experience of the expert, decisions can be optimal or even counterproductive.

By further development, such experts could be replaced in the future by an optimization service within the BAS. Such a service tries to make an optimal schedule for every controllable device by estimating, for example, the user behavior, energy demand, weather conditions, or solar radiation. Such estimations are done by forecasts. Similar to the experts, also the optimization service uses the recorded data as a database. Depending on the length of a forecast, there is a distinction between short-term and long-term forecasts. Usually, a forecast is called short-term if the forecast period is bounded up to months and long-term for periods of years [5]. Within this thesis, only short-term forecasts are of interest, because they are suitable for operations management [5].

The focus of this thesis is to determine which algorithms are capable of producing reliable forecasts based on sensor data in a BAS. For this purpose, historical data records on energy and heat consumption, photovoltaic production, indoor and outdoor temperature, as well as indoor and outdoor humidity are used. For these sensors, a forecast of one week ahead should provide a reasonable basis for any optimization service. To work out a meaningful answer to this problem statement, several minor questions have to be answered:

- Do sensors correlate with others in the same building?
- Can sensor correlation data enhance the forecast?
- Which forecasting methods are appropriate?

1.3 Aim of this work

The primary objective of this thesis is to compare the existing approaches autoregressive integrated moving average (ARIMA), artificial neural network (ANN), and support vector machine (SVM) to develop a system for reliable short-term forecasts of energy and heat consumption, photovoltaic production, and temperature and humidity behavior for applications in the field of building automation. To get a better understanding of the relationship between different sensors and time intervals the correlation and dependencies are analyzed and the outcomes are used to improve the forecasts. Moreover, trends and seasonality are identified and considered by the forecast. Irregularities and wrong data values are corrected or interpolated. It should be possible to forecast the data up to one week with a granularity of one hour. The performance of the system is analyzed by the use of the following error metrics: mean absolute error (MAE), symmetric mean absolute percentage error (SMAPE), mean absolute scaled error (MASE). The approach with the best performance is implemented in JAVA to provide an open API for further use.

The overall aim is to provide an approach were the operator do not need to know the building characteristics or the user behavior to forecast the mentioned sensors. Such forecasts should also be used as inputs for fully automated optimization services to, e.g., reduce the energy costs or improve the comfort within the building.

1.4 Methodological approach

The methodological approach to achieve the expected result contains the following steps:

First, an extensive literature review of currently available technologies to predict sensor data will be performed. Following approaches are significant: linear regression models, statistical methods like the ARIMA model, or machine learning methods like ANNs and SVMs. In particular, the combination of methods should be intensively reviewed regarding mutual effects and benefits.

As a second step, the test data should be analyzed to find dependencies between different sensors, detect trends and seasonality. One primary task will be to determine whether the underlying statistical process is stationary or not. Moreover, it will be explored how to divide the provided data into a training and test set to prevent underfitting or overfitting of the trained model. In the case of ANN and SVM, it should be checked if a continuous learning algorithm has an advantage over a one-time training algorithm.

Following the literature review and preparation of the test data, the most promising forecast models, i.e., ARIMA, ANN, and SVM, will be implemented and tested with MATLAB [6]. An iterative approach will be used, to find the best parameters for each model. After every iteration, the MAE, SMAPE, and MASE of the model will be calculated and compared with the previous iteration. According to the outcome, the parameters will be adjusted, so that the performance of each model increases with every iteration. This process will be repeated with various partitions between the training set and test set. Therefore, the sample data set gets partitioned into a training and test set, whereas various partitions, will be evaluated. In the end, it results in extensive cross-validation.

To evaluate the individual methods, following performance measures will be used: MAE, SMAPE, and MASE. The MAE is simple to calculate, but it has a major disadvantage as it is not free of scale. This means that it is impossible to compare the performance of a temperature forecast with a power consumption forecast. For this case, there are scale-free performance measures, like SMAPE or MASE. The latter will be used to compare the performance of a selected forecast model with different time series.

After the evaluation of the MATLAB models, the best performing method will be implemented in JAVA. The purpose is to provide an open API for further applications.

1.5 Related work

In the research area of forecasting the following three methods are mainly used: ARIMA, ANN, and SVM. The ARIMA method is often used in statistics or econometrics to predict sales numbers and other interesting indicators in business life. Further research showed that this method is also useful in the field of time series analysis. The ANN and the SVM method is a machine learning approach which is widely used for pattern recognition tasks. Besides the classification tasks it can also solve regression problems. The following paragraph summarizes related work where these three algorithms are used to predict sensors in the field of building automation, like the energy consumption or the cooling capacity.

Sapankevych and Sankar [7] published a survey on SVMs in the field of time series prediction. “The underlying motivation for using SVMs is the ability of this methodology to accurately forecast time series data when the underlying system processes are typically nonlinear, non-stationary and not defined a-priori” [7, p. 25]. The main challenges are to select the right kernel function, the right parameters and the tradeoff between complexity and technical advantages. They concluded that SVMs provide a good method to predict and forecast time series data for a wide range of applications.

Lixing, Jinhu, Xuemei, *et al.* [8] discussed a one-hour ahead load forecasting approach for building cooling using a SVM. Therefore they used the ant colony algorithm to optimize three parameters of the support vector regression (SVR), namely the regularization term C, the loss function ϵ and the kernel function φ.

Son and Kim [9] used a SVM to provide a precise model to forecast the electricity demand in the residential sector. They used fuzzy-rough feature selection with particle swarm optimization to find, 10 out of 19 important variables that may influence the electricity demand. After that, they used the 10 variables as input for the SVM. The benefit of this approach is that the feature selection is unsupervised.

Xuemei, Lixing, Yuyuan, *et al.* [10] built a hybrid model to forecast the building cooling load. They used an SVM and ARIMA model to improve the accuracy. They started with building an ARIMA model. In the next step, they fed the residual error into an SVM to improve the accuracy of the ARIMA model. Basically, they tried to estimate the prediction error and used it to improve the forecast. This method resulted in an improvement over the individual algorithms. Qu, Chen, and Liu [11] also described a hybrid forecast model to predict the building cooling load. Instead of using an SVM, they used an ANN with a radial basis transfer function.

Jinhu, Xuemei, Lixing, *et al.* [12] used the principle component analysis to transform the high dimensional data to a lower dimension and fed it to a weighted SVM to predict the cooling load of a building. They concluded that the approach has a better generalization then a regular SVM.

Hunter, Yu, Pukish III, *et al.* [13] showed the difficulty in selecting the right neural network architecture, i.e., multilayer perceptron (MLP), bridged multilayer perceptron (BMLP), fully connected cascade (FCC), and selecting the proper number of neurons. Furthermore, they investigated how many patterns and the which training algorithm, i.e., error backpropagation algorithm (EBP), Levenberg-Marquardt algorithm (LM), or neuron by neuron (NBN), should be used for training.

Escrivá-Escrivá, Álvarez-Bel, Roldán-Blay, *et al.* [14] presented “an artificial neural network (ANN) method for short-term prediction of total power consumption in buildings with several independent processes” [14, p. 3112]. The used input features for the neural network were the maximum, minimum, and average temperature of the day of prediction and the average temperature of the day before.

Mandal, Senjyu, Urasaki, *et al.* [15] presented a method for short-term load forecasting using an ANN combined with a similar days approach. The forecast accuracy was evaluated in different seasons, i.e., summer, winter, spring, and autumn. They concluded that the ANN-based approach provides a reliable forecast for several-hour-ahead load forecasts.

Cui and Peng [16] presented an improved autoregressive integrated moving average with exogenous inputs (ARIMAX) model which uses the temperature data to predict the short-term city energy load. The approach was evaluated against an autoregressive (AR) model of order 1, an autoregressive moving average (ARMA) model of order (3,2), and an ANN. For the evaluation, several days in August were predicted.

1.6 Structure of this work

Chapter 2 starts with a general introduction of basic definitions for time series analysis. The definitions for the terms *time series*, *stationary*, *trend*, *seasonality*, and *correlation* are presented.

Chapter 3 introduces linear models for time series forecasting. It starts with a short historical introduction followed by presenting the AR and moving average (MA) model. The chapter ends with the ARIMA model by Box, Jenkins, Reinsel, *et al.* [17].

Chapter 4 presents non-linear models, especially ANN and SVM. It starts with defining basic terms, like *supervised* and *unsupervised* learning, followed by the sections, Neural Network Section 4.1 and Support Vector Machine Section 4.2. Section 4.1 begins with a summary and the basic definition of a perceptron followed by the feedforward-network and recurrent-network. For every network, an appropriate training algorithm is presented. Section 4.2 starts with a brief introduction followed by the definition of an SVM for a linear decision decision boundary. Furthermore, the definition is enhanced by the kernel trick to solve arbitrary problems. It also describes how to find the optimal weight vector.

Chapter 5 deals with the analysis of historical data. It describes in Section 5.1 the used datasets and defines five different types of time series. It starts with presenting ways to preprocess the historical data to handle incorrect data, trends, and seasonality. Furthermore in Section 5.2, different sensors in the same building are analyzed if there exists a correlation between them. In Section 5.3, the reader gets a deep introduction into feature extraction and generation of ANN and SVR.

After the general theory presented in the previous chapters, Chapter 6 presents the modifications and adjustments made to the state-of-the-art methods, i.e., ARIMA, ANN, and SVM to forecast sensor data. Furthermore, the performance of these methods is evaluated by the presented error metrics MAE, SMAPE, and MASE. At the end of the chapter, the best forecast method gets implemented in JAVA.

The thesis ends with a brief summary of the presented approach, in Chapter 7. It further presents possible future extensions or optimizations to the presented approach.

CHAPTER 2

Definitions for time series analysis

Time series analysis investigates the behavior of a phenomenon over time. Such a phenomenon can be, for example, the number of monthly sales, daily temperature, or hourly electricity consumption. The mathematical description of such a phenomenon is called a random process x. All random processes have a common property: They are measurable over time. This property leads us to the definition of the principal term time series.

Definition (time series)**.** A time series is a discrete ordered set of samples, indexed by time and taken from a random process [5] [17].

Equation 2.1 presents a more mathematical notation of a time series, where x denotes the time series, and x_t is a single observation at time t.

$$x = \{x_0, x_1, \cdots, x_t\}, t \in \mathbb{Z} \tag{2.1}$$

Usually, each sample taken from a random process is equally distributed over time, e.g., every minute, every 10 minutes, or every hour. If this is not the case, preprocessing of the data is required (see Chapter 5 for more details). For further analysis and simplicity, every sample is taken in an equal time interval. Figure 2.1 illustrates a time series of the outdoor temperature of a building.

Because of natural law, it is impossible to capture samples in the future. Thus, the time t of the samples is bounded up to the current time of observation. Any sample predicted beyond the time t up to time $t + l$ is called forecast [5]. The time interval from t to $t + l$ is called the forecast horizon [5] or lead time [17], where $l \in \mathbb{Z}$.

Mathematical models can be used to get a better understanding of a time series. Some models can be derived from physical laws so that the output is always deterministic.

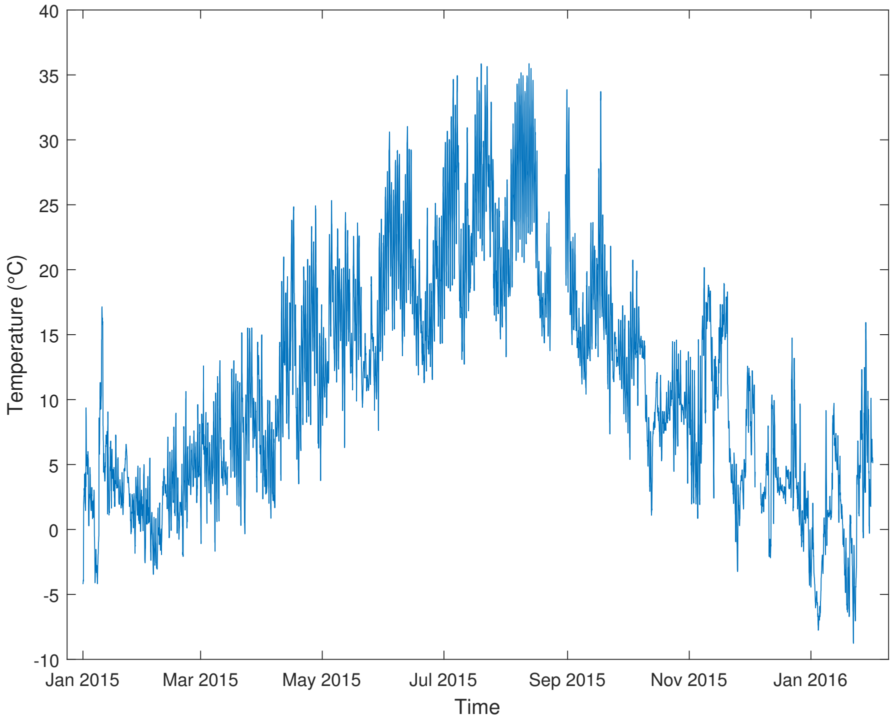


Figure 2.1: Sample time series

In many cases, it is too complicated to get a perfect deterministic model, like for the ambient temperature behavior. There are numerous factors, which would affect the model. Thus, it is impossible to estimate all of these factors. However, it is possible to calculate the probability that future values are within specific limits; these models are called stochastic models [17].

Furthermore, we have to distinguish between a stochastic model and a random process. The stochastic model is the precise description of the probability structure of an observed random process and a time series is one random realization of a random process. Depending on the statistical properties of a random process it can be stationary or non-stationary. For common stochastic models, a mandatory requirement is that the underlying process is at least wide-sense stationary.

Definition (stationary)**.** Stationary means that the properties of a random process are unaffected by a change in the time origin. Furthermore, the joint probability distribution is the same for any chosen interval in time. A random process is wide-sense stationary if the the mean is constant and autocorrelation depends only on the time lag.[5] [17]

In reality, many time series reveals some patterns, such as in Figure 2.2. Such patterns violate the conditions for stationary processes. Figure 2.2 visualizes the power consumption of a building for two weeks. The data was recorded by an electricity meter.

Without any mathematical calculation, it is obvious that the underlying random process is not stationary. As the time series is increasing over time, the mean cannot be constant, which violates the definition of a stationary process. There are two phenomena visible in Figure 2.2. First, the time series has a trend and second it has a seasonal component. The trend component is a result of the way how electricity consumption is metered. It is the cumulated sum of the energy used up to a point in time. Therefore the number is increasing. For further information, how to detrend a time series, see Chapter 5. Furthermore, there are different seasonality patterns for weekend and weekday, or day and night visible.

Definition (trend)**.** In this case, a trend means there exists an underlying, mostly linear, component in the time series, which defines a direction of further development.

Definition (seasonality)**.** The term seasonality describes a recurring pattern after a certain period in the time series, for example, every hour or every day.

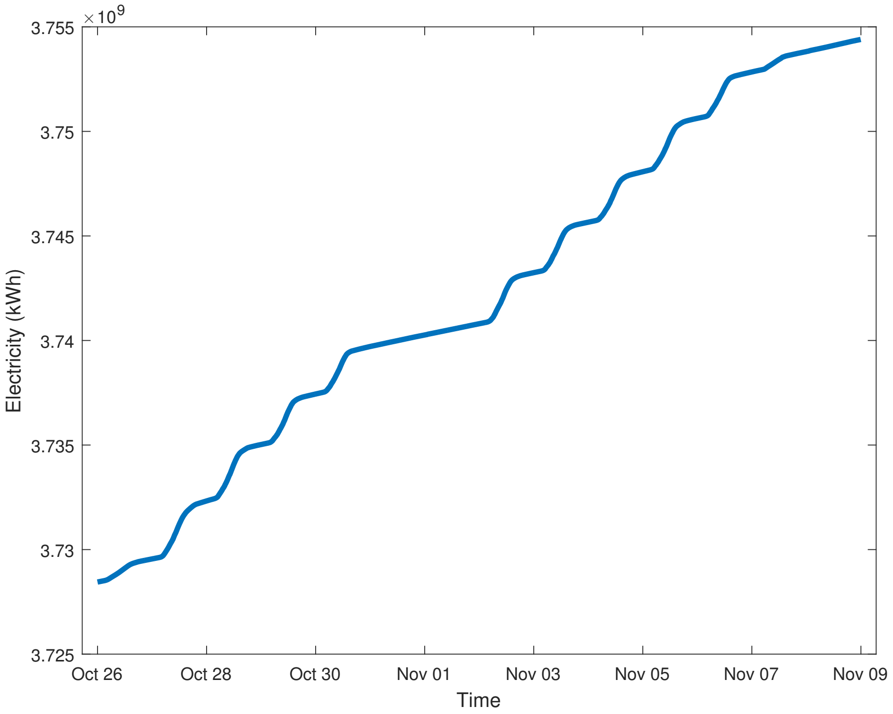


Figure 2.2: Illustration of a time series with trend and seasonality

The similarity between two random processes can be measured by the cross-correlation. It is essential to know the random model to calculate the cross-correlation. In many cases, the random process is unknown, and only a time series is available. Instead of using the cross-correlation, the available time series can be used to calculate a sample cross-correlation. For the sake of simplicity, the term cross-correlation is used for both, cross-correlation and sample cross-correlation. The meaning should be evident from the context.

Definition (correlation coefficient)**.** The correlation coefficient (also known as Pearson correlation coefficient) r_{xy} is a metric to rate the similarity between two random processes. The value of the correlation coefficient is between -1 and 1, where -1 indicates a negative correlation, 0 no correlation, and $+1$ a strong correlation.

The co-variance c_{xy} of two random processes x and y divided by the standard deviation s_{x} of x times the standard deviation s_{y} of y is the correlation coefficient r_{xy}. Equation 2.2 shows the sample co-variance, where T is the number of observations, and $\bar{x}$, $\bar{y}$ are the sample means of x respectively y. The standard deviation is the square root of the variance, see Equations 2.3 and 2.4. Equation 2.5 shows the sample cross-correlation.

$$c_{xy} = \frac{1}{T-1}\sum_{t=1}^{T}(x_t - \bar{x})(y_t - \bar{y}) \tag{2.2}$$

$$s_x = \sqrt{c_{xx}} = \sqrt{Var(x)} \tag{2.3}$$

$$s_y = \sqrt{c_{yy}} = \sqrt{Var(y)} \tag{2.4}$$

$$r_{xy} = \frac{c_{xy}}{s_x s_y} \tag{2.5}$$

In most cases, it is of particular interest to see how a time series correlate with another time series shifted in time. Therefore the *lag* variable k is introduced, which represents the time offset. The Equations 2.2 and 2.5 can be extended with the lag variable to the Equations 2.6 and 2.7. Equation 2.7 is called the cross-correlation.

$$c_{xy}(k) = \begin{cases} \frac{1}{T-1}\sum_{t=1}^{T-k}(x_t - \bar{x})(y_{t+k} - \bar{y}), k = 0, 1, 2, \cdots, K \\ \frac{1}{T-1}\sum_{t=1}^{T+k}(x_t - \bar{x})(y_{t-k} - \bar{y}), k = -1, -2, \cdots, -K \end{cases} \tag{2.6}$$

$$r_{xy}(k) = \frac{c_{xy}(k)}{s_x s_y}, -T \leq k \leq T \tag{2.7}$$

Definition (cross-correlation)**.** The cross-correlation measures the similarity between two random processes at different time offsets (lags). The result is a set of correlation coefficients for every lag $k \in \mathbb{Z}$ and $-T \leq k \leq T$.

On the other hand, the autocorrelation is used to identify the seasonality in a time series. The autocorrelation is a particular case of the cross-correlation, where a single random process is correlated with itself at different lags $k \in \mathbb{N}_0$. The autocorrelation is the covariance $c_{\mathsf{x}}(k)$ at lag k of a random process x divided by the variance of x. Equation 2.8 shows the sample autocorrelation. Which provides a meaningful output if there are at least 50 observations in the time series. The lag should be calculated up to $T/4$ [5].

$$r_x(k) = \frac{c_x(k)}{s_x^2} \tag{2.8}$$

Definition (autocorrelation)**.** The autocorrelation measures the similarity between a random process and itself shifted at different time offsets (lags).

The autocorrelation can be used to identify the seasonality and to estimate the recurring time span. Figure 2.3 shows the autocorrelation of the trend and seasonality sample of Figure 2.2. An alternating pattern with precisely 24 lags between positive peak correlations is visible. This interval represents the 24 hours of a day and is caused by the hourly samples of the time series. Furthermore, every week (i.e., 168 lags) there is a high correlation, marked by the red square, which is in the following referred to as a weekly pattern. A weekly pattern means that successive weeks correlate with each other.

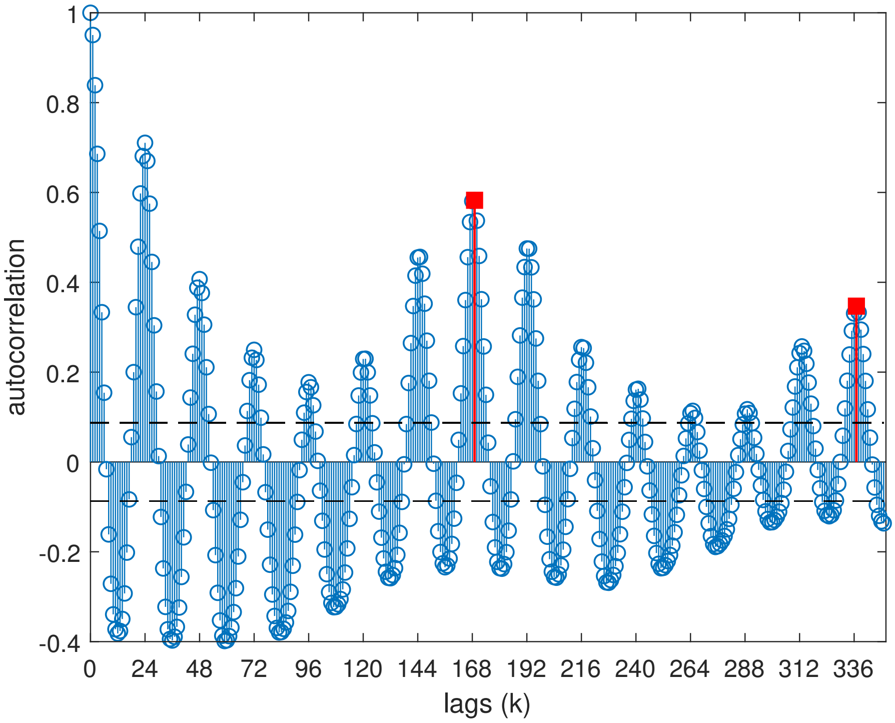


Figure 2.3: Autocorrelation of the trend and seasonality sample in Figure 2.2.

Definition (partial autocorrelation)**.** The partial autocorrelation is the conditional autocorrelation adjusted by the variables between x_t and x_{t+k}. [17]

CHAPTER 3

Linear models

As by Box, Jenkins, Reinsel, *et al.* [17] presented, linear stationary and non-stationary models can be used to represent a random process mathematically. In the following sections, the linear filter, AR, MA, ARMA, and ARIMA model are presented. These models are used in the following books and papers to predict the future of a time series: Cui and Peng [16] analyzed the relationship between temperature and electricity demand for the case of short-term forecasting. They used an improved ARIMAX model for predicting the short-term electric load under the influence of some exogenous inputs. They discovered an improvement over traditional time series models. Andrews, Dean, Swain, *et al.* [18] used the ARIMA and ARIMAX model to predict the long-term disability application rates in the public and private sectors. They discovered that both ARIMA and ARIMAX have the ability to deliver accurate four quarter forecasts.

The following simple operators are defined to be consistent with the literature [5] [17]. This operators are extensivly employed by Box, Jenkins, Reinsel, *et al.* [17] for linear filter, AR, MA, ARMA, and ARIMA.

Definition (simple operators)**.** *Backward shift operator* B : $Bx_t = x_{t-1}$, hence $B^m x_t = x_{t-m}$. *Forward shift operator* F : $F = B^{-1}, Fx_t = x_{t+1}$, hence $F^m x_t = x_{t+m}$. *Backward difference operator* ∇, $\nabla x_t = x_t - x_{t-1} = (1 - B)x_t$. [17]

3.1 Linear filter model

The linear filter model was introduced by Yule [19]. The basic idea is that every time series x can be modeled through a transformed white noise time series a. Figure 3.1 illustrates the transformation. White noise is a random process generated through a series of independent shocks with an identical distribution. In general, it is normally distributed with zero mean and variance σ_a^2. The transformation is realized with a linear filter $\Psi(B)$. Therefore the current white noise value a_t is added to a weighted sum of all previous noise values (see Equation 3.1).

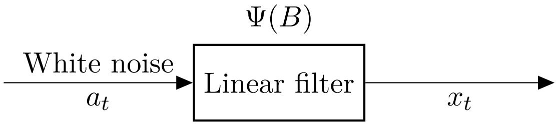


Figure 3.1: Linear filter model, adapted from [17, p. 8]

$$\begin{aligned} x_t &= \mu + a_t + \Psi_1 a_{t-1} + \Psi_2 a_{t-2} + \cdots \\ &= \mu + a_t + \sum_{i=1}^{t} \Psi_i a_{t-i} \\ &= \mu + \Psi(B) a_t \end{aligned} \tag{3.1}$$

In Equation 3.1 μ represents the level of the random process x and $\Psi(B)$ is the transfer function of the presented filter [17].

$$\Psi(B) = 1 + \Psi_1 B + \Psi_2 B^2 + \cdots \tag{3.2}$$

The set of weights $\Psi_1, \Psi_2, \cdots$ can either be finite or infinite. The linear filter is stable if the weights are finite, or infinite and the sum of the absolute weights is less than infinite (see Equation 3.3). If the linear filter is stable then the random process x is weakly stationary. Based on this definition, some essential stochastic models can be derived, like the AR or MA model.

$$\sum_{i=0}^{\infty} |\Psi_i| < \infty \tag{3.3}$$

3.2 Autoregressive model

An important stochastic model to represent a random process is the autoregressive (AR) model. It is a special case of a linear filter. Assuming that the values are equally sampled over time $t, t-1, t-2, \cdots$, then the current value x_t is represented by a linear combination of previous values $x_{t-1}, x_{t-2}, \cdots, x_{t-p}$ and a random shock a_t [17, p. 9]. Equation 3.4 represents the mathematical definition of an AR model, whereas μ represents the level. If a random process corresponds to this model, it is called an AR process of order p. Such a model is very flexible depending on the choice of the weights ψ [20].

$$x_t = \mu + a_t + \psi_1 x_{t-1} + \psi_2 x_{t-2} + \cdots + \psi_p x_{t-p} \tag{3.4}$$

For further simplicity, the random process is centered $\tilde{x}_t = x_t - \mu$. Thus, Equation 3.4 is simplified to

$$
\begin{aligned}
\tilde{x}_t &= a_t + \sum_{i=1}^{p} \psi_i \tilde{x}_{t-i} && (3.5) \\
\tilde{x}_t - \sum_{i=1}^{p} \psi_i \tilde{x}_{t-i} &= a_t && (3.6) \\
\tilde{x}_t - \psi_1 \tilde{x}_{t-1} - \psi_2 \tilde{x}_{t-2} - \cdots - \psi_p \tilde{x}_{t-p} &= a_t && (3.7) \\
B^0 \tilde{x}_t - \psi_1 B^1 \tilde{x}_t - \psi_2 B^2 \tilde{x}_t - \cdots - \psi_p B^p \tilde{x}_t &= a_t && (3.8)
\end{aligned}
$$

After rearranging Equation 3.5 and using the backward shift operator the AR transfer function $\psi(B)$ is defined as follows

$$\psi(B) = 1 - \psi_1 B - \psi_2 B^2 - \cdots - \psi_p B^p \tag{3.9}$$

So that Equation 3.8 can be simplified to

$$\psi(B)\tilde{x}_t = a_t \tag{3.10}$$

The AR model has $p + 2$ unknown parameters, $\mu, \psi_1, \psi_2, \cdots, \psi_p, \sigma_a^2$, which can be estimated from a time series [17]. The short notation for AR model of order p is $AR(p)$. An AR model of order p can be stationary or non-stationary. It is stationary if the roots of $\psi(B)$, $\psi(B) = 0$, are greater than 1 in absolute value, this means that all roots lie outside the unit circle. The partial autocorrelation is used to estimate the order p of the AR model. It measures the linear relationship between a random variable x_t and x_{t+k} by eliminating the linear dependence of the intermediate variables.

Figure 3.2 visualizes an example of an AR process of order 1 with the associated partial autocorrelation diagram. Typical for an AR process is that the partial autocorrelation ideally cuts off to 0 after p lags. In a real-world application the AR time series will be near 0 and within the confidence interval (black dashed lines) after p lags.

3.3 Moving average model

Similar to the AR model, the moving average (MA) model is an important model to describe a random process. Unlike the AR model, the MA model is a linear combination of a finite number of random shocks $a_t, \forall t \in \mathbb{Z}$, see Equation 3.11.

$$\tilde{x}_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \cdots - \theta_q a_{t-q} \tag{3.11}$$

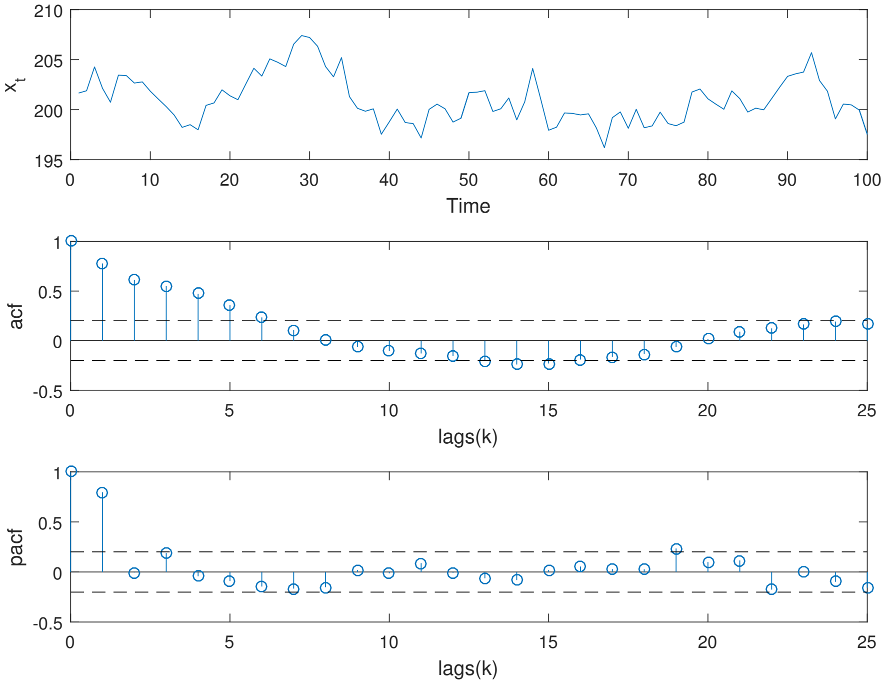


Figure 3.2: Autocorrelation and partial autocorrelation functions for the realizations of an AR time series, $x_t = 40 + a_t + 0.8x_{t-1}$, $\sigma_a^2 = 2$

A random process is called an MA process of order q if it can be represented by Equation 3.11. Every MA process is stationary because it is the sum of a finite number q of stationary white noise processes a_t.

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q \tag{3.12}$$

Equation 3.12 defines the MA transfer function $\theta(B)$, so that Equation 3.11 can be simplified to

$$\tilde{x}_t = \theta(B) a_t \tag{3.13}$$

In order to solve the Equation 3.13, there are q+2 unknown parameters, $\mu, \theta_1, \theta_2, \cdots, \sigma_a^2$, to estimate [17]. As in the AR, model these parameters can be estimated from a realization of the MA process. The short notation for a MA model with order q is $MA(q)$. For estimating the order q of the MA model, the autocorrelation can be used. In an MA process, the autocorrelation ideally cuts off to 0 after q lags. Figure 3.3 visualizes an $MA(1)$ process. In a real-world application, the MA time series will be near 0 and within the confidence interval (black dashed lines) after q lags.

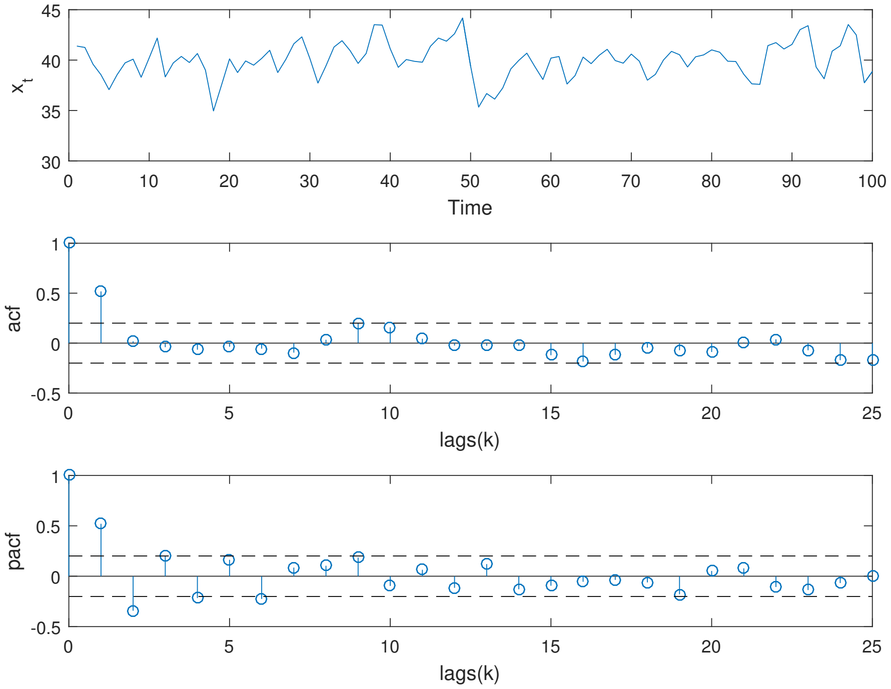


Figure 3.3: Autocorrelation and partial autocorrelation functions for the realizations of a MA time series, $x_t = 40 - a_t - 0.8a_{t-1}$, $\sigma_a^2 = 2$

3.4 Autoregressive moving average model

Many random processes in the real world do not strictly correspond to either AR or MA models, but they are more or less a combination of both models. Equation 3.14 shows the combined model, which is also called the autoregressive moving average (ARMA) model.

$$\tilde{x}_t = \psi_1 \tilde{x}_{t-1} + \psi_2 \tilde{x}_{t-2} + \cdots + \psi_p \tilde{x}_{t-p} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \cdots - \theta_q a_{t-q} \tag{3.14}$$

Such a model is well suited to model nearly every stationary process in the real world, with p and q not greater than 2 [17]. Simplifying Equation 3.14 with the use of 3.10 and 3.13 leads to

$$\psi(B)\tilde{x}_t = \theta(B)a_t \tag{3.15}$$

The number of unknown parameters is the sum of unknown parameters of the AR and MA model, so that $p+q+2$ unknown parameters, $\mu, \psi_1, \cdots, \psi_p, \theta_1, \cdots, \theta_q, \sigma_a^2$, needs to be estimated. The short notation for a ARMA model with order (p, q) is $ARMA(p, q)$.

An ARMA model can be transformed to an AR or MA model by either setting $q = 0$ or $p = 0$.

3.5 Autoregressive integrated moving average model

As mentioned in Section 3.4, the ARMA model works well for stationary processes but not for non-stationary processes. A random process is non-stationary if the mean is not constant and the autocorrelation depends not only on the time lag, which implies that the statistical properties differs over time. Many real-world time series are non-stationary as most of them have a trend or seasonality. To deal with a trend component in a random process, the autoregressive integrated moving average (ARIMA) model was introduced by Box, Jenkins, Reinsel, *et al.* [17]. If an ARMA process is non-stationary, either the AR or MA process must be non-stationary. As in Section 3.3 described an MA process is always stationary, which implies that the AR process of the ARMA process must be non-stationary. As mentioned in Section 3.2, an AR process is stationary if the roots of the AR transfer function $\psi(B) = 0$ are greater then 1 in absolute value. The roots lie outside the unit circle. If a random process is non-stationary, then there is at least one root that lies inside the unit circle. The number of roots inside the unit circle is denoted with d. There are several ways to transform a non-stationary process to a stationary process, furthermore in Chapter 5, one possible way is to differentiate the time series. The ARIMA model uses the difference operator $\nabla x_t = (1 - B)x_t$ to differentiate the time series. It depends on the random process, how often the difference operator has to be applied. The number of roots inside the unit circle d denotes how often the difference operator has to be applied. Applying the difference operator to equation 3.15 leads to

$$\psi(B)\nabla^d x_t = \theta(B)a_t \tag{3.16}$$

A process that can be described by this model is called autoregressive integrated moving average (ARIMA) process of order (p, d, q) or short $ARIMA(p, d, q)$. Furthermore, the ARIMA model can be extended to support seasonality. For this purpose a second ARIMA model with a seasonality parameter s is multiplied with the base ARIMA model, in short $SARIMA(p, d, q) \times (P, D, Q)_s$. The corresponding seasonal difference operator is defined as $\nabla_s^D x_t = (1 - B^D)x_t = x_t - x_{t-D}$.

$$\psi_p(B)\Psi_P(B^s)\nabla^d\nabla_s^D x_t = \theta_q(B)\Theta_Q(B^s)a_t \tag{3.17}$$

CHAPTER 4

Nonlinear models

In the previous chapter, raw statistical algorithms were presented, where an expert has to analyze the data of the time series and needs to select an appropriate model and parameters. If the underlying random process changes over time, the parameters must be manually adjusted. This adjustment is a massive effort because this requires a permanent monitoring and performance evaluation. Machine learning is an entirely different approach. It is a collection of algorithms which do not rely on an expert. The method tries to learn all necessary parameters from the data itself. There are two fundamentally different approaches in the way of presenting the training data to the machine learning algorithm: supervised learning and unsupervised learning.

Definition (supervised learning)**.** “In supervised learning, a teacher, provides a category label or cost for each pattern in a training set, and we seek to reduce the sum of the costs for these patterns.” [21, p. 16]

Definition (unsupervised learning)**.** “In *unsupervised learning* or *clustering* there is no explicit teacher, and the system forms clusters or “natural groupings” of the input patterns. “Natural” is always defined explicitly or implicitly in the clustering system itself, and given a particular set of patterns or cost function, different clustering algorithms lead to different clusters.” [21, p. 17]

Furthermore, a distinction must be made between classification and regression problems. The latter is in the focus of this thesis. In classification theory, a machine learning algorithm tries to separate two or more classes by a discrete set of discriminate features. In regression theory, the machine learning algorithm tries to approximate the presented time series. This approximation can be used to predict the future behavior of the time series based on historical data.

In the following sections two supervised learning algorithms are presented, namely ANN and SVM.

4.1 Artificial neural network

The concept of neural networks is based on the biological brain. Therefore, in the early days the science of neural networks was called “brain theory”. A neural network consists of a massive number of neurons, which are connected to a complicated structure, where many excitatory and inhibitory inputs are received. In the theory of artificial neural network (ANN), only one neuron is specified, the perceptron in contrast to biology where many different neurons exist. In ANN, the neuron is an abstraction of the biological neuron and can be cascaded to form different structures of neural networks, for example, feedforward neural network (FFNN) or recurrent neural network (RNN). Even a single neuron can form a network. The limit for such a structure is the ability to train the network. If it is not possible to train all the weights of the neural network, it is useless. A widely used supervised training technique is the backpropagation training that is explained in the next sections. Neural networks, in contrast to statistical models like ARMA, can be used to model stationary and non-stationary random processes. Furthermore, they can be used to model non-linear behavior.

The following symbols are introduced for the following subsections:

- n is the index of input nodes/weights and N is the number of input nodes/weights
- m is the index of hidden nodes, and M is the number of hidden nodes
- r is the index of output nodes, and R is the number of output nodes
- p is the index of training patterns, and P is the number of patterns

4.1.1 The perceptron

A perceptron is a mathematical representation of a biological neuron that was introduced by Rosenblatt [22]. Unlike the biological neuron, where continuous “trains” of impulses are fed into it. The perceptron takes discrete values as input: $x_1, \cdots, x_N$ and the bias b. These inputs are multiplied with the weight vector $\boldsymbol{w} = w_0, \cdots, w_N$ and summed up. The sum is the input for the activation function h. The output of the perceptron y is the result of the activation function. Figure 4.1 visualizes a perceptron. The number of inputs to the perceptron is variable, but must be a finite number N. The Equations 4.1 and 4.2 show the mathematical representation of Figure 4.1, where x_0 represents the bias b, $x_1, \cdots, x_N$ the inputs, h the activation function and y the output.

The activation function of the perceptron can be any differentiable function. Table 4.1 provides a list of commonly used activation functions. Except for the linear function, the output of the activation function is in the range between -1 and 1 or 0 and 1. This means that the input of the perceptron should also be in this range (see Chapter 5).

$$net = \sum_{n=0}^{N} x_n w_n \tag{4.1}$$

$$y = h\left(net\right) \tag{4.2}$$

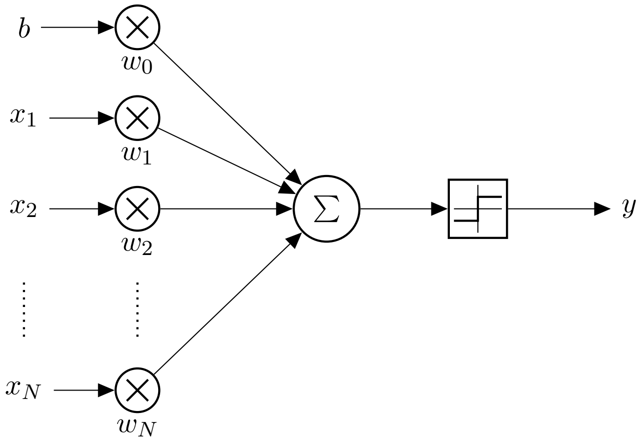


Figure 4.1: Perceptron

Table 4.1: Commonly used activation functions for perceptrons

activation function	formula	value range
linear function	$h(x) = x$	-
step function	$h(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases}$	$0, 1$
tanh function	$h(x) = \tanh(x)$	$-1 \ldots 1$

With a single perceptron and a linear activation function, linear regression tasks can be solved. For example, a perceptron is fed with two inputs x_1 and x_2, and the activation function is linear. Then, the perceptron represents a linear equation, see Equation 4.3.

$$y = w_0 + x_1 w_1 + x_2 w_2 \tag{4.3}$$

To approximate any arbitrary function, either the activation function can be varied, or multiple perceptrons can be cascaded to a network. Two basic types of neural networks are feedforward and recurrent neural networks.

There are several learning methods to find a suitable weight vector $\boldsymbol{w}$, that minimizes the loss (error) l. The loss for a given input vector $\boldsymbol{x}$ and a weight vector $\boldsymbol{w}$ is the difference between the desired value t, also called target, and the output of the perceptron y (see Equation 4.4).

$$l(\boldsymbol{x}, \boldsymbol{w}, t) = t - y = t - h\left(\sum_{n=0}^{N} x_n w_n\right) \tag{4.4}$$

Given a training set $\{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \cdots, (\boldsymbol{x}_P, t_P)\}$, the overall loss $L(\boldsymbol{w})$ is the sum of squared differences of each sample (see Equation 4.5).

$$L(\boldsymbol{w}) = \frac{1}{2}\sum_{p=1}^{P} l(\boldsymbol{x_p}, \boldsymbol{w}, t_p)^2 \tag{4.5}$$

The EBP (gradient descent) training algorithm, proposed by Rumelhart, Hinton, and Williams [23], is a simple training algorithm for perceptrons. It is a first-order method, which means it calculates the gradient g of the total loss function, to find a minimal error (see Equation 4.6).

$$\boldsymbol{g} = \frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \nabla \boldsymbol{w} \tag{4.6}$$

Algorithm 4.1 shows the pseudo-code. After every iteration k of the training data, the gradient $\boldsymbol{g}$ is calculated, and the weight update rule is applied (see Equation 4.7). Then $\boldsymbol{w}_{k+1}$ is the new weight vector for the next iteration $k+1$ and α is the learning constant.

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha \boldsymbol{g}_k \tag{4.7}$$

Algorithm 4.1: EBP training algorithm

```
initialize $\boldsymbol{w}, \alpha$;
for $k \leftarrow 1$ to $K$ do
    for $p \leftarrow 1$ to $P$ do
        $y_p = h\left(\sum_{n=0}^{N} x_n w_n\right)$;
        $L \leftarrow L + \frac{1}{2}\left(t_p - y_p\right)^2$;
    end
    $g \leftarrow \frac{\partial L}{\partial \boldsymbol{w}}$;
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha g$;
end
```

Usually, the learning constant α is a fixed value, but it can also be a more sophisticated variable value. If the value is too big, the algorithm will not find a minimum. If it is too small, the algorithm will slowly converge against the minimum.

4.1.2 Feedforward neural network

A feedforward neural network (FFNN) is characterized by its layered design and the direction of the information flow [21] [24]. Every layer consists of a discrete number of perceptrons, called nodes, and is connected to another layer. Every node of a layer is fully connected to the nodes of the following layer, but not among each other. There are at least three layers: the input, hidden and the output layer. The information is propagated from the input to the output layer without any feedback or recurrent loops. The input layer consists of input nodes, which are *passive* nodes. Thus, they only propagate the input further to the hidden layer, without any computation. Equation 4.8 illustrates the mathematical representation for one node m of the hidden layer, where x_n is the output of the input layer, $w_{n,m}$ the according weights and y_m is the output of the hidden node.

$$y_m = h_y\left(\sum_{n=0}^{N} x_n w_{n,m}\right) \tag{4.8}$$

The hidden layer has several nodes, and there can be more than one hidden layer. The exact number depends on the application. If there is more than one hidden layer with a huge number of nodes, the network is called *deep* or *deep neural network* [13].

The output of the last hidden layer y_m is fed as input to the output nodes, to generate the network output (see Equation 4.9). Figure 4.2 visualizes an example feedforward network, where the input nodes are green (solid), the hidden nodes are blue (dashed), and the output nodes are yellow (dotted).

$$o_r = h_o\left(\sum_{m=0}^{M} y_m w_{m,r}\right) \tag{4.9}$$

For the hidden and output layer, the activation function can be chosen individually depending on the actual application. In general, every node of one layer has the same activation function according to related literature.

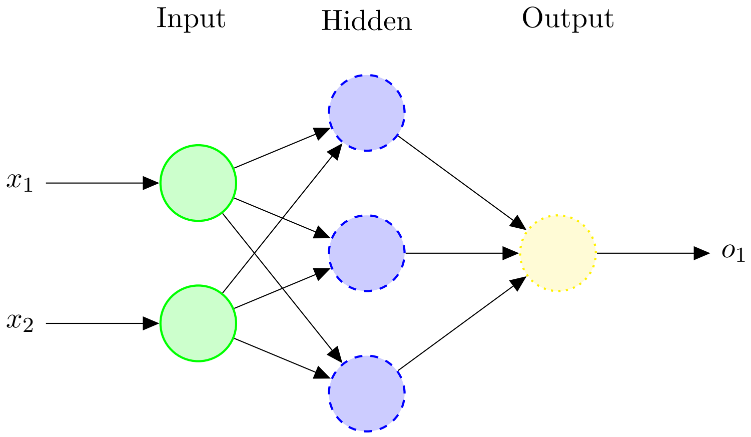


Figure 4.2: Exemplary FFNN with two input nodes, three hidden nodes and one output node

FFNNs can be trained with several algorithms. Most of the algorithms are related to the EBP algorithm. There are some extensions to this algorithm like Newton's method, the Gauss-Newton algorithm, and the LM. For this reason, the loss function must be extended so that the loss can be calculated for more than one output node, and the overall loss function as well. Therefore the Equation 4.10 and Equation 4.11 is extended by the subscript r and the overall loss functions sums over all output nodes.

$$l(\boldsymbol{x_p}, \boldsymbol{w}, t_{p,r}) = t_{p,r} - o_r \tag{4.10}$$

$$L(\boldsymbol{w}) = 1/2 \sum_{p=1}^{P} \sum_{r=1}^{R} l\left(\boldsymbol{x_p}, \boldsymbol{w}, t_{p,r}\right)^2 \tag{4.11}$$

Newton's Method

One significant drawback of the EBP algorithm is that it converges very slowly to a minimum. To speed it up, the Newton method uses the first and second derivative of the loss function. The second derivative is grouped in the Hessian matrix $\boldsymbol{H}$. The gradient is defined as following

$$-\boldsymbol{g} = \boldsymbol{H} \nabla \boldsymbol{w} \tag{4.12}$$

By rearranging it follows

$$\nabla \boldsymbol{w} = -\boldsymbol{H}^{-1} \boldsymbol{g} \tag{4.13}$$

Therefore the update rule for Newton's method is

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \boldsymbol{H}_k^{-1} \boldsymbol{g}_k \tag{4.14}$$

This method converges faster but does not always find a minimum. A second drawback is that the computation of the Hessian matrix is computational expensive [25].

Gauss-Newton Algorithm

The Gauss-Newton algorithm is an improved version of Newton's method. It avoids the computation of the Hessian matrix by using an approximation. Therefore the Jacobian matrix $\boldsymbol{J}$ is used (see Equation 4.15).

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial l(\boldsymbol{x}_1,\boldsymbol{w},t_{1,1})}{\partial w_1} & \frac{\partial l(\boldsymbol{x}_1,\boldsymbol{w},t_{1,1})}{\partial w_2} & \dots & \frac{\partial l(\boldsymbol{x}_1,\boldsymbol{w},t_{1,1})}{\partial w_N} \\ \frac{\partial l(\boldsymbol{x}_1,\boldsymbol{w},t_{1,2})}{\partial w_1} & \frac{\partial l(\boldsymbol{x}_1,\boldsymbol{w},t_{1,2})}{\partial w_2} & \dots & \frac{\partial l(\boldsymbol{x}_1,\boldsymbol{w},t_{1,2})}{\partial w_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial l(\boldsymbol{x}_1,\boldsymbol{w},t_{1,R})}{\partial w_1} & \frac{\partial l(\boldsymbol{x}_1,\boldsymbol{w},t_{1,R})}{\partial w_2} & \dots & \frac{\partial l(\boldsymbol{x}_1,\boldsymbol{w},t_{1,R})}{\partial w_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial l(\boldsymbol{x}_P,\boldsymbol{w},t_{P,1})}{\partial w_1} & \frac{\partial l(\boldsymbol{x}_P,\boldsymbol{w},t_{P,1})}{\partial w_2} & \dots & \frac{\partial l(\boldsymbol{x}_P,\boldsymbol{w},t_{P,1})}{\partial w_N} \\ \frac{\partial l(\boldsymbol{x}_P,\boldsymbol{w},t_{P,2})}{\partial w_1} & \frac{\partial l(\boldsymbol{x}_P,\boldsymbol{w},t_{P,2})}{\partial w_2} & \dots & \frac{\partial l(\boldsymbol{x}_P,\boldsymbol{w},t_{P,2})}{\partial w_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial l(\boldsymbol{x}_P,\boldsymbol{w},t_{P,R})}{\partial w_1} & \frac{\partial l(\boldsymbol{x}_P,\boldsymbol{w},t_{P,R})}{\partial w_2} & \dots & \frac{\partial l(\boldsymbol{x}_P,\boldsymbol{w},t_{P,R})}{\partial w_N} \end{bmatrix} \tag{4.15}$$

The gradient $\boldsymbol{g}$ is defined as

$$\boldsymbol{g} = \boldsymbol{J}\boldsymbol{l} \tag{4.16}$$

where the vector $\boldsymbol{l}$ is defined as

$$\boldsymbol{l} = \begin{bmatrix} l(\boldsymbol{x}_1, \boldsymbol{w}, t_{1,1}) \\ l(\boldsymbol{x}_1, \boldsymbol{w}, t_{1,2}) \\ \vdots \\ l(\boldsymbol{x}_1, \boldsymbol{w}, t_{1,R}) \\ \vdots \\ l(\boldsymbol{x}_P, \boldsymbol{w}, t_{P,1}) \\ l(\boldsymbol{x}_P, \boldsymbol{w}, t_{P,2}) \\ \vdots \\ l(\boldsymbol{x}_P, \boldsymbol{w}, t_{P,R}) \end{bmatrix} \tag{4.17}$$

The Hessian matrix is approximated by the product of the transposed Jacobian matrix with itself.

$$\boldsymbol{H} \approx \boldsymbol{J}^T \boldsymbol{J} \tag{4.18}$$

By using Equation 4.16 and 4.18 in Equation 4.14, the weight update rule for the Gauss-Newton algorithm results.

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - (\boldsymbol{J}_k^T \boldsymbol{J}_k)^{-1} \boldsymbol{J}_k \boldsymbol{l}_k \tag{4.19}$$

The algorithm is, similar to the Newton's method, unstable to find a minimum in the error space. The reason is that the matrix $\boldsymbol{J}^T\boldsymbol{J}$ is not always invertible [25].

Levenberg-Marquardt Algorithm

As the Gauss-Newton algorithm, the Levenberg-Marquardt algorithm (LM) is an extension of Newton's method [26]. The approximation for the Hessian matrix is

$$\boldsymbol{H} \approx \boldsymbol{J}^T \boldsymbol{J} + \mu \boldsymbol{I} \tag{4.20}$$

where μ is always positive and $\boldsymbol{I}$ is the identity matrix. Both together ensure that the approximated Hessian matrix $\boldsymbol{H}$ is positive definite [27]. Thus, the matrix $\boldsymbol{H}$ is always invertible [25]. Inserting Equation 4.20 into 4.14 leads to

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \left(\boldsymbol{J}_k^T \boldsymbol{J}_k + \mu \boldsymbol{I}\right)^{-1} \boldsymbol{J}_k \boldsymbol{l}_k \tag{4.21}$$

which is the update rule for the LM. The LM algorithm updates the parameter μ depending on the total squared loss L. This means if L is increased after an iteration, μ is multiplied by some factor β. If L is reduced after an iteration, μ is divided by β. When the parameter μ is large, the algorithm becomes EBP with step size $1/\mu$. If μ is small, the algorithm approximates to the Gauss-Newton algorithm[25] [26].

4.1.3 Recurrent neural network

Similar to the FFNN, the recurrent neural network (RNN) can have multiple layers, i.e., input, hidden and output layer. Each layer consists of a discrete number of nodes and can have a different activation function. Compared to a FFNN, where the information flow is always from the input to the output layer, a RNN has at least one feedback loop, also called recurrent loop. Such a loop acts as a *memory* or *delay unit*. In FFNNs, every presented input is treated as new information. In RNN, past values can be fed as input to the network, through the feedback loop. RNN can be used to build nonlinear AR models. There are several different implementations of the concept of recurrent neural networks, like Jordan or Elman networks.

Jordan network

The Jordan network was introduced by Jordan [28]. He claimed that any human behavior reveals a certain set of serially ordered action sequence [28]. Therefore, he evaluated a way to represent serially ordered action sequences with neural networks. Such a single action in the sequence depends highly on the action before, so that they have a high correlation. He extended an FFNN by adding context nodes, such that recurrent loops act as memory cells. These context nodes propagate the output of the network without any further computation but with a time delay to the hidden layer. The context nodes enhance an FFNN to represent time constraints within the network. Figure 4.3 presents an exemplary Jordan neural network. The context nodes are red (loosely dashed). The number of context nodes depends only on the number of outputs of the neural network.

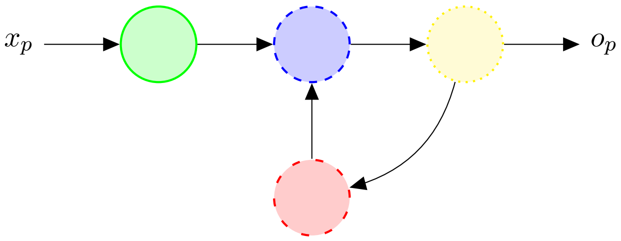


Figure 4.3: Jordan neural network

A basic example of a Jordan neural network is to predict the next state o_p of a process. Every process depends on some input variables x_p and the previous state o_{p-1}.

A Jordan neural network can easily be transformed to a FFNN by unrolling the recurrent loop as visualized in Figure 4.4. Therefore, the recurrent node is transformed to an input node, such that the new input vector is $\boldsymbol{x} = \{x_p, o_{p-1}\}$. Such a network can be trained with any algorithm suitable for FFNN.

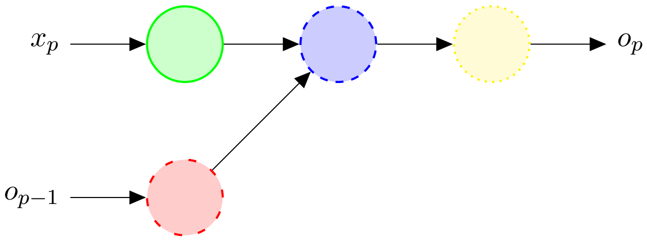


Figure 4.4: Unrolled Jordan neural network

Elman network

Elman described an approach, where the time is modeled implicitly within a network, rather than explicitly by an input feature [29]. The Elman neural network is, as the Jordan network, a type of recurrent neural network. The difference to a Jordan network is that the output of the network is not fed back to the hidden layer. Instead, the output of the hidden layer is fed back. Similar to a Jordan network, context nodes are used to delay the information flow. Figure 4.5 visualizes an example Elman neural network, where the context node is red (loosely dashed). The number of context nodes is equal to the number of hidden nodes. Similar to a Jordan network, an Elman network can be unrolled to a FFNN as well (see Figure 4.6). Such a network can be trained with any training algorithm for FFNN.

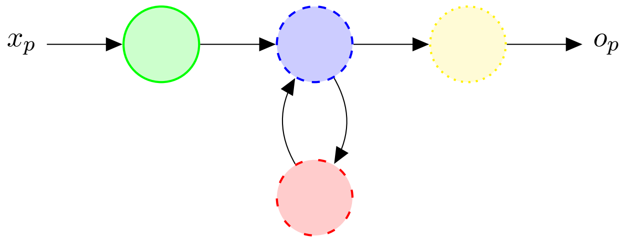


Figure 4.5: Elman neural network

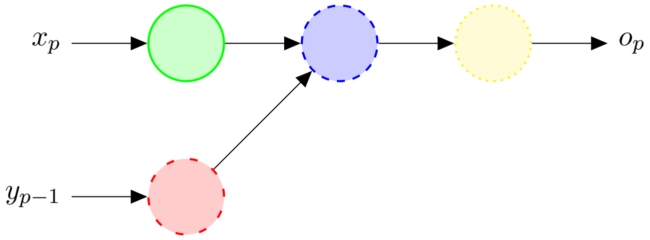


Figure 4.6: Unrolled Elman neural network

4.2 Support vector machine

The theory behind the support vector machine (SVM) is called statistical learning theory or VC theory developed by Vapnik and Chervanenkis [30]. Based on this theory, the SVM was developed at the AT&T Bell Laboratories by Vapnik and his co-workers [31]. The SVM is used for many classification and regression tasks, like object classification or time series prediction [7]. For example, an SVM can be used to classify two different objects. Therefore, discriminant properties of the objects are used as inputs. In the training process for the SVM, a linear boundary between the two classes is sought.

When an SVM is used to solve a regression task, it is called support vector regression (SVR). In the linear case, the SVR tries to estimate a linear function based on some given sample points. In contrast to ARIMA, there is no need to estimate certain parameters. Equation 4.22 shows the mathematical definition for a linear SVR, where $f(\boldsymbol{x})$ is the outcome of the SVR applied to the input vector $\boldsymbol{x}$ with $\boldsymbol{w}$ as weight vector and b as threshold.

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b \tag{4.22}$$

Solving non-linear tasks with SVR, Equation 4.22 must be enhanced by the kernel trick. The basic idea is that a kernel k maps the input space to a higher dimensional feature space, where the problem is a linear function or linearly separable [7]. Such a mapping is defined as $\varphi : X \to F$, where X is the input space and F is the feature space. Using the mapping the kernel is defined as

$$k(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T \varphi(\boldsymbol{x}') \tag{4.23}$$

so that if Equation 4.22 and Equation 4.23 are combined, it follows

$$f(\boldsymbol{x}) = \boldsymbol{w}^T k(\boldsymbol{x}, \boldsymbol{x}') + b \tag{4.24}$$

Table 4.2 shows commonly used kernel functions.

Table 4.2: Common kernels for SVM

kernel	formula
linear	$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$
polynomial	$k(\boldsymbol{x}, \boldsymbol{x}') = \left(\boldsymbol{x}^T \boldsymbol{x}' + c\right)^d$
radial basis function	$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\lvert\lvert\boldsymbol{x}-\boldsymbol{x}'\rvert\rvert^2}{2\sigma^2}\right)$
sigmoid	$k(\boldsymbol{x}, \boldsymbol{x}') = \tanh\left(\gamma \boldsymbol{x}^T \boldsymbol{x}' + c\right)$

There are different strategies to find the optimal weight vector $\boldsymbol{w}$, e.g., ν-SVR or ϵ-SVR. They differ mostly in the way how to tolerate values that lie outside of a margin around the linear border. In this thesis, the ϵ-SVR is used, in later sections, called SVR. The ϵ-SVR tries to find a function $f(\boldsymbol{x})$, where the deviation from the real obtained targets y is at most ϵ for all training data. At the same time, is as flat as possible [31] [32]. Empirical risk management can achieve this. The Euclidean norm can measure the flatness of the weights ($\min ||\boldsymbol{w}||$).

Equation 4.25 is minimized with subject to Equation 4.26, where C is a positive numeric value, which is a regularization term. It controls the penalty for outliers of the ϵ-margin, which prevents overfitting. The value of C also depends on the tradeoff between the flatness of $f(\boldsymbol{x})$ and the toleration of outliers of the ϵ-margin. As stated before, the ϵ-SVR tries to find a linear solution where all presented data points are within the ϵ-margin. Sometimes, it is not possible to find such a solution because there are some outliers in the presented data. For this case, the parameter ξ defines a slack variable so that errors up to the value of ξ are tolerated. This is similar to the concept of soft margin in SVM classification. Figure 4.7 visualizes the connection between the data points, the ϵ margin and the slack variables ξ and ξ^*.

$$\frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{p=1}^{P} \left(\xi_p + \xi_p^*\right) \tag{4.25}$$

$$\text{subject to} \begin{cases} t_p - \boldsymbol{w}^T k(\boldsymbol{x}_p, \boldsymbol{x}_p') - b \leq \epsilon + \xi_p \\ \boldsymbol{w}^T k(\boldsymbol{x}_p, \boldsymbol{x}_p') + b - t_p \leq \epsilon + \xi_p^* \\ \xi_p \geq 0 \\ \xi_p^* \geq 0 \end{cases} \tag{4.26}$$

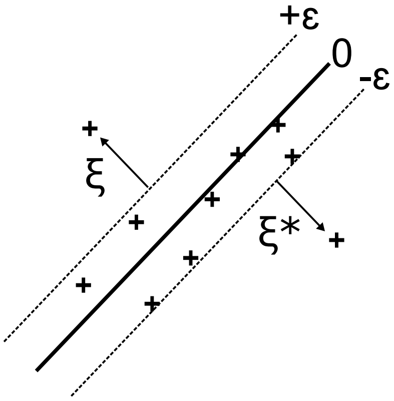


Figure 4.7: Linear SVM, soft margin visualization, adapted from [31]

Equation 4.27 shows the loss function for an ϵ-SVR. This means that all losses, where points are within an ϵ-margin, are ignored, and all others were counted.

$$L_\epsilon = \begin{cases} 0, & \text{if } |y - f(\boldsymbol{x})| \leq \epsilon \\ |y - f(\boldsymbol{x})| - \epsilon, & \text{otherwise} \end{cases} \tag{4.27}$$

The presented minimization is called primal formula [32] or primal objective function [31]. A major drawback of this method is the high computational effort to solve the minimization problem. The problem can be simpler solved in its Lagrange dual formulation, which is a quadratic programming optimization problem. Another efficient solver is the sequential minimal optimization (SMO) algorithm it is primarily as the quadratic programming algorithm, but instead of trying to solve one large quadratic programming problem, it breaks the big problem into a series of small problems. The algorithm was invented by Platt in the year 1998 at Microsoft Research. The main difference between these two approaches is that, previously, the quadratic programming problem is numerically solved, whereas SMO uses analytical steps. [33]

CHAPTER 5

Data preparation and preprocessing

For most buildings, there exists no exact model, which can predict future behavior, e.g., energy consumption, heating, or temperature in the building. These models are very expensive to create and maintain due to the changing conditions inside and outside of the building. Therefore, approaches were searched which can create a model without any user interaction and detailed knowledge of the building. After extensive literature research, there were two classes of approaches which can achieve this goal. First, the ARIMA approach is widely used in econometrics to predict sales figures, but it is also capable of predicting any given time series. Therefore the time series needs preprocessing to ensure proper accuracy. Different from the ARIMA approach, the ANN and SVM were two machine learning approaches. These approaches try to build a model out of the historical data by themselves. The difference between an ANN and an SVM is that the first uses a dedicated network structure to predict future values, whereas the SVM uses a transformation to model the forecasting problem as a linear regression. Both approaches were trained with input features, which were generated out of the historical data. All three approaches need significantly less effort than a concrete mathematical building model to either create and maintain. Therefore, these approaches were further investigated in this thesis. This chapter presents ways to prepare the historical data so that a forecasting algorithm can be applied.

5.1 Data selection and preparation

This thesis uses three different types of data sources. Every data source is a building and provides different sensor data. All buildings are located in Austria and have following characteristics:

- *Building 1* is an office building that is constructed according to the passive house standard. It was built in 2008 and offers 7500m^2 of office space. It was specially constructed to meet the goals of energy efficiency, user comfort, and use of renewable energy.

- *Building 2* is an office building and was built in 2010 with a focus on energy efficiency, intelligent BAS, and sustainable construction.

- *Building 3* is a modern and intelligent residential complex built in 2012.

In general, every building provides different types of sensors, like energy, district heating, cooling, temperature, humidity, and CO_2. The length of the provided historical data is between one and three years, and the sampling interval is between 15 minutes and one hour. Every data entry consists of a timestamp and the corresponding value.

As mentioned in the problem statement in Section 1.2, a future BAS should provide an optimization service. This service should optimize the building regarding different aims, like energy efficiency, user comfort, or cost efficiency. Therefore, many building services need to work together coordinated by an optimization service. This service should predict future development, e.g., photovoltaic energy production, heat consumption, or temperatures to make the right decision so that the aims chosen by the operator are reached. Therefore, the following types of sensors are relevant: electricity, district heating, humidity, temperature, and photovoltaic. Forecasts of the electricity and photovoltaic production can be used for demand side management, where either the energy costs or the stability of the network is optimized. Furthermore, the use of district heating, temperature, and humidity forecasts can help to ensure user comfort. Figure 5.1 visualizes an example times series of each type. A further assumption is that the data of the type electricity and district heating can be handled the same since both types are energy meters and have the same unit.

First, a representative number of sensors was selected. From every category six sensors were selected, to get a meaningful result at the performance evaluation. They were chosen based on following criteria:

- many historical records
- few anomalies and missing data
- similar behavior compared to other sensors of the same category

The selection was based on a visual inspection of the data. The sensors with the best matching on the above criteria were shortlisted. In this step, the number of sensors was reduced from approximately 200 to 60. In a second round, these 60 sensors were evaluated regarding similar time ranges and information value. The records of electricity and district heating were taken from Building 2. All other records were taken from Building 1. No sensors were selected from Building 3 because the lengths of the time series were to short. At the time of writing this thesis, only a limited number of historical

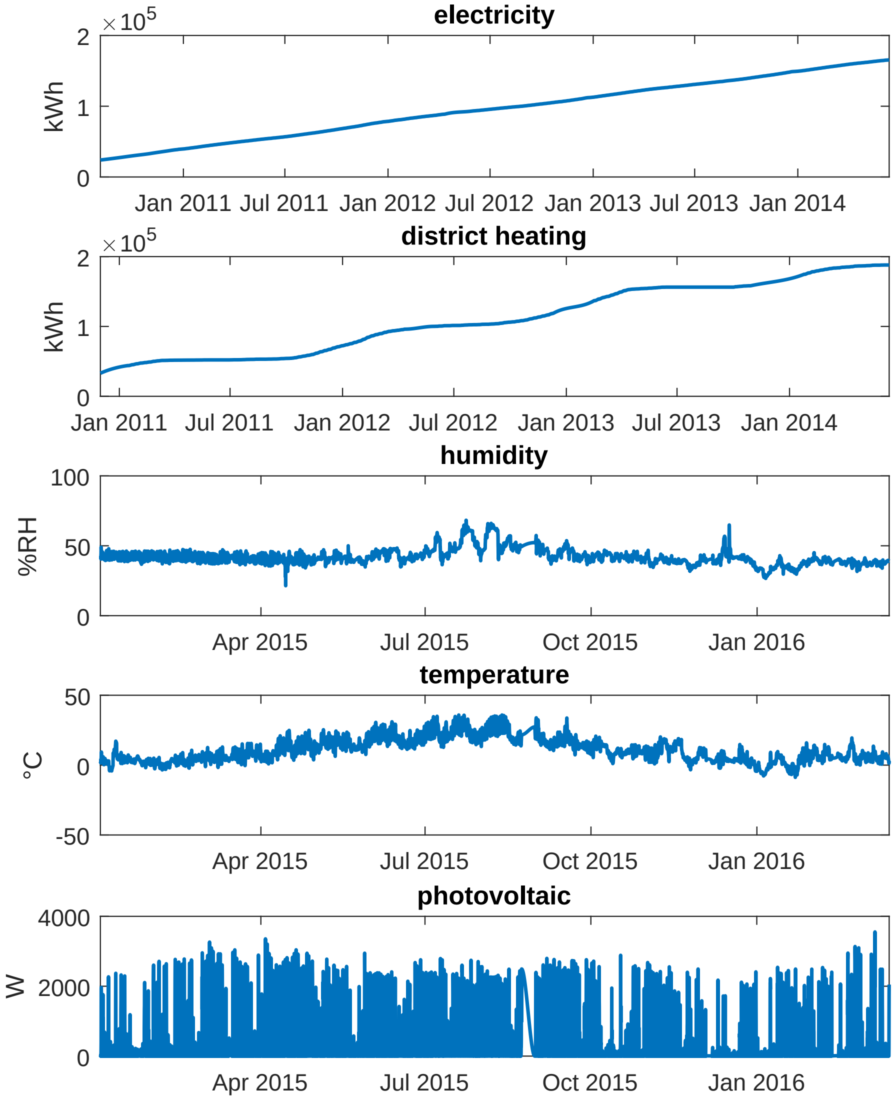


Figure 5.1: Example of the different types of data used in the thesis

data was available. The records for Building 1 started in January 2015 and ended in April 2016, and for Building 2 data ranges from January 2011 to June 2014.

The sensors of the category electricity measure the electricity consumption in kWh. There was no further information available to identify the part of the building or devices which are metered. The same holds for the district heating sensors. Only the general information of the building was available. The humidity sensors recorded the relative humidity measured as a percentage value and following additional information about the selected sensors are available:

Table 5.1: Used sensors for performance evaluation.

electricity	district heating	humidity	temperature	photovoltaic
E-1	DH-1	RH-3	PT-1	PV-1
E-2	DH-2	FH-1	PT-2	PV-2
E-3	DH-3	FH-2	OT-1	PV-3
E-4	DH-4	RH-1	RT-3	PV-4
E-5	DH-5	RH-2	RT-1-M	PV-5
E-6	DH-6	OH-1	RT-4	PV-6

RH-3 The sensor RH-3 measured the humidity of the used air at the office located in the west part of the third floor on the south side.

FH-1 The sensor FH-1 measured the inside ambient humidity on the west side of the third floor, sensor 1.

FH-2 The sensor FH-2 measured the inside ambient humidity on the west side of the third floor, sensor 2.

RH-1 The sensor RH-1 measured the humidity of the room R1, which is located on the third floor on the outer corner of the west and north side of the building. On the eastern side there is an adjoining room, and on the southern side, there is a corridor.

RH-2 The sensor RH-2 measured the humidity of the room R2, which is located on the third floor on the north side. On the west and eastern side there is an adjoining room, and on the southern side, there is a corridor.

OH-1 The sensor OH-1 monitors the outside humidity of the building.

The temperature was recorded in degree Celsius, and the following information about the sensors are available:

PT-1 The sensor PT-1 measured the temperature of a photovoltaic panel mounted at the top row in the fourth floor, and the sensor PV-1 is the corresponding electricity meter of this grid.

PT-2 The sensor PT-2 measured the temperature of a photovoltaic panel mounted at the bottom row in the fourth floor, and the sensor PV-2 is the corresponding electricity meter of this grid.

OT-1 The sensor OT-1 measured the outside temperature of the building.

RT-3 The sensor RT-3 measured the indoor ambient temperature of the third floor on the west side.

RT-1-M	The sensor RT-1-M measured the temperature of the room R1, which is located on the third floor on the outer corner of the west and north side of the building. On the eastern side there is an adjoining room, and on the southern side, there is a corridor.
RT-4	The sensor RT-4 measured the temperature of the room R2, which is located on the third floor on the north side. On the west and eastern side there is an adjoining room, and on the southern side, there is a corridor.

The photovoltaic sensors are energy meters, which record how much energy was transferred into the electricity network of the building. This corresponds to the production in Watts at one particular moment in time. All the panels are mounted at the facade of the building and not at the rooftop. There was no further information on the photovoltaic panels like the brand, size, or technology.

PV-1	The sensor PV-1 measured the actual production of an inverter connected to two strings, and both are mounted at the facade in the fourth floor on the left side.
PV-2	The sensor PV-2 measured the actual production of an inverter connected to two strings, and both are mounted at the facade in the fourth floor on the right side.
PV-3	The sensor PV-3 measured the actual production of an inverter connected to a series of strings, and the strings are mounted on a test bench. The panels are also mounted at the facade in the second floor.
PV-4	The sensor PV-4 measured the actual production of an inverter connected to a series of strings, and the strings are mounted on a test bench. The panels are also mounted at the facade in the second floor.
PV-5	The sensor PV-5 measured the actual production of an inverter connected to a series of strings, and the strings are mounted on a test bench. The panels are also mounted at the facade in the third floor.
PV-6	The sensor PV-6 measured the actual production of an inverter connected to a series of strings, and the strings are mounted on a test bench. The panels are also mounted at the facade in the third floor.

Before any prediction can be made, the selected sensors were analyzed. In the first place, the recorded data is filtered for anomalies, like spikes or invalid data. Furthermore, the data needs to be equally spaced in time. Therefore, missing sample points are reconstructed out of existing ones (interpolation), or sample points are grouped (downsampling). These steps are done by preprocessing the data.

There are different types of faults which affect the temporal behavior of a sensor. Such faults can be permanent, transient, intermittent, noise, or drift [34]. Without knowing when and if repairs have been carried out, it is not possible to assign the individual faults to a specific type. However, following anomalies were identified by visual inspection of the sensor data:

(a) the sensor value got stuck at the last valid value
(b) the sensor value was outside of possible bounds
(c) the sensor value was constantly zero
(d) no sensor value was available for the time

The anomalies of the types (a) and (c) were found with the use of the MATLAB function `findpeaks`. This function can either be used to identify sudden drops to zero as well as constant values before a drop. Furthermore, it can be configured to find points in a time series where the values are not constantly increasing. In this thesis, anomalies of the types (a), (b), and (c) were deleted such that all anomalies were reduced to type (d). Then the missing data were interpolated using a shape-preserving piecewise cubic interpolation (pchip) algorithm of MATLAB. In the same step, the historical data was resampled such that for every hour one data point was generated. As an example, Figure 5.2 shows a time series of an electricity meter. The blue crosses visualize the raw data, and the solid green line shows the time series after interpolating missing data points. There are three points highlighted with a red circle. At these points, the sensor started providing wrong data. The first two failures are short time failures, but the last one was over a long period. Here the sensor value stuck at a certain value till its fall to zero. Such failures would have a negative impact on the training of the forecasting algorithms.

After removing outliers and interpolating the missing data, the time series was tested if it is stationary, has a trend, or any seasonality pattern. Therefore, several test methods exist. In this thesis, the augmented Dickey-Fuller test was used to indicate if the time series are stationary or not. Box, Jenkins, Reinsel, *et al.* [17] proposed that it is also possible to check if the time series is stationary by calculating the autocorrelation function, the partial autocorrelation function, and plotting the time series. Table 5.2 shows the results of the augmented Dickey-Fuller test applied to the data presented in Figure 5.1. Thereby, the result 0 indicates non-stationary and 1 stationary. Not surprisingly, the data of the type electricity and district heating are non-stationary. As described in Chapter 2, stationary means that the statistical properties are invariant in time, which is violated by the sensors of the type electricity and district heating.

There are two ways to remove the trend in an unknown time series. First, a linear regression model, which describes the trend component, is fitted to the time series. This generated model can be subtracted from the initial time series, and therefore the trend is removed [5]. Another approach is to differentiate the time series, which is useful if the sensor reading is cumulative, e.g., energy meter. These two strategies can also be used to remove seasonality. For this purpose, the linear regression model has to be replaced by a

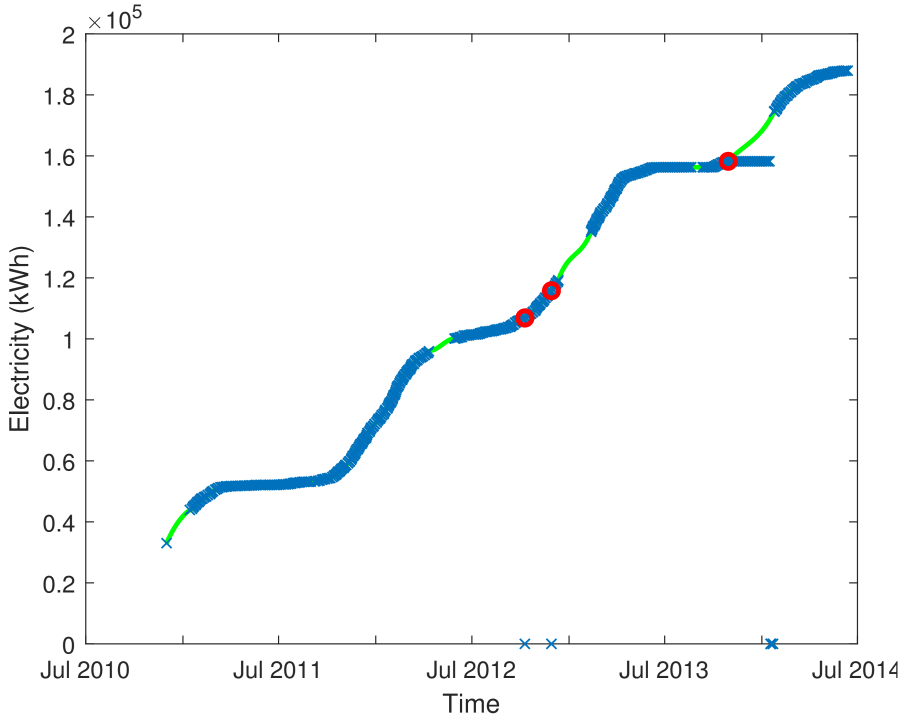


Figure 5.2: Example for invalid data and sensor faults.

Table 5.2: Results of the augmented Dickey-Fuller test applied to the sample data presented in Figure 5.1

	ADF-Test
electricity	0
district heating	0
humidity	1
temperature	1
photovoltaic	1

sine wave model which is fitted to the seasonality pattern. In the case of differentiating the time series, the seasonal difference is computed, this means, e.g., in case of a weekly seasonality, that the actual value at time t is computed by subtracting the value at time $t - 168$. By removing the trend and seasonality, the time series becomes stationary, and as mentioned before this is necessary for further computation.

5.2 Correlation analysis

Correlation of sensors is a good indicator of dependency between them. This dependency can be used to get a better understanding of the behavior of a building. For a BAS, this behavior information can be useful, for example, knowledge about the correlation of the energy consumption among the same days of the week. An optimization service in a BAS can automatically generate this correlation and adjust parameters for better performance. Furthermore, the information that two time-series correlate with each other can be used as a feature which may improve a forecast. However, sometimes data correlate, but the outcome of the correlation has no sense as mentioned by Yule [35]. He claimed that it is not uncommon that two different time series are highly correlated, but there is no physical or logical explanation why this is the case. Therefore the correlation has no sense. Furthermore, preprocessing a time series is essential so that potential trend or seasonality is recognized and appropriately handled. If this is not the case, Figure 5.3 shows exemplary the difference regarding the correlation between a time series with and without a trend. Therefore, two random AR(1) ($x_t = 20 + a_t + 0.8x_{t-1}$, $\sigma_a^2 = 2$) time series are generated, see upper left box. Below is the corresponding cross-correlation and scatter plot to visualize that these two time-series are uncorrelated. The two AR(1) time series are uncorrelated because they are generated from independent identically distributed (iid) random shocks. In the upper right box, two different trend components ($x_{1,trend} = 0.5x_1$, $x_{2,trend} = 0.8x_2 + 1$) are added to the time series. Below them, the cross-correlation and scatter plots are visualized. If the scatter-plot indicates a diagonal arrangement of the points, it indicates correlation. It can be observed that by adding a trend, the two time-series correlate at lag 0. The statement of this outcome does not provide any information, and it is not useful. It indicates only a common trend.

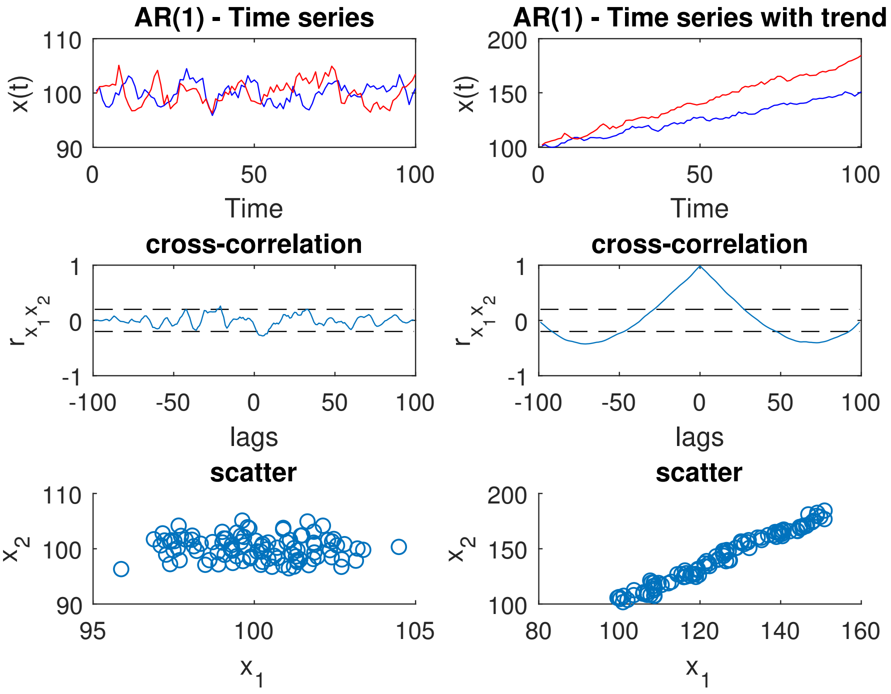


Figure 5.3: Example for misleading interpretation of the correlation.

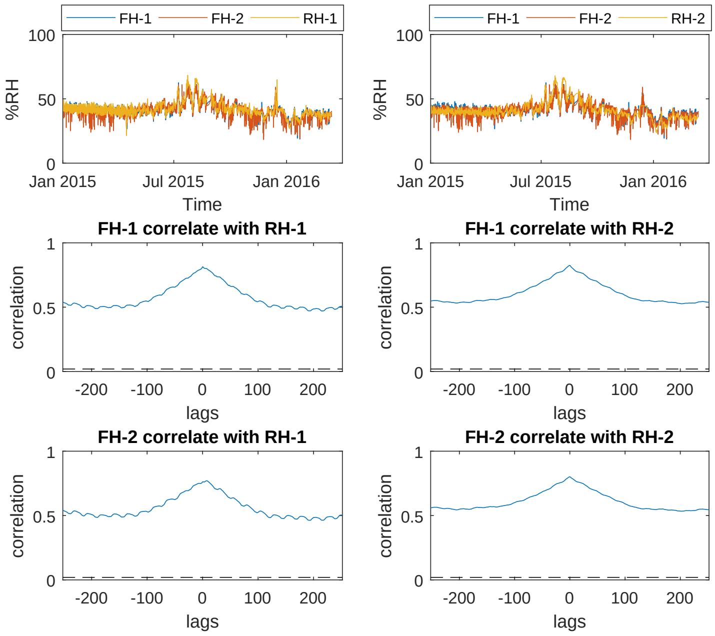


Figure 5.4: Correlation-Experiment 1

There are two important types of correlations analyzed in this thesis. First, there is the correlation between different sensors, and second, there is the correlation between different days/months/years. Using the data from *Building 1*, the correlation between different sensors was investigated. Following experiments were made to test the correlation between sensors:

- The first experiment investigates the correlation between the humidity of two distinct rooms R1 and R2 on the same floor with the general floor humidity. The general humidity is measured by two sensors, FH1 and FH2. Figure 5.4 shows in the top two charts the signals overlapped separated for the rooms R1 and R2. Without further calculation, by viewing the plot, it is evident that there is some correlation between the presented time series. The strength of the correlation is evident by the cross-correlation. In the lower four boxes of Figure 5.4, the cross-correlations are visualized. As assumed, the correlation is high, ranging from 0.76 to 0.83. This high correlation leads to the result that in the long term the humidity among different rooms behaves equally to the general humidity at the same floor.
- Experiment 2 investigates if the temperature (RT-1-middle) and the humidity (RH-1) of room R1 correlate. Figure 5.5 shows the correlation and the visualization of the two input signals. It leads to the result that the humidity and temperature are not highly correlated. The highest correlation is at lag 0 with 0.58357. Also,

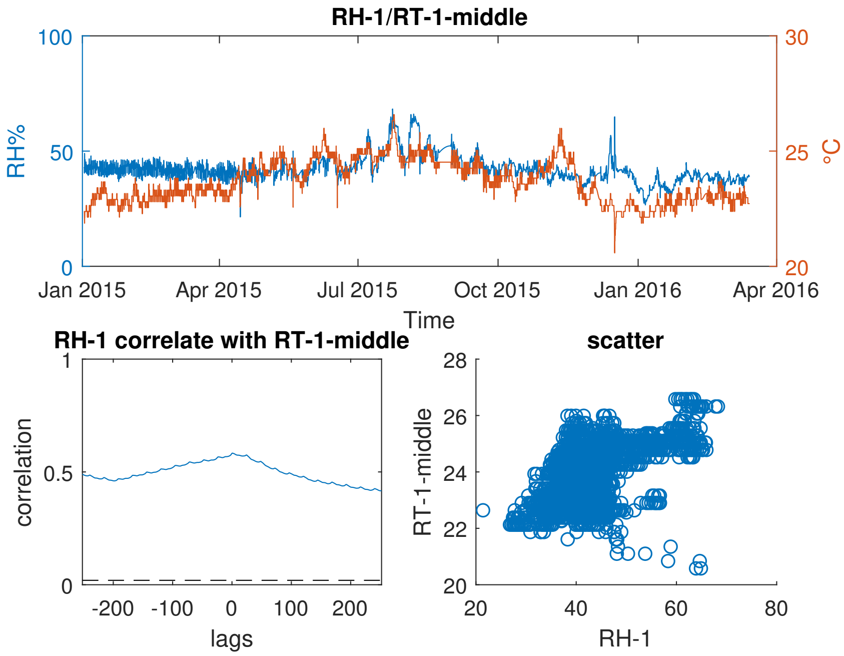


Figure 5.5: Correlation-Experiment 2

the scatter-plot of temperature and humidity shows a low correlation, as the data points do not form a straight line. This result is unexpected and can only be explained by the HVAC system of the building. In general temperature and the relative humidity have a negative correlation.

- Experiment 3 investigates if the north wall temperatures (bottom, middle, top) of room R3 (RT-2-bottom, RT-2-middle, RT-2-top) are equal in value and if there is a correlation. Figure 5.6 visualizes the three input signals in the upper box. Furthermore, it can be seen that the three signals are common in scale and magnitude. They also behave very similar over time. Both cross-correlations (middle and lower left boxes) show high correlations between the sensors. Besides the high correlation, a daily pattern is revealed.

- Experiment 4 investigates if the temperatures RT1 (north, middle and south) in the room R1 correlate. In the upper box of Figure 5.7 the three signals are visualized. The three signals are very similar in behavior. The correlation diagram shows that the temperatures correlate very high and that these time series have a seasonal pattern. The high correlation leads to the conclusion that the temperature changes are similar over time and independent of the position.

- Experiment 5 investigates if the relative outside humidity (OH-1) correlates with the outdoor temperature (OT-1). Figure 5.8 shows in the upper box, the two signals, on the bottom left side the associated cross-correlation, and on the right

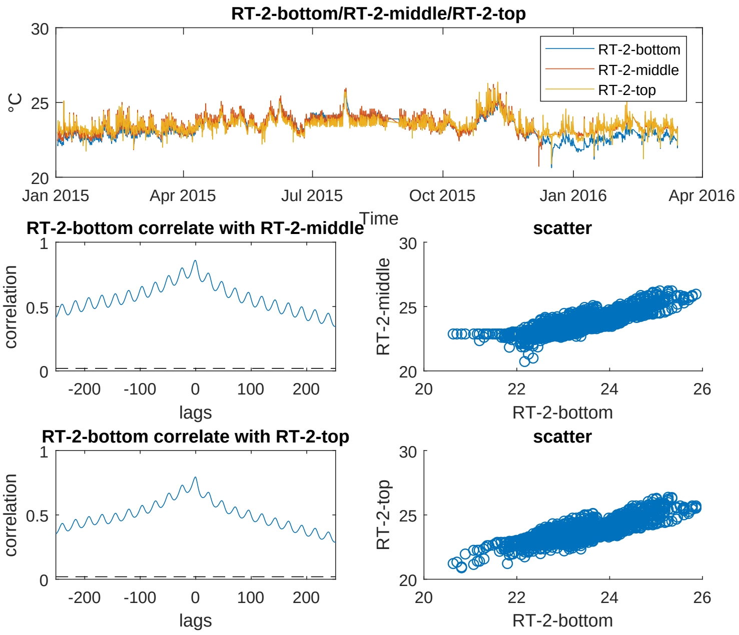


Figure 5.6: Correlation-Experiment 3

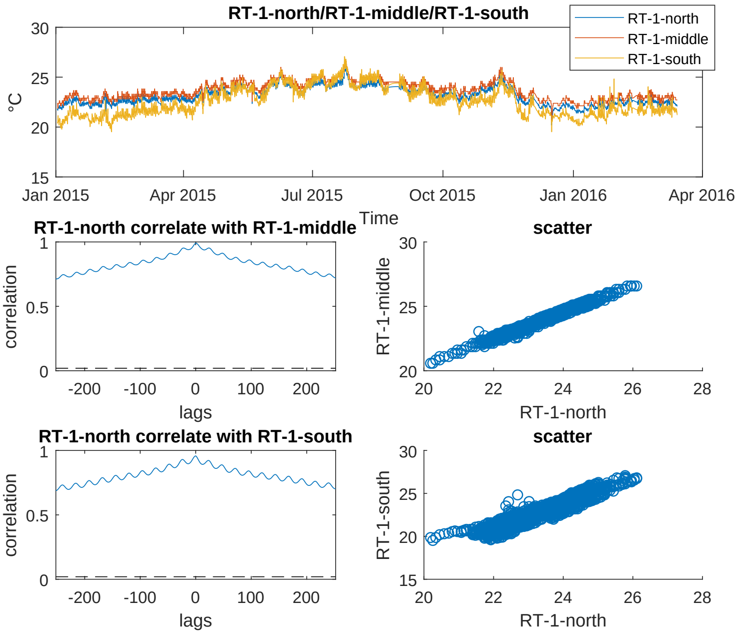


Figure 5.7: Correlation-Experiment 4

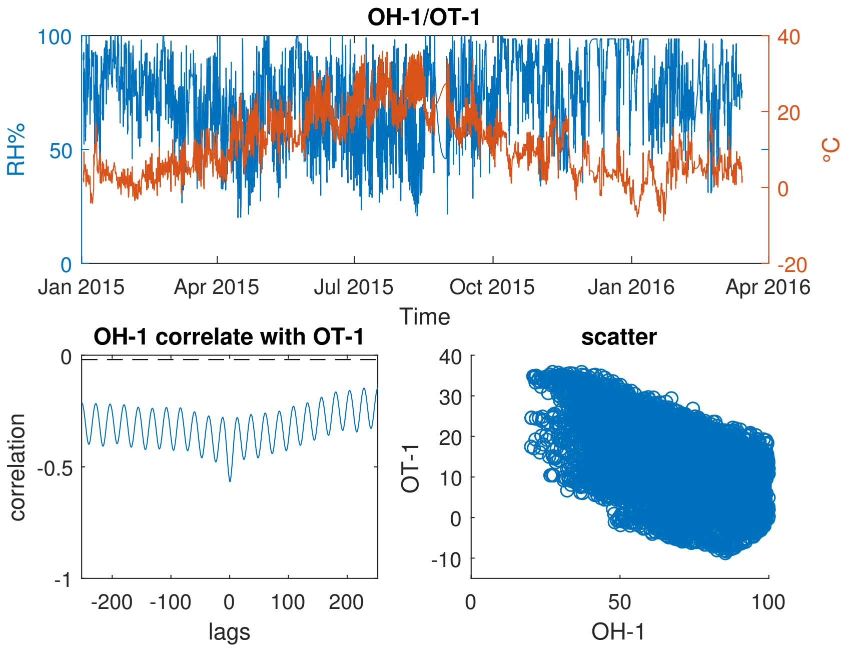


Figure 5.8: Correlation-Experiment 5

side the scatter plot. The cross-correlation shows a weak negative correlation at lag 0, which is expected. This weak negative correlation is the case because the relative humidity is decreasing with rising temperature and the other way around. Furthermore, it reveals a daily pattern, which is the result of the day-night cycle, i.e. there is a daily routine.

The results of the above correlation experiments were used to find additional features for the machine learning algorithms. For example, the temperature in room R1 should be predicted, therefore the temperature of the nearby room R2 was added to the input vector of an ANN, before training. To evaluate the trained network, the future temperature of room R2 is needed as input. Nevertheless, the future room temperature is unknown and must, therefore, be predicted. Such a forecast is not free of failures, which have a negative impact on the forecast of the temperature in room R1. It showed that this approach does not result in more accurate forecasts. The results of this correlation analysis delivered further understanding of the used data. After the calculation and visualization it was evident that nearby sensors cannot directly provide additional information to a forecast. As shortly described before, this would result in a situation were future data of the correlated sensor is needed. On a wide view, this can be done by an additional forecast, but forecast is also error prone, which will increase the error in the actual forecast.

As described before, the correlation between different days, months, and years can contain important information. Since all provided data are from office buildings, it is expectable that individual workdays will correlate, and weekends will be significantly different from weekdays. Therefore, the electricity meter E-7 from the *Building 2* dataset is chosen and plotted. Figure 5.9 shows for this meter all days of the year 2015 grouped by weekdays. Furthermore, the mean value for every weekday is calculated (solid red line) and the standard deviation (blue dashed line). As expected, the electricity consumption is relatively similar from Monday to Friday. The weekend (Saturday and Sunday) is significantly different from the weekdays. The last plot in Figure 5.9 shows the national holidays for Austria in the year 2015. In the first moment, the data look very similar to a weekday but the energy consumption is much less. If not noted in a different way, holidays are treated as weekends. For individual days, most data is within the boundaries of the standard deviation. This suggests that data, which is outside of these boundaries can either be an anomaly or a special day. This information can be used to filter out such data to improve the training process of a forecast. The next section uses this knowledge to point out meaningful features for the machine learning algorithms.

5.3 Feature selection

This section deals with the problem, how to generate or select useful features for an ANN or an SVM. The performance of both methods relies highly on the quality of the provided input features as well as on the amount of training data. In the average case, more input features or more training data will not necessarily tend to provide better results, but the computational effort will rise. Therefore, feature selection is used to find the most relevant inputs to improve the performance and the training time of an ANN or an SVM by eliminating irrelevant information. In this context, results of the correlation analysis of Section 5.2 were used. The previous section encountered sensors with correlation to other sensors. This information is conditionally useful as a feature. For the sake of completeness of this section, it is mentioned that these sensors can provide valuable input features. A good example is the outdoor temperature which influences the energy demand of the heating system. As mentioned in the previous section, the exact future outdoor temperature cannot be easily computed, due to its complexity. Therefore the minimum and maximum values are used. As described in the papers [14] and [15], the minimum and maximum outdoor temperature were used as an input feature for the neural network. These values can be obtained from a weather service and provide good accuracy.

Furthermore, it is proposed by [36] and [21] to scale or normalize the input data. Normalizing should prevent that features with a high magnitude in the value penalize other features with a much lower magnitude. Therefore, there are several algorithms for normalization available. In the case of an ANN, the right algorithm highly depends on the expected value range of the transfer function of the layers. For example, if the transfer function of the hidden-layer is *tanh*, the input value should be in the range of $[-1; 1]$ for all features. Equation 5.1 shows the min-max normalization as proposed by [36], where the feature value is mapped to the range defined by the upper bound variable b_u, and lower bound variable b_l. Crone, Guajardo, and Weber [36] concluded that scaling the features to the range of $[-0.5; 0.5]$ with a *sigmoid* activation function yields to a better

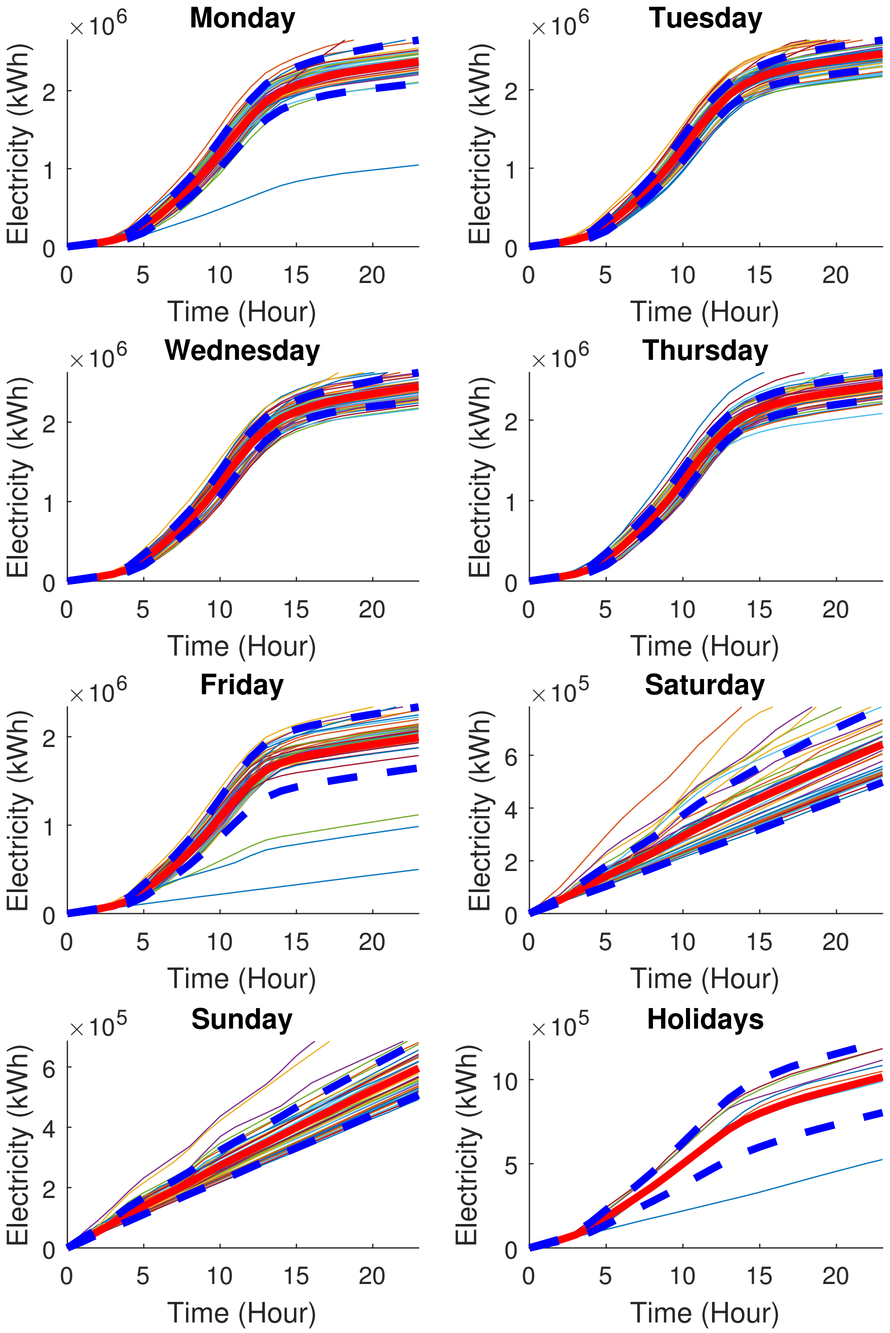


Figure 5.9: Electricity consumption grouped by the weekday for the sensor E-7 in the year 2015 from the *Building 2* dataset. The red line shows the mean of the day, and the dashed blue lines the standard deviation of the day. In the plot *Holidays*, only national holidays of Austria are included.

performance than a $[-1;1]$ scaling. Furthermore, it is vital to remove features without any information. This is the case if the values of this feature are constant. Removing a constant feature does not reduce the performance of the ANN or SVM, but it reduces the computational time for the training.

$$x_{norm,i} = b_l + \frac{x_i - x_{min}}{x_{max} - x_{min}}(b_u - b_l) \tag{5.1}$$

As described in Chapter 2, a time series is a chain of ordered time and value pairs so that the first two features are the timestamp and the value. An ANN or an SVM cannot process a timestamp because it is a complex value. As described before the input of an ANN or an SVM should be normalized to get better performance, thus the value must be numerical. If the timestamp is represented in seconds, then the resulting time series is an increasing series and would not provide any further information. Therefore, the timestamp is split into individual components, e.g., hour, minute, day, month, year, and week of the year. Except for the year component, the components are recurring so that the algorithm can learn, e.g., that every day at a specific time a pump starts, and therefore the energy consumption rises. Depending on the length of the training data, the year component is not always useful, e.g., the time series is within one year. These recurring features are also called periodic variables [24, p. 105] and can be transformed to polar coordinates. The advantage of doing this transformation is that the distance between the last and the first value is the same as for two values in between. Equation 5.2 shows the transformation, where a feature x is transformed into x_1 and x_2 while S is the total number of time slices, e.g., 24 for the feature *hour*. For better differentiation between the transformed and real-valued feature, they are marked with the word *cyclic*.

$$[x_1, x_2] = T(x) = \left[\sin\left(\frac{2\pi x}{S}\right), \cos\left(\frac{2\pi x}{S}\right)\right] \tag{5.2}$$

Furthermore, the weekdays were analyzed to identify similarities. Figure 5.9 shows that weekends and holidays behave differently then weekdays. This different behavior is represented by the feature *is holiday*, which is a boolean flag. This feature depends on the public holidays of the country and the holidays of the individual company. This thesis uses the Austrian holidays. The next feature indicates weekends as a boolean flag, it is called *is weekend*. The presented features should ensure that the algorithms will learn patterns for days, weeks or months.

Besides the timestamp, also the values have dependencies among each other. As proposed in the papers [37] and [38], it is of advantage, especially for forecasting a time series, to use not only the last value as input but also the last two or more values. Adding these features empowers the algorithm to identify sequential dependencies. Figure 5.9 visualizes that every weekday behaves the same plus some deviation. Thus, the next proposed feature is *recurrent week*. It is the value of the data point precisely 168 hours (one week) before.

All these information obtained by the correlation analysis was essential to gain a better understanding of the observed time series and is used in the next section to setup the different forecasting methods.

CHAPTER 6

Implementation and evaluation

For the implementation and evaluation, the numerical computing environment MATLAB [6] was used. The advantage of such an environment is that many algorithms are available in form of toolboxes, which can be easily adapted or reused to solve new problems. In this thesis, the Neural Network Toolbox, the Statistics and Machine Learning Toolbox, and the Econometrics Toolbox of MATLAB were used. The Econometrics Toolbox provides the ARIMA model, the Statistics and Machine Learning Toolbox provides an SVM implementation, and the Neural Network Toolbox provides an FFNN implementation with the LM training algorithm.

6.1 Performance measures

The performance of the individual algorithms is not a function of computational effort but rather a function of the similarity between the measured and predicted time series. Therefore, following error metrics were chosen to evaluate the performance of the individual forecasts:

MAE The Mean Absolute Error is a measure to quantify the distance between the forecasts and actual values so that the MAE value has the same unit as the real values. Therefore it can only be used to compare forecasts with the same unit and magnitude. The MAE is calculated by computing the mean over all sample points of the absolute difference between the predicted value and the real value. Equation 6.1 shows the formula of the MAE, where y_i is the predicted value and x_i is the real value of a sample point.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{6.1}$$

The advantage of the MAE is the fast and straightforward computation, and it is useful to compare individual forecasts on the same data set. A disadvantage is that it is not a scale-independent measure.

SMAPE The Symmetric Mean Absolute Percentage Error is the percentage of the error between forecast and real value. The SMAPE is calculated by taking the mean of the absolute error divided by the sum of the forecast and real value and scale it by 200 (see Equation 6.2). There are more than one possibilities to calculate the SMAPE, the introduced one is recommended by Hyndman and Koehler [39].

$$SMAPE = \frac{1}{n}\sum_{i=1}^{n} 200\frac{|y_i - x_i|}{x_i + y_i} \tag{6.2}$$

The advantage of the SMAPE is its scale independence. Therefore it is possible to compare the accuracy of different data sets. However, it is hard to interpret the outcome of the measure. For example, indicates a SMAPE of the value 10.2%, a good or bad accuracy? In theory the outcome of the SMAPE can be in the range of $[-\infty; \infty]$. Therefore, the SMAPE is mostly used to compare different forecasts and not judge a single forecast. The forecast with the closest value to 0 is better than the other ones.

MASE The Mean Absolute Scaled Error is a scale independent error measure to compare forecasts of different units proposed by Hyndman and Koehler [39]. Scale independent means that it can be used to compare the accuracy of forecasts from different units and magnitude, e.g., to compare the accuracy of an energy forecast with a temperature forecast. Equation 6.4 shows the formula to calculate the MASE for seasonal forecasts. The variable q_i represents the scaled in-sample MAE. This means the error $(y_i - x_i)$ is scaled by the seasonal naïve forecast (see Equation 6.3), so that it is independent of scale and magnitude. The variable s is the seasonal parameter which defines the seasonal distance between the samples. The seasonal naïve forecast approach uses the value in the same season as the next forecast value. The MASE is defined as the absolute mean of the scaled in-sample MAE (see Equation 6.4). If the value is greater than one, the forecast is worse than the naïve forecast. If the value is less than 1, the forecast is better than the naïve forecast computed on the training data.

$$q_i = \frac{y_i - x_i}{\frac{1}{n-s}\sum_{j=s+1}^{n} |x_j - x_{j-s}|} \tag{6.3}$$

$$MASE = \frac{1}{n}\sum_{i=1}^{n} |q_i| \tag{6.4}$$

6.2 Model architecture

In this theses, there are three different types of forecasting algorithms used and compared. The ARIMA model is a statistical way of modeling stationary random processes. The ANN and SVM are machine learning algorithms, which try to find patterns in the data to predict the future. Each of these algorithms has to be differently configured (as described in the Chapter 3 and Chapter 4). The next three subsections describe the process of finding a good setup for each presented algorithm. Thereby, every algorithm is separately evaluated for every category of sensors shown in Figure 5.1. For the evaluation, the time series of every category was divided into a training set and test set. The records of the last week were used as test set and the previous four weeks were used as training set. This approach should determine if it is necessary to use different configurations for every category. The test set will be used to evaluate the accuracy of the forecasts using the performance measures of Section 6.1.

6.2.1 ARIMA

The ARIMA model and the corresponding modelling process is proposed by Box, Jenkins, Reinsel, *et al.* [17]. Figure 6.1 visualizes this iterative process. The model parameters are evaluated by checking if the time series is stationary or not, and by visualizing the autocorrelation and partial autocorrelation as described in Chapter 3. The Dickey-Fuller test was used to verify if the presented time series is stationary. If the time series is not stationary, it is differentiated and then checked again. The number of iterations specify the model parameter d. The computation of the model parameter d was already done in Section 5.1. The autocorrelation and the partial autocorrelation are calculated and visualized to get a feeling of the time series and pre-assign proper model parameters. Then an iterative process started where the model is evaluated and if necessary further adjustments of the parameters are made. After every adjustment, the model is newly evaluated. This process is repeated till no further improvement of the accuracy was reached.

In order to estimate all parameters for the electricity time series from Figure 5.1, the according autocorrelation and partial autocorrelation are computed and visualized in Figure 6.2(a). It is used to estimate the parameters p, q, and the seasonality s. The parameter p specifies the order of the AR part of the time series. Therefore the partial autocorrelation was analyzed if the values cut off after some lags and stay then within the confidence interval. After one lag, the partial autocorrelation is all most within the boundaries. This behavior indicates that the parameter p is 1. In the next step, the order q of the MA part of the time series was evaluated. Therefore, the autocorrelation should cut off after a number of lags and stay then within the confidence interval. This behavior is not shown in the Figure 6.2(a), therefore it is assumed that there is no MA part in this time series and thus the parameter q is 0. After identifying the parameters p, d, and q, the time series is analyzed to find seasonal indicators. The autocorrelation shows a repeating pattern every 24 and 168 lags. The first pattern is not as strong as the

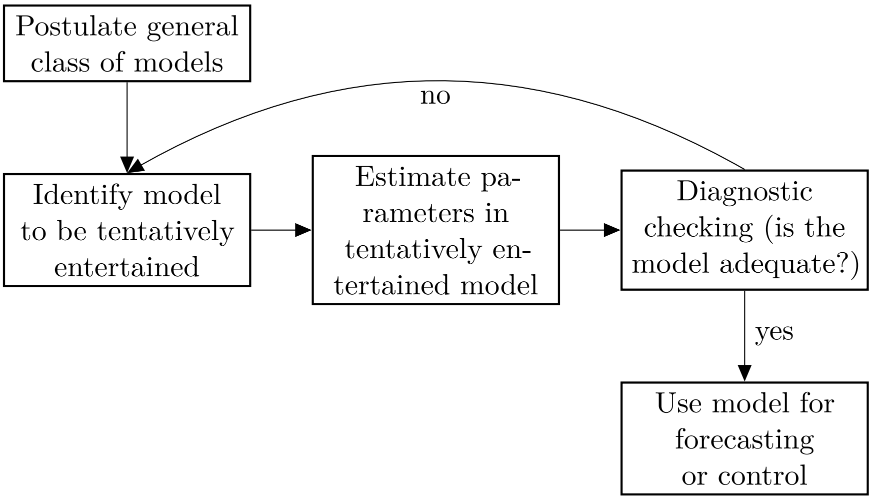

Postulate general class of models
no
Identify model to be tentatively entertained
Estimate parameters in tentatively entertained model
Diagnostic checking (is the model adequate?)
yes
Use model for forecasting or control

Figure 6.1: Iterative model design, adapted from [17, p. 16]

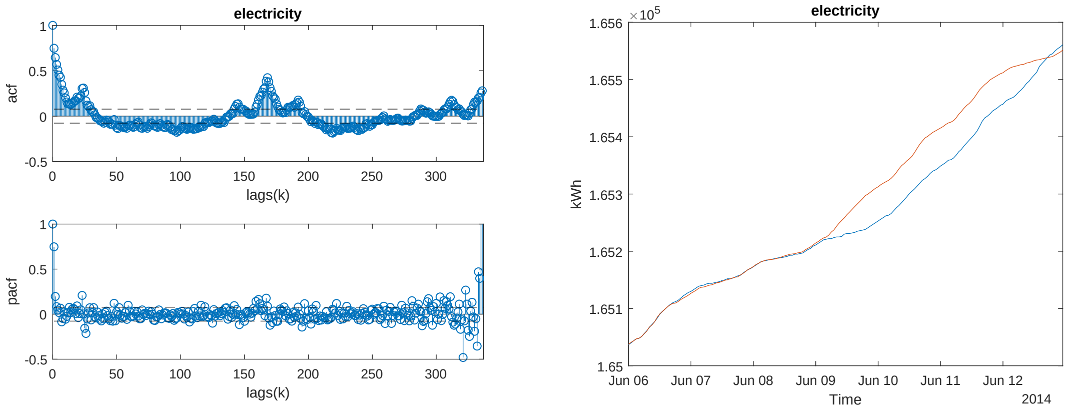
second one. This result means that there is a weak repeating pattern every 24 hours and a stronger pattern every 168 hours. The parameter s for the seasonal ARIMA model has to be evaluated if it is 24 or 168 hours. For the seasonal ARIMA model the parameter P is 1.

After all parameters have been estimated, the MATLAB function `estimate` was used to evaluate the unknown variables of the polynomial representing the $SARIMA(1,1,0) \times (1,1,0)_s$ model. Therefore, the function uses the maximum likelihood to estimate the unknown variables based on the given training set. Afterward, the next 168 hours were predicted and compared to the test set. This was done either with a seasonality of 24 hours or 168 hours. The result was that a 168 hours seasonality worked better than 24 hours. Figure 6.2(b) shows the test set (blue) and the forecast (red). The forecast accuracy was measured using the error metrics described in Section 6.1 and the following accuracy was achieved:

- MAE: 27.188
- SMAPE: 0.016
- MASE: 0.384

The results confirm the observation from Figure 6.2(b) that the model performs well. The total power consumption in the predicted week was 523.933 kWh, and the MAE is 27.188 kWh, which is an acceptable result. Also, the MASE is smaller than 1 which indicates a better prediction than the naïve forecasting approach.

This procedure was repeated for the remaining categories district heating, humidity, temperature, and photovoltaic production. Therefore, the autocorrelation and partial autocorrelation for every category was calculated as described in the previous paragraphs and visualized in Figure 6.3. After analyzing the autocorrelation and partial autocorrela-



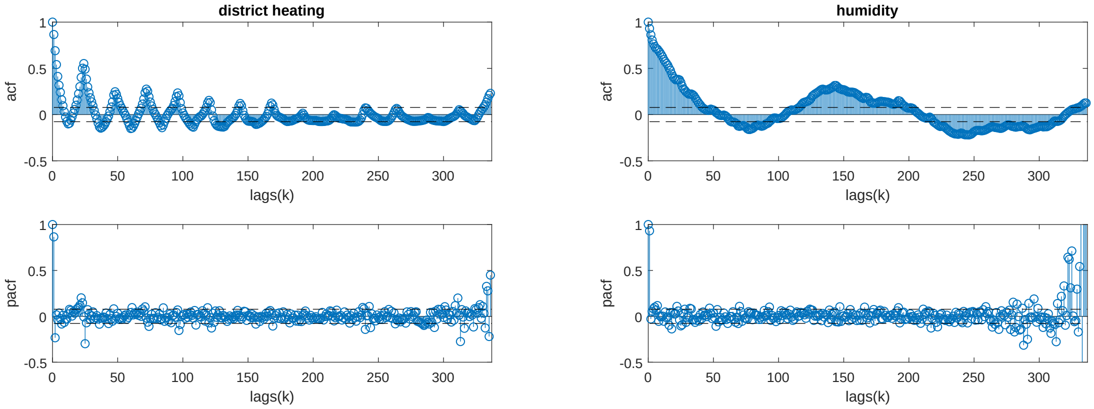
(a) Autocorrelation and partial Autocorrelation

(b) Evaluation forecast

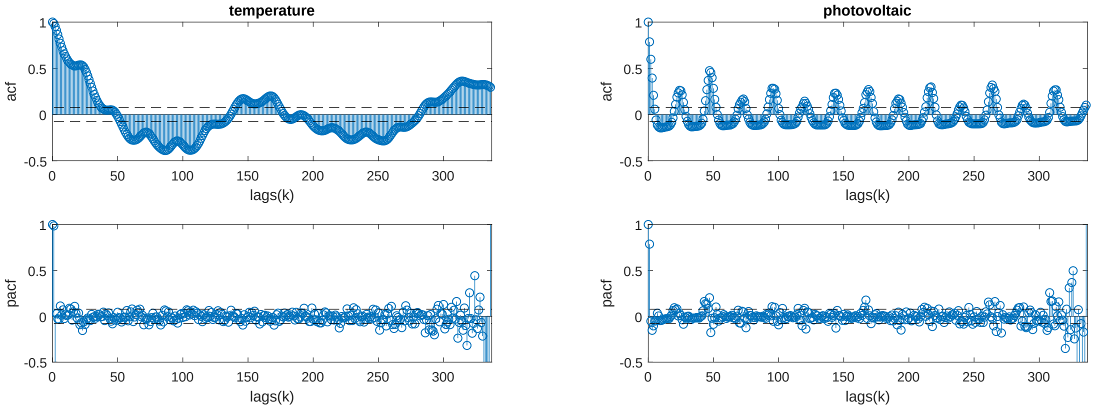
Figure 6.2: The left figure shows the autocorrelation and partial autocorrelation plot, and the right figure shows the one week evaluation forecast of the electricity time series in Figure 5.1 for the ARIMA approach.

tion of the district heating time series in Figure 6.3(a), the following parameter values were estimated: $p = 1$, $q = 0$, and $s = 24$. Contrary to the electricity, the autocorrelation for the district heating time series revealed only a 24 hour pattern. This observation indicates that the heat load of the building is mostly affected by the day-night cycle. For the category humidity, Figure 6.3(b) visualizes the autocorrelation and partial autocorrelation. It shows a nearly weekly repeating pattern as well as a weak daily pattern. After considering the result of the electricity parameter estimation where the autocorrelation showed a similar behavior, the seasonality is set to $s = 168$. Furthermore, the remaining parameters were set to $p = 1$ and $q = 0$. Analyzing the autocorrelation and partial autocorrelation of the temperature time series in Figure 6.3(c), the following parameters were estimated: $p = 2$, $q = 0$, and the seasonality $s = 24$. Figure 6.3(d) showed a very steady repeating pattern at the autocorrelation diagram, so that the seasonality was set to $s = 24$. The partial autocorrelation shows that the parameter p is 1 and due to the steady seasonal pattern the parameter P is 24.

The estimated model parameters for the categories district heating, humidity, temperature, and photovoltaic were tested by an one week forecast similar to electricity. Figure 6.4 shows the results of the forecasts and Table 6.1 shows the corresponding results of the MAE, SMAPE, and MASE error metrics. In Figure 6.4, the measured time series is blue and the predicted time series is red. Analyzing the forecast for the district heating in Figure 6.4(a) a phenomenon, which is only present in this category, was revealed. This phenomenon is a zero energy usage over a long period. Further observed time series of the same category showed, that the district heating is turned off during the summer time. Even though this event is hard to predict, it should be handled properly. Through this event, the ARIMA approach was not accurate for the category district heating.



(a) district heating (b) humidity

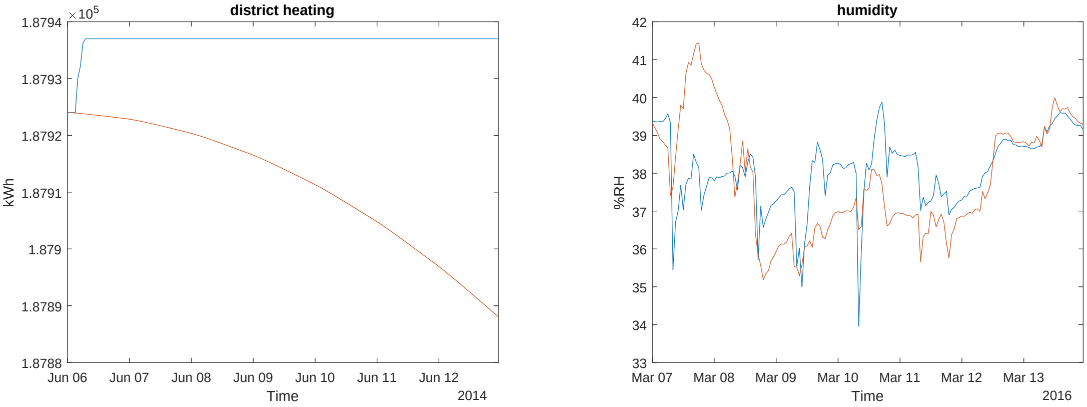


(c) temperature (d) photovoltaic

Figure 6.3: Autocorrelation and partial autocorrelation for the categories: district heating, humidity, temperature, and photovoltaic production

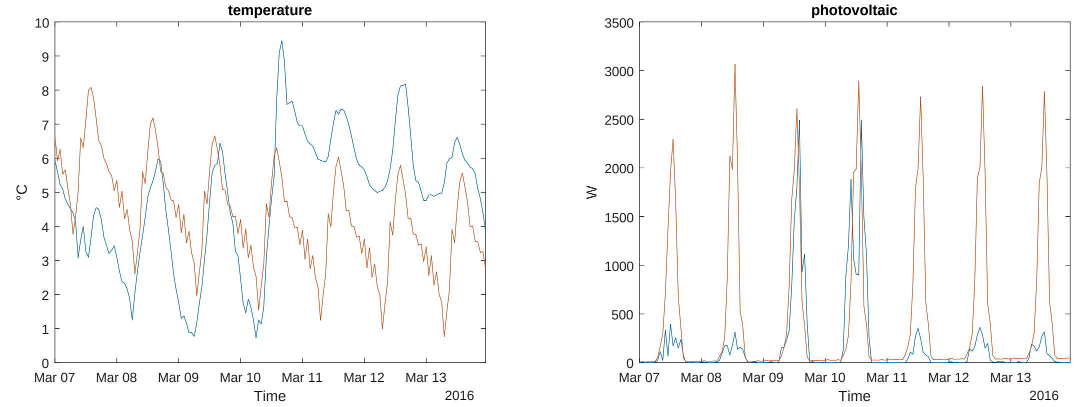
Comparing the MASE of the district heating with the MASE of the electricity forecast it is clear that the forecast is not good. Figure 6.4(b) shows a good forecast for the humidity time series. The algorithm was able to forecast the changes in the behavior of the time series mostly very well. This results were supported by the results of the error metrics in Table 6.1. Figure 6.4(c) shows that the ARIMA model can fit the day-night pattern very well, but the allover fitment was not good. Figure 6.4(d) shows a good forecast for days with normal sunshine. If the weather conditions were unstable, then the model failed.

Table 6.2 summarizes the previously estimated model parameters for every category of sensors. Surprisingly, a seasonality of 168 hours (1 week) worked better than 24 hours for the electricity and humidity time series. These parameters were used in the evaluation



(a) district heating

(b) humidity



(c) temperature

(d) photovoltaic

Figure 6.4: One week evaluation forecast for the categories: district heating, humidity, temperature, and photovoltaic production for the ARIMA approach

Table 6.1: Results of the parameter evaluation for the ARIMA approach.

Type	MAE	SMAPE	MASE
electricity	27.188	0.016	0.384
district heating	25.215	0.013	56.205
humidity	1.040	2.745	1.007
temperature	2.015	48.455	1.472
photovoltaic	371.299	141.487	1.957

Table 6.2: *SARIMA* model parameters for the categories defined in Figure 5.1.

Type	p	d	q	P	Seasonality	Q
electricity	1	1	0	1	168	0
district heating	1	1	0	0	24	0
humidity	1	0	0	0	168	0
temperature	2	0	0	0	24	0
photovoltaic	1	0	0	1	24	0

in Section 6.4 at the cross-validation.

6.2.2 Neural Network

This section describes the setup for an ANN to predict the next 168 hours of a time series. Therefore, a RNN network is used. After the correlation experiment in Section 5.2 and the evaluation of the autocorrelation in Section 6.2.1, it was clear that every category has some sort of seasonality. The repeating pattern can be used in a RNN for the feedback loop. The correlation experiments showed that individual days of consecutive weeks correlate. This means that the same weekday, e.g., Monday, follows the same schema plus/minus some deviation every week. The neural network can use this information to predict future values. Furthermore the time of the day, and the two indicators for weekend and holiday are used as input for the RNN. Figure 6.5 shows the unfolded presentation of the chosen RNN. The input *recurrent week* is the output value of the network at time $t - 168$. As mentioned by Hippert, Pedreira, and Souza [40], past values are often used as inputs. Commonly the number of hidden layers is one, and only in special cases, it is two [40]. In this thesis, one or two hidden layers with different numbers of nodes did not perform well, so that the number of hidden layers was enlarged. After testing different numbers of hidden layers, it resulted in five hidden layers with six or three nodes per layer. This layout of the hidden layers is similar to the concept of an autoencoder for ANN. The *tanh* function was chosen as activation function of the hidden layers, and for the input/output-layer a linear function was set. The output of the network is the value of the next hour. This means after training the network, every hour needs to be evaluated separately. The input vector $\boldsymbol{x}$ for the unfolded FFNN is as follows:

- *cyclic* hour
- is holiday
- is weekend
- recurrent week

Before the training is started, the bias and the initial weights need to be set. Therefore, the Nguyen-Widrow initialization algorithm was used [41]. It is a layer-based algorithm, which initializes the weights and bias based on random values, but with the constraint that every neuron of the layer is approximately evenly active. The benefit of this algorithm is that fewer neurons are wasted, and therefore the training works faster [42].

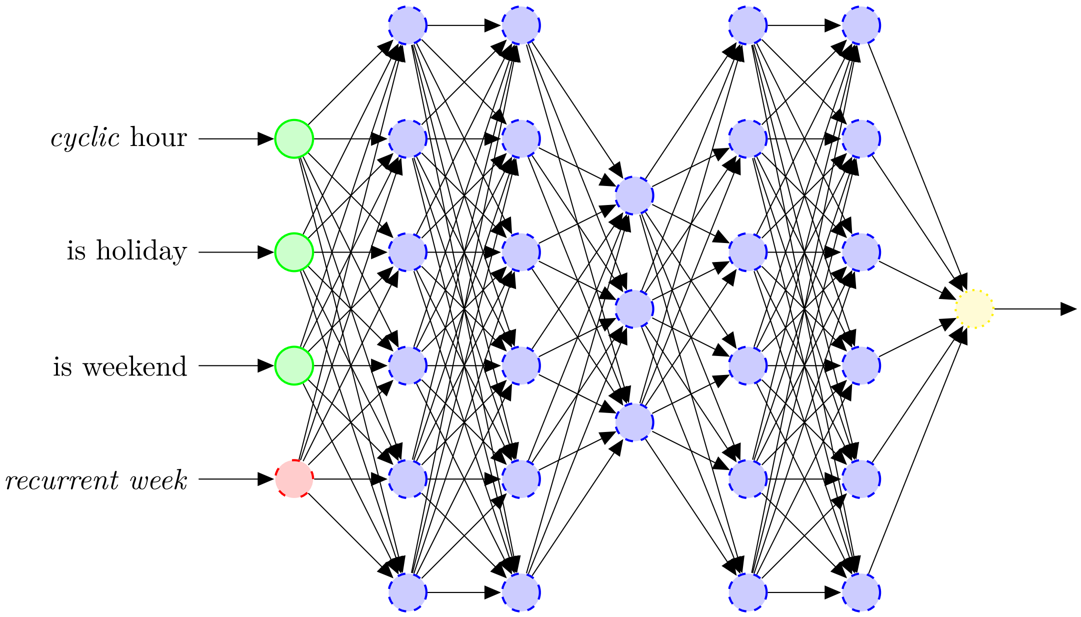
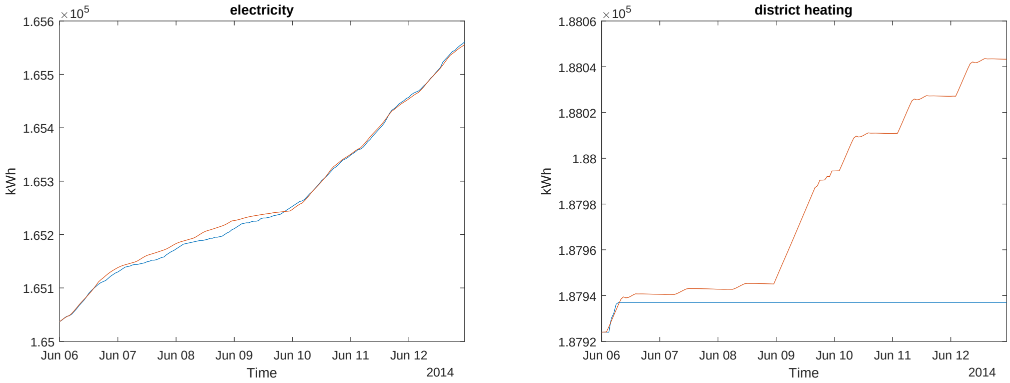


Figure 6.5: FFNN used to forecast a time series of the category electricity. There are four input nodes, 27 hidden nodes spread over five hidden layers, and one output node.

The neural network is trained with the LM algorithm as described in Section 4.1.2. For this reason, the previously described training set is divided randomly into following sets: the LM training set, the LM test set and LM validation set. The ratio between these three sets is 75% for LM training set, 15% for LM test set and 15% for LM validation set. The LM training set and LM test set are used to train the neural network, whereby the LM validation set is used to measure the generalization of the neural network and stops the training if the generalization is not getting better, or 1000 iterations are reached. The LM test set is used to calculate the network performance during and after the training. This network performance is calculated with sum of squared error (SSE) formula.

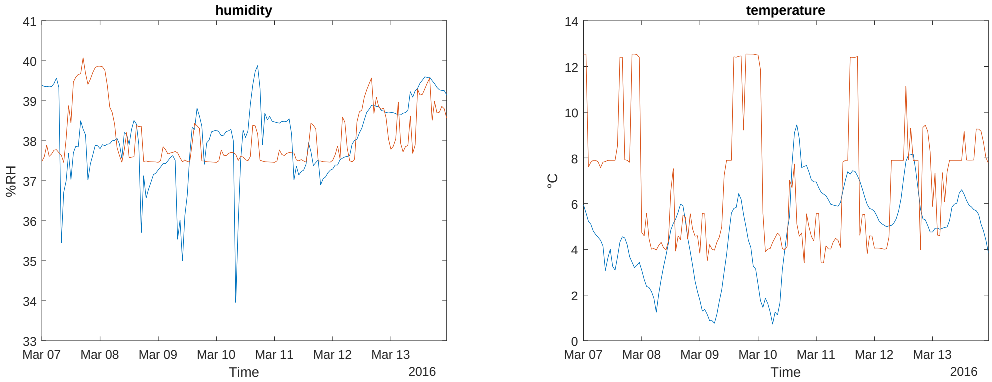
As in the previous section, this setup is applied to the sensors of the categories electricity, district heating, humidity, temperature, and photovoltaic production. The FFNN got trained, and afterwards, the next 168 hours were simulated. Figure 6.6 shows the outcome of the simulation (red line) versus the real measured values (blue line) for the categories: electricity, district heating, humidity, temperature, and photovoltaic production. Table 6.3 shows the corresponding accuracy measures.

This approach was more accurate for the categories electricity, humidity, and photovoltaic than the ARIMA approach. Like the ARIMA approach also the ANN has problems to predict the transition from summer to winter for the district heating. Therefore the forecast was useless.



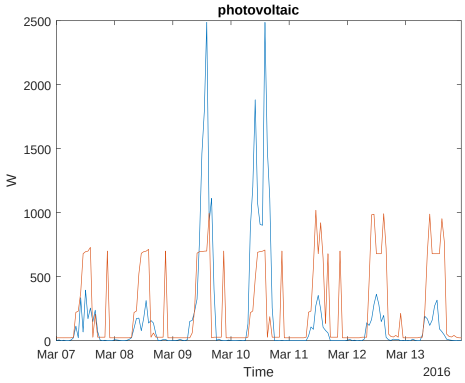
(a) electricity

(b) district heating



(c) humidity

(d) temperature



(e) photovoltaic

Figure 6.6: Measured (blue) and predicted (red) time series for the categories electricity, district heating, humidity, temperature, and photovoltaic production for the ANN approach

Table 6.3: Results of the ANN approach.

Type	MAE	SMAPE	MASE
electricity	5.916	0.003	0.083
district heating	44.248	0.023	98.630
humidity	0.803	2.109	0.777
temperature	3.026	53.228	2.212
photovoltaic	230.062	141.744	1.213

6.2.3 Support Vector Machine

In this section, the setup for the SVM to solve a regression task is explained. An SVM which solves regression Tasks is called SVR. As the ANN approach, the SVR is also a machine learning approach. Therefore, it also uses an input vector x of features, which in the best case provides all information so that the SVR can predict the future values of a time series. Therefore, all findings of the previous sections regarding the feature selection and analysis of the autocorrelation and partial autocorrelation were used. As described in Section 5.3 a single historical data point could be split into different features. There are two main components the time and the actual value. As for the ANN, the SVR uses the data of the week before to enhance the accuracy of the forecast, and this feature was called recurrent week. The following feature were used:

- *cyclic* hour
- is holiday
- is weekend
- recurrent week

This feature combination was derived from the autocorrelation and the correlation experiments in Section 5.2. The output of the SVR is the predicted value for the given time.

The ϵ-SVR model was chosen, because it is suitable for regression tasks, as mentioned in Section 4.2, thus also for forecasting. The ϵ-SVR is configured to use a linear kernel, and this is similar to an AR model [43]. The parameters C was set to 1 and the parameter ϵ is estimated by a tenth of the standard deviation using the interquartile range of the response variable [44]. The SMO solver was chosen, because of the fast calculation and low resource requirements. To ensure that the solver terminates after a finite time, the number of iterations was fixed to 1000. Fixing the number of iterations ensures that the training time is bound to an acceptable level.

As in the previous sections, the SVR is applied to the sensors of the categories electricity, district heating, humidity, temperature, and photovoltaic production. Therefore, the MATLAB function `fitrsvm` was used to generate the SVR model and to apply the SMO solver. After the model was generated, the MATLAB function `predict` was used to predict the future values. Figure 6.7 shows the estimated and the measured values for a one week (168 hours) forecast of the different categories, and Table 6.4 presents the according results for the accuracy measures. Comparing the results to the ARIMA and ANN approach, it shows that the SVR performed good for district heating, humidity, temperature, and photovoltaic. Especially, the humidity forecast in Figure 6.7(c) shows a good prediction of the time series behavior.

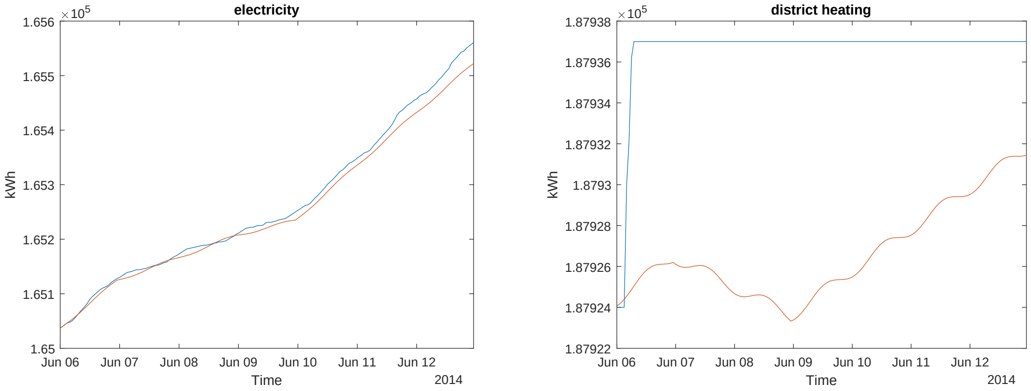
Table 6.4: Results of the SVM approach.

Type	MAE	SMAPE	MASE
electricity	11.751	0.007	0.166
district heating	9.980	0.005	22.245
humidity	0.848	2.230	0.821
temperature	1.539	36.379	1.125
photovoltaic	157.651	12.474	0.831

6.3 Evaluation method

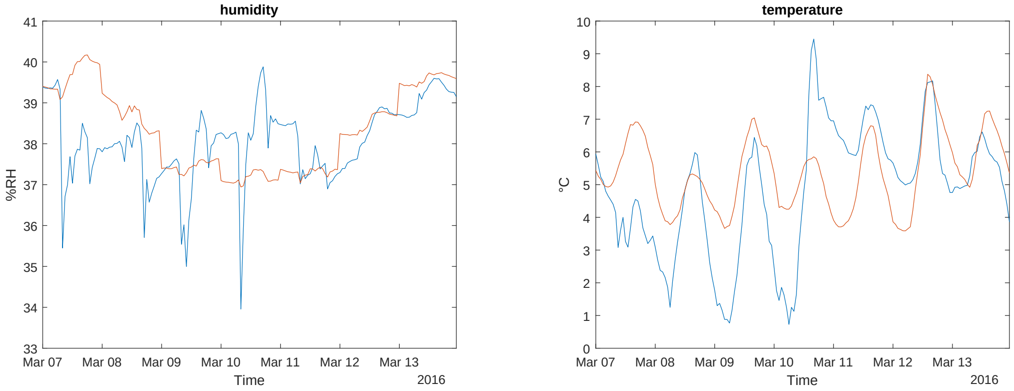
The previously developed forecasting models, i.e., ARIMA, ANN, and SVR were all tested with the same data so that it ends in a comprehensive performance test. Therefore, the 30 sensors from Section 5.1 were used. The algorithms were configured as described in Section 6.2. Every algorithm was evaluated multiples times on the same data set to ensure that the random components of the algorithms were discovered and visualized. This procedure is essential to make a statement of the repeatability of an algorithm. In the field of building automation, the time horizon of a forecast is between several hours up to a week [8] [14] [15]. Therefore, a forecast horizon of one week was used to evaluate the presented algorithms. This long forecast horizon should ensure that the developed forecasting models can be used in any building-related task, which requires predicted sensor data.

Mandal, Senjyu, Urasaki, *et al.* proposed to evaluate a forecasting model at different seasons of the year to ensure better accuracy [15]. Depending on the season of the year the behavior of the measurand is different. The forecasting algorithms are tested if they work regardless of the season in a year by choosing different time frames, e.g., spring, summer, fall, and winter. This procedure should result in a cross-validation where, in the best case, all seasons of a year are covered. The goal of the cross-validation was to show that regardless of the season in a year the forecasting algorithm provides the same performance. Since all selected sensors are located in Vienna, Austria, the seasons of the year were the same for all sensors. They vary only from year to year. So that for every season two timeframes were chosen, the first in the middle of the season and the second on the transition to the following season. Regardless of the season, every timeframe was split into two parts, the trainings set, and the test set, so that both sets are disjoint. This approach is essential to ensure that the algorithms have no prior knowledge of the forecast. The training set is used to train the network or to estimate the model parameters, and the test set is used to evaluate the forecast accuracy [20]. The length of



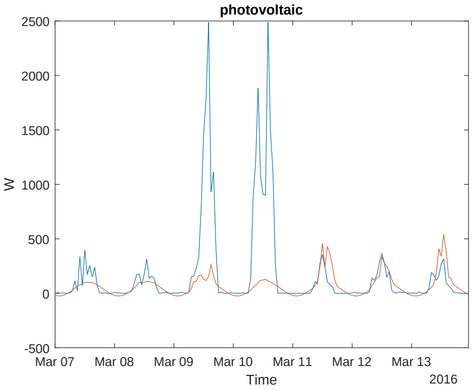
(a) electricity

(b) district heating



(c) humidity

(d) temperature



(e) photovoltaic

Figure 6.7: Measured (blue) and predicted (red) time series for the categories electricity, district heating, humidity, temperature, and photovoltaic production for the SVM approach

the training set was one season (three months) and for the test set one week (seven days). The granularity of the training set and test set was set to one hour. The test set was only used to evaluate the performance of the algorithms. Table 6.5 shows the selected periods of time. The timeframes were oriented on the meteorological seasons. So that the months December, January, and February represent the winter season, March to May the spring, June to August the summer season and September to November autumn.

Table 6.5: timeframes for the forecasting algorithm evaluation.

	Training		Test	
Type	Start	End	Start	End
Electricity District heating	01.12.2013	28.02.2014	01.03.2014	07.03.2014
	01.03.2014	31.05.2014	01.06.2014	07.06.2014
	01.06.2013	31.08.2013	01.09.2013	07.09.2013
	01.09.2013	30.11.2013	01.12.2013	07.12.2013
	15.10.2013	15.01.2014	16.01.2014	22.01.2014
	15.01.2014	15.04.2014	16.04.2014	22.04.2014
	15.04.2013	14.07.2013	16.07.2013	22.07.2013
	15.07.2013	15.10.2013	16.10.2013	22.10.2013
Humidity Temperature Photovoltaic	02.01.2015	28.02.2015	01.03.2015	07.03.2015
	01.03.2015	31.05.2015	01.06.2015	07.06.2015
	01.06.2015	31.08.2015	01.09.2015	07.09.2015
	01.09.2015	30.11.2015	01.12.2015	07.12.2015
	15.01.2015	15.04.2015	16.04.2015	22.04.2015
	15.04.2015	14.07.2015	16.07.2015	22.07.2015
	15.07.2015	15.10.2015	16.10.2015	22.10.2015
	15.10.2015	15.01.2016	16.01.2016	22.01.2016

The ANN uses a random initialization algorithm to set the initial weights before it is trained. Depending on the weights the performance of the algorithm is not the same. For reproducibility, the random seed could be fixed, so that every time the initialization algorithm computes the same values. This procedure will not ensure that the optimal weights were selected. For this reason, the random seed was not fixed, and the ANN was executed several times in a row, and then the average output was computed. After some testing, the number of retries was fixed to five. With this number, the mean output of the ANN was nearly reproducible.

6.4 Results

As described in the previous section, all three algorithms were tested for the timeframes defined in Table 6.5 to show the accuracy of the algorithm regardless of the time. For better readability, the results were divided into individual parts and then presented. The full detailed results of the evaluation can be found in the Appendix A. The results where split into three tables, the MAE results (Table A.1), the SMAPE results (Table A.2), and the MASE results (Table A.3). The seasons with a * indicate a mid-season forecast and otherwise the end of the season. Furthermore, the best algorithm with the smallest error and therefore the best accuracy for a specific timeframe is marked light grey and called a win. An algorithm is then superior to another one if in the majority of all timeframes the algorithm has the smallest MAE and therefore the most wins. If two algorithms had the same amount of wins, then both were counted. Table 6.6 shows the counted number of wins per sensor type and algorithm.

Table 6.6: Overall results of the forecast accuracy, counted from the MAE result table (see Table A.1)

	ARIMA	ANN	SVM
Electricity	3	4	2
District heating	4	6	0
Humidity	0	6	0
Temperature	6	0	0
Phovotoltaic	1	5	1
Total	14	**21**	3

The most accurate algorithm was the ANN. It was in 21 timeframes superior or equal to the other algorithms. After analyzing the results, it turned out that not a single algorithm was superior in all seasons. Furthermore, the result table showed that the SMAPE error metric resulted for the types electricity and district heating in unexpected small values compared to the other types. The next three sections explain the detailed results for every algorithm.

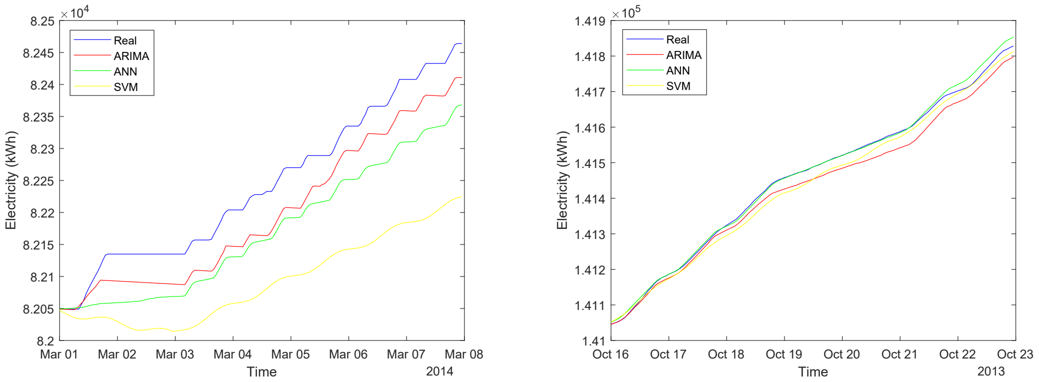
Electricity

The results in Table 6.6 showed that the ANN and the ARIMA algorithm were both superior for this category. On a close look at the results in Table A.1 for the sensors of the type Electricity (E-1 to E-6), it can be seen that the forecast for the meters E-1 to E-4 in the season summer is worse than in the rest of the seasons. Mainly, the ARIMA algorithm performed very poorly, the other two algorithms performed better but not as good as in the other seasons. Further investigations have shown that in the case of the summer season, either in season and at the end, the sensor had a malfunction. Due to the malfunction, no real data was available so that the sensor data was interpolated, as described in Section 5.1, and therefore the results are not accurate. Table 6.7 shows the mean MAE over all seasons for the electricity sensors.

Surprisingly, for the majority of the electricity sensors, the ANN algorithm had the smallest mean MAE grey marked values (see Table 6.7). Comparing this result to Table 6.6, it shows that in the mean the ANN algorithm is more accurate than the ARIMA although the ARIMA and ANN had nearly the same amount of counted wins. Figure 6.8 shows two forecasts. On the right side a very accurate forecast with a MAE of 6.7 for the ANN algorithm, and on the left side a bad forecast with a MAE of 72.5. From the

Table 6.7: Mean MAE of the Electricity type of sensors for all seasons.

	E-1	E-2	E-3	E-4	E-5	E-6
ARIMA	38.96	55.65	112.14	287.12	36.56	4.98
ANN	33.41	33.45	89.67	275.81	36.08	2.96
SVM	23.17	32.40	153.76	340.09	128.31	8.08



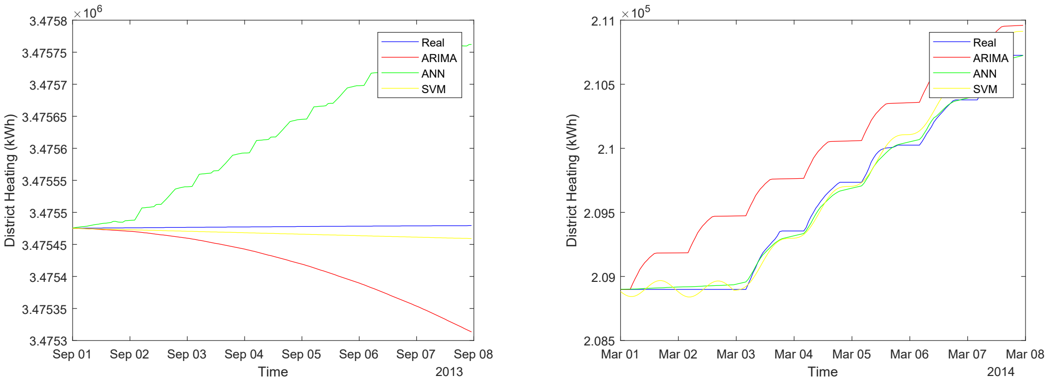
(a) High MAE (b) Low MAE

Figure 6.8: Electricity forecasts for the sensor E-5 (left) in the season Winter and E-2 (right) in the season Autumn*.

figure, it can be concluded that even the bad forecast reveal information about the future electricity behavior. The knowledge of the future behavior can be sufficient for further processing. Looking at the results in Table A.2 and Table A.3 for the sensors E-1 to E-6, it can be seen that the SMAPE and the MASE result leads to the same interpretation as the MAE.

District heating

As by the sensors of category electricity, both the ARIMA and the ANN algorithm were superior and led to a good result in a majority of sensors (see in Table 6.6). Furthermore, a big difference between electricity and district heating is the fact that in the summer the district heating is not used. Therefore, during the summer time, no consumption was monitored so that the meter value was zero (see Figure 6.9(a)). The zero values caused infinite values in Table A.3, especially for the season summer. Looking at Equation 6.3 it can be seen that if the measured value $x_j = 0, \forall j$, then the divisor is zero, and this results in a division by zero. Matlab interprets a division by zero as infinite. This means the MASE error metric is not useful in the case of periods with zero values. The heating season in Austria usually starts in autumn. Table A.1 shows that the district heating sensors (DH-1 to DH-6) did not perform well in the season Autumn. In average the absolute MAE of the district heating sensors were higher than electricity sensors. Table



(a) High MAE (b) Low MAE

Figure 6.9: Direct heating forecasts for the sensor DH-3 (left) in the season Summer and DH-5 (right) in the season Winter.

6.8 shows the mean MAE calculated over all seasons for every sensor of the category district heating. It can be seen that the ANN algorithm achieved the smallest mean MAE for the majority of the sensors. Comparing this result with Table 6.6 it is clear that the ANN algorithm is superior. Table A.2 shows the SMAPE results for the sensors DH-1 to DH-6, the results were similar regarding wins per timeframe as for the MAE results.

Table 6.8: Mean MAE of the District Heating type of sensors.

	DH-1	DH-2	DH-3	DH-4	DH-5	DH-6
ARIMA	116.97	559.36	2548.10	66.09	273.01	78.31
ANN	133.48	364.14	1462.5	146.97	90.40	113.15
SVM	249.40	797.21	3214.2	321.70	316.98	254.77

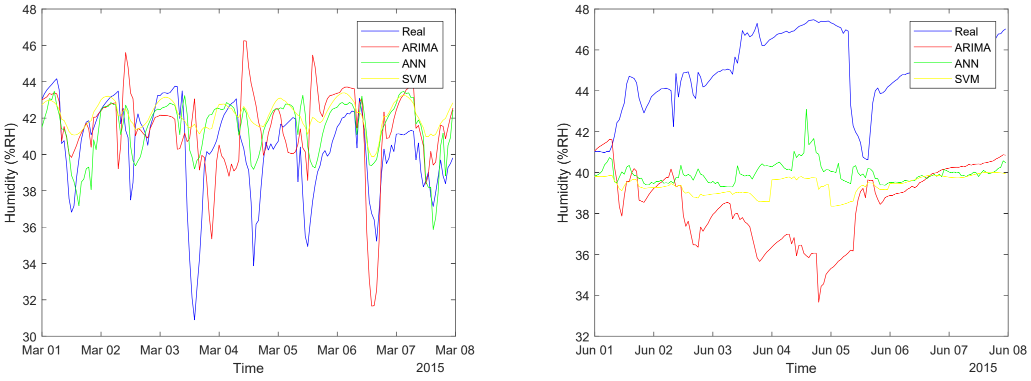
Humidity

The results in Table 6.6 for the humidity sensors show that the ANN algorithm performed the best. Comparing the MAE values in Table A.1, it is shown for the humidity sensors that the results were very close together. By further investigation of the MAE results, it unveiled a poor performance in the seasons Spring and Autumn. The MASE results in Table A.3 also showed a bad accuracy in the season Spring, Summer, and Autumn. Despite the small MAE values for the humidity sensors, the performance was not accurate. Figure 6.10 shows the best and the worst forecasts for the humidity sensors. Even the best forecast is not accurate enough for further use. Table 6.9 shows the mean MAE over all seasons for every humidity forecast.

It can be concluded, that additional features are necessary to forecast the humidity in order to improve the accuracy. For indoor humidity sensors, the number of persons

Table 6.9: Mean MAE of the Humidity type of sensors.

	RH-3	FH-1	FH-2	RH-1	RH-2	OH-1
ARIMA	4.50	3.57	3.64	3.83	4.00	16.81
ANN	3.82	3.19	3.17	3.52	4.07	13.88
SVM	3.80	3.22	3.34	3.81	4.34	14.12



(a) Low MAE

(b) High MAE

Figure 6.10: Humidity forecasts for the sensor FH-1 (left) in the season Winter and RH-2 (right) in the season Spring.

in the room is a significant feature. In the case of outdoor humidity, there are better forecasting models available.

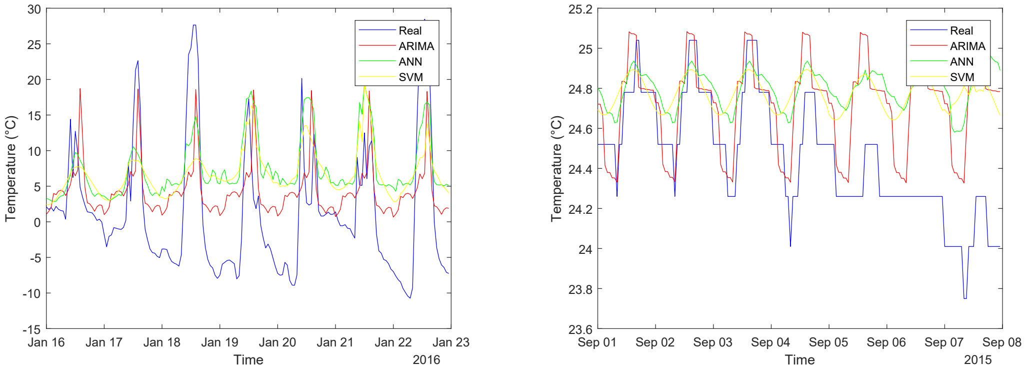
Temperature

For the sensors of category temperature, Table 6.6 indicates that the ARIMA algorithm performed the best. Analyzing Table A.1 and Table 6.10, it is shown that the sensors PT-1, PT-2, and OT-1 had a significant deviation between the forecast value and the measured value (see Figure 6.11(a)). These sensors are mainly influenced by the weather conditions. For further research, the forecasts for the sensors PT-1 and PT-2 should be enhanced by a sun position model to get better accuracy. The sensors RT-3, RT-1-M, and RT-4 had small MAE values, but as for the sensors of the category humidity, this did not indicate a reasonable accuracy (see Figure 6.11(b)). Comparing the MAE results with the MASE results in Table A.3 it showed that most of the result values were greater than one.

The results for the ANN and SVM indicate that the selected features are not proper. It is assumed that feature *recurrent week* is not sufficient. This assumption means that two successive weeks highly deviate from each other, and therefore the forecast gets inaccurate.

Table 6.10: Mean MAE of the Temperature type of sensors.

	PT-1	PT-2	OT-1	RT-3	RT-1-M	RT-4
ARIMA	6.72	6.74	5.45	1.17	0.29	0.32
ANN	8.28	8.63	6.55	1.44	0.45	0.37
SVM	8.35	8.26	6.13	1.54	0.44	0.38



(a) High MAE

(b) Low MAE

Figure 6.11: Temperature forecasts for the sensor PT-1 (left) in the season Autumn* and RT-1-M (right) in the season Summer.

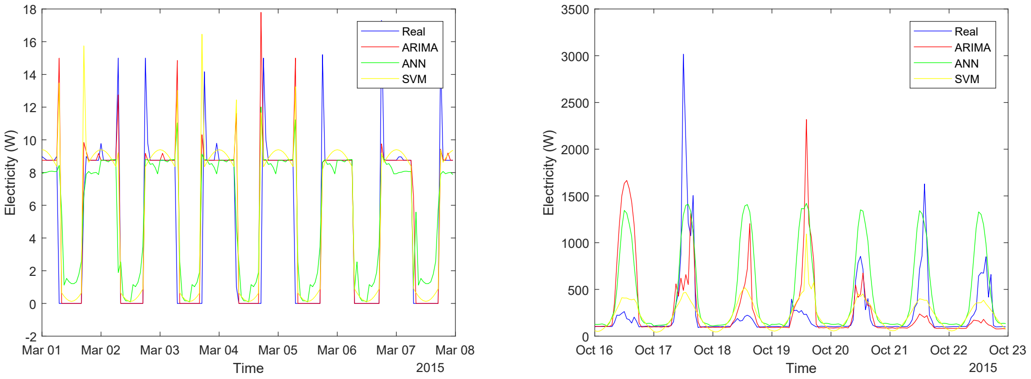
Photovoltaic

By analyzing the prediction accuracy of the sensors within the category photovoltaic, it was possible to determine that the ANN algorithm worked the best, but the overall performance was not good. Table 6.11 shows that the mean MAE over all timeframes is in the range from 2 to 500 Watts. This result was not very satisfying only the sensor PV-3 had accurate forecasts. Table A.1 shows that for the sensor PV-3, the MAE was in the best case in the range between 0.8 and 3.8 Watts. Table A.3 shows the MASE results of sensor PV-3. It can be seen that the majority of the timeframes is greater than 1, and this indicates a bad accuracy. Figure 6.12(a) shows the forecast of PV-3 for the Winter season, despite of a MASE greater 1 the forecast showed a good approximation. Analyzing the results of the sensors PV-1, PV-2, PV-4, PV-5, and PV-6 it was evident that due to changing weather conditions the output of the photovoltaic panels was not constant over time. This inconsistency cannot be predicted by the forecasting algorithms so that the accuracy decreases. Figure 6.12 shows at the left figure the forecast of the sensor PV-3 for the season Winter and on the right the sensor PV-4 for the season Summer. The figure shows that if the energy production is stable over a period than the algorithms predict the photovoltaic production well. If the energy production is unstable, the forecasts are inaccurate.

These variations imply that further features are necessary to raise the forecast

Table 6.11: Mean MAE of the Photovoltaic type of sensors.

	PV-1	PV-2	PV-3	PV-4	PV-5	PV-6
ARIMA	490.89	193.00	3.32	302.99	359.67	371.94
ANN	357.69	182.09	2.45	295.96	330.87	312.38
SVM	362.48	186.09	2.40	305.72	319.30	295.90



(a) Consistent PV production.

(b) Inconsistent PV production.

Figure 6.12: Photovoltaic production forecasts for the sensor PV-3 (left) in the season Winter and PV-4 (right) in the season Summer.

accuracy. As stated before, the photovoltaic production depends on the solar radiation, and therefore the sun position and the cloudiness are the main factors of influence.

6.5 Implementation

The last part of this thesis provides a library for easy integration of a forecasting service into third-party applications. Such applications can be a BAS where a computer controls the building. Furthermore, to avoid the high license cost of MATLAB the library was implemented in JAVA. The library is built with the JDK 1.8 and the Encog Machine Learning library [45] in version 3.3.0. The Encog Machine Learning library was chosen because of the easy to use API and the ability to set up a custom configuration of an FFNN and train it with the Levenberg-Marquardt Algorithm. The library is divided into distinct namespaces, namely

- Pps.NeuralNetwork
- Pps.NeuralNetwork.Data
- Pps.NeuralNetwork.Features
- Pps.NeuralNetwork.Norm

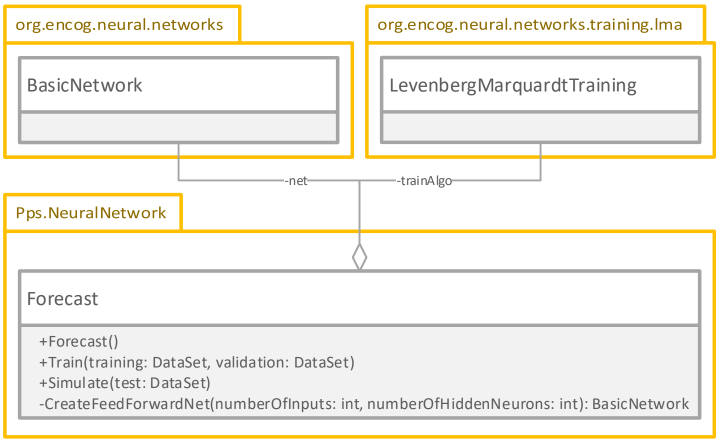

org.encog.neural.networks
BasicNetwork
org.encog.neural.networks.training.lma
LevenbergMarquardtTraining
-net
-trainAlgo
Pps.NeuralNetwork
Forecast
+Forecast()
+Train(training: DataSet, validation: DataSet)
+Simulate(test: DataSet)
-CreateFeedForwardNet(numberOfInputs: int, numberOfHiddenNeurons: int): BasicNetwork

Figure 6.13: UML class diagram for the namespace Pps.NeuralNetwork

This partitioning of the namespaces ensures that it is easily extendable without the need of recompiling. Therefore, new features can be created by deriving the abstract class `Feature`. Likewise, the abstract class `Normalizer` can easily be derived and adapted for new normalizers.

Pps.NeuralNetwork

The namespace Pps.NeuralNetwork is the interface between the Encog Machine Learning Library and the third-party application. For the user of the library, it acts as a wrapper, but behind the scenes, it completely initializes the neural network as described in Section 4.1. Figure 6.13 shows the class diagram of the namespace.

The function `Train` generates a new neural network and trains this with the given data set. The performance of the validation set is used to stop the training as soon as no increase is achieved. The function `CreateFeedForwardNet` generates a new empty neural network. The function `Simulate` starts the forecast and the output of the function is written directly to the test set.

Pps.NeuralNetwork.Data

The namespace Pps.NeuralNetwork.Data provides the data-structure for the neural network. The main class in this namespace is the `DataSet`. It is designed to store a time series and automatically generate features related to it. Every entry in the `DataSet` is represented by a `DataSetRow`. Every `DataSetRow` gets in the constructor a list of inputs, the corresponding names, an output and a timestamp. Figure 6.14, shows the class diagram of this namespace. With the functions `addFeature` and `removeFeature` features can be dynamically added to a `DataSet`. The features are handed over to the `DataSetRow` class.

The functions `getTestSet`, `getTrainingSet` and `getValidationSet` are splitting the whole data set into three parts: the test set, the training set and the validation set. First, at the function call `getTestSet` the number of days is specified for testing. After this, the remaining data is split into validation and training set. Furthermore, the class `DataSet` provides the methods to get the size, combine data sets, to sort or to filter the data. Additionally, it implements the interface `Iterable<T>` so that it is easy to loop through all elements. The features added to this class are not generated immediately, the function call `generateFeatureValues` will generate the values in a batch. The class `Scaler` provides the ability to scale the time series to hourly time slices. Therefore, the function `ScaleToHour` is called. It can be either a start and end timestamp or without. The start and end timestamp are used to truncate data which is not within the time span automatically. The class `DataSetRow` implements the interface `Comparable<T>` to compare two data-set-entries if they have the same timestamp. The class `Normalizer` is a abstract helper class. It provides two functions `Normalize` and `DeNormalize`. These two functions can be used to implement a custom class which can be used to normalize or scale the value of each data row in the data set.

Pps.NeuralNetwork.Features

The namespace Pps.NeuralNetwork.Features provide a basic set of features. All classes implement the abstract class `Feature` of the namespace Pps.NeuralNetwork.Data. Figure 6.15 shows the class diagram of this namespace.

The classes `Year`, `Hour`, `DayOfMonth`, and `DayOfWeek` are straight forward. The class `Holiday` generates a boolean feature, which indicates whether the timestamp corresponds to an Austrian holiday or not. The class `Weekend` provides a flag which indicates if the day is within a weekend. The class `Recurrent` generates a feature which takes the `DataSetRow` output value and delays it for a given day offset. The offset is set at the constructor of this class. The class `CyclicCosHour` and `CyclicSinHour` generates the *cyclic* hour feature presented in Section 5.3.

Pps.NeuralNetwork.Norm

The namespace Pps.NeuralNetwork.Norm provides two basic classes to normalize the data set. All classes in this namespace are derived from the abstract class `Normalizer` of the namespace Pps.NeuralNetwork.Data, see Figure 6.16.

The class `Derivation` provides the ability to derive the data set once. The derivative is taken by subtracting the value of time $t - 1$ from the value of time t. The class `MaxMinNormalizer` provides the ability to normalize the data set, this is done by applying the Formula 5.1.

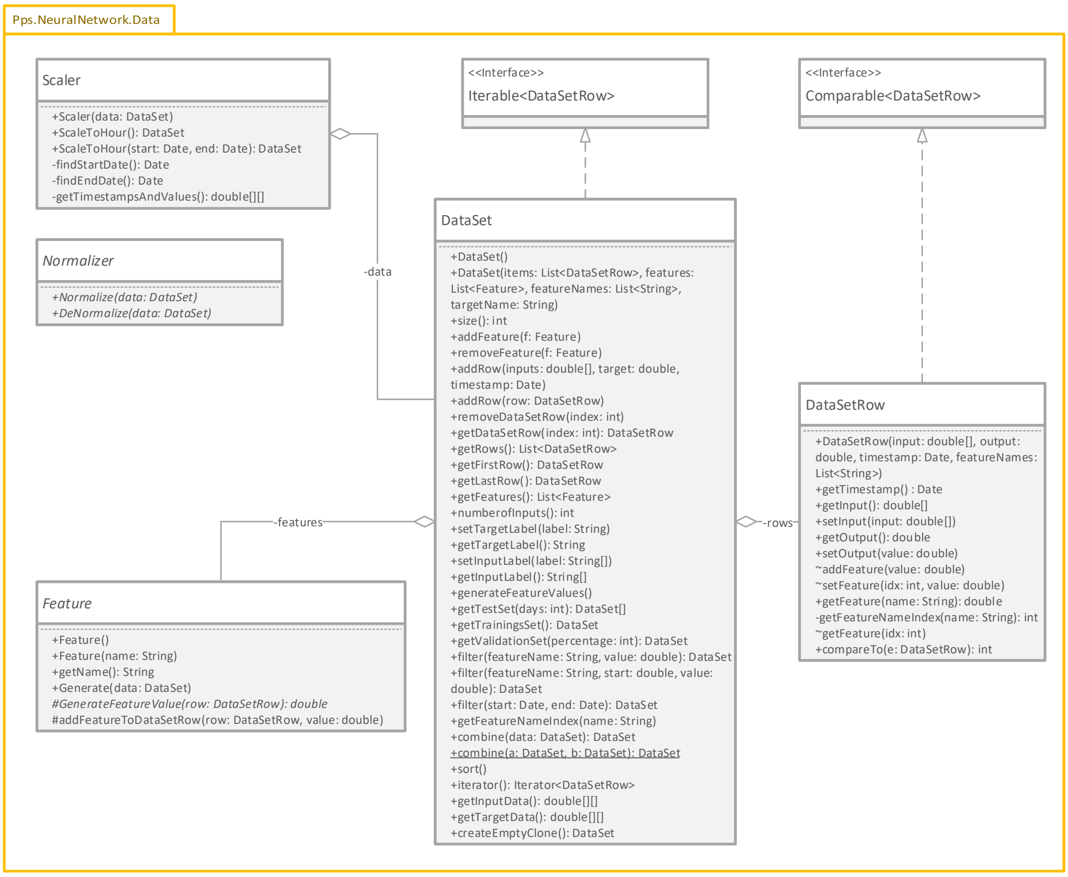

Pps.NeuralNetwork.Data
Scaler
+Scaler(data: DataSet)
+ScaleToHour(): DataSet
+ScaleToHour(start: Date, end: Date): DataSet
-findStartDate(): Date
-findEndDate(): Date
-getTimestampsAndValues(): double[][]
<<Interface>>
Iterable<DataSetRow>
<<Interface>>
Comparable<DataSetRow>
Normalizer
+Normalize(data: DataSet)
+DeNormalize(data: DataSet)
-data
DataSet
+DataSet()
+DataSet(items: List<DataSetRow>, features: List<Feature>, featureNames: List<String>, targetName: String)
+size(): int
+addFeature(f: Feature)
+removeFeature(f: Feature)
+addRow(inputs: double[], target: double, timestamp: Date)
+addRow(row: DataSetRow)
+removeDataSetRow(index: int)
+getDataSetRow(index: int): DataSetRow
+getRows(): List<DataSetRow>
+getFirstRow(): DataSetRow
+getLastRow(): DataSetRow
+getFeatures(): List<Feature>
+numberofInputs(): int
+setTargetLabel(label: String)
+getTargetLabel(): String
+setInputLabel(label: String[])
+getInputLabel(): String[]
+generateFeatureValues()
+getTestSet(days: int): DataSet[]
+getTrainingsSet(): DataSet
+getValidationSet(percentage: int): DataSet
+filter(featureName: String, value: double): DataSet
+filter(featureName: String, start: double, value: double): DataSet
+filter(start: Date, end: Date): DataSet
+getFeatureNameIndex(name: String)
+combine(data: DataSet): DataSet
+combine(a: DataSet, b: DataSet): DataSet
+sort()
+iterator(): Iterator<DataSetRow>
+getInputData(): double[][]
+getTargetData(): double[][]
+createEmptyClone(): DataSet
-features
-rows
DataSetRow
+DataSetRow(input: double[], output: double, timestamp: Date, featureNames: List<String>)
+getTimestamp() : Date
+getInput(): double[]
+setInput(input: double[])
+getOutput(): double
+setOutput(value: double)
~addFeature(value: double)
~setFeature(idx: int, value: double)
+getFeature(name: String): double
-getFeatureNameIndex(name: String): int
~getFeature(idx: int)
+compareTo(e: DataSetRow): int
Feature
+Feature()
+Feature(name: String)
+getName(): String
+Generate(data: DataSet)
#GenerateFeatureValue(row: DataSetRow): double
#addFeatureToDataSetRow(row: DataSetRow, value: double)

Figure 6.14: UML class diagram for the namespace Pps.NeuralNetwork.Data

Comparison of the MATLAB and JAVA implementation

This subsection compares the ANN approach implemented in MATLAB with the presented JAVA implementation. Therefore, the same time series as in Section 6.2 were used to evaluate the performance of the JAVA implementation. Figure 6.17 shows the outcome of the evaluation. The forecast of the electricity time series in Figure 6.17(a) showed for the first two days a good forecast but the following days were less accurately predicted. The other forecasts showed an oscillating output, which leads to high and low peaks. These peaks have a negative impact on the performance of the JAVA implementation. The according error measures are illustrated in Table 6.12. Comparing the values to Table 6.3, it is evident that the JAVA implementation did not perform as well as the MATLAB implementation. After further investigations of the neural network implementation in MATLAB and in the Encog Machine Learning library, no difference in the used algorithms could be found. Both use the Nguyen-Widrow weight initialization and the LM algorithm to train the network. One difference between the MATLAB and the JAVA implementation

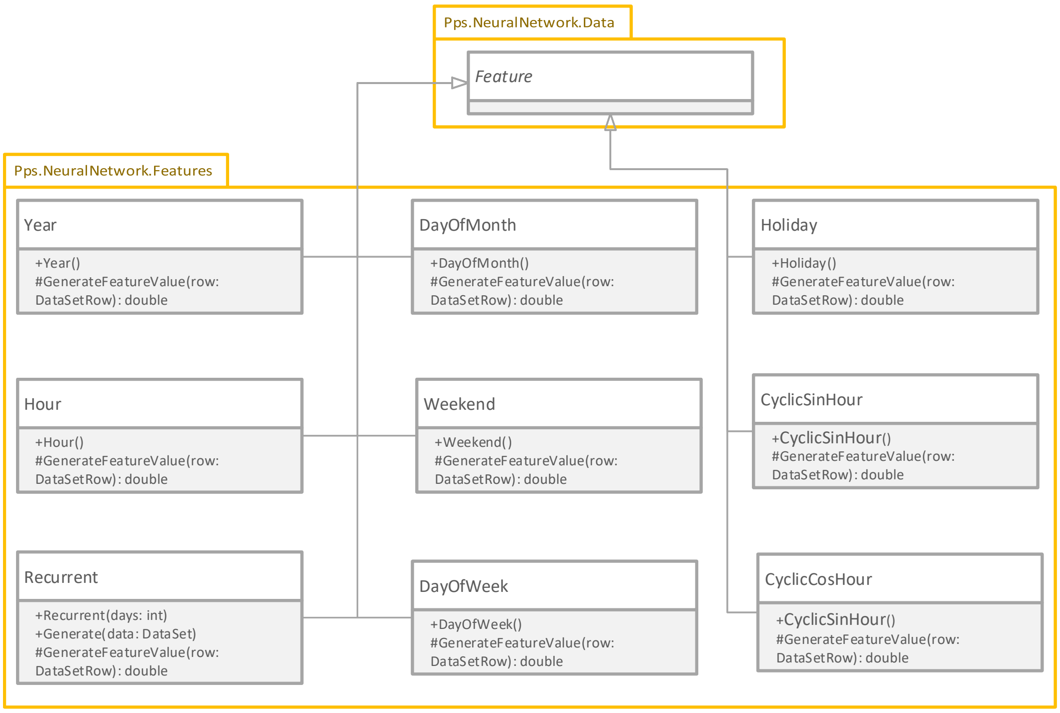

Pps.NeuralNetwork.Data
Feature
Pps.NeuralNetwork.Features
Year
+Year()
#GenerateFeatureValue(row: DataSetRow): double
DayOfMonth
+DayOfMonth()
#GenerateFeatureValue(row: DataSetRow): double
Holiday
+Holiday()
#GenerateFeatureValue(row: DataSetRow): double
Hour
+Hour()
#GenerateFeatureValue(row: DataSetRow): double
Weekend
+Weekend()
#GenerateFeatureValue(row: DataSetRow): double
CyclicSinHour
+CyclicSinHour()
#GenerateFeatureValue(row: DataSetRow): double
Recurrent
+Recurrent(days: int)
+Generate(data: DataSet)
#GenerateFeatureValue(row: DataSetRow): double
DayOfWeek
+DayOfWeek()
#GenerateFeatureValue(row: DataSetRow): double
CyclicCosHour
+CyclicSinHour()
#GenerateFeatureValue(row: DataSetRow): double

Figure 6.15: UML class diagram for the namespace Pps.NeuralNetwork.Features

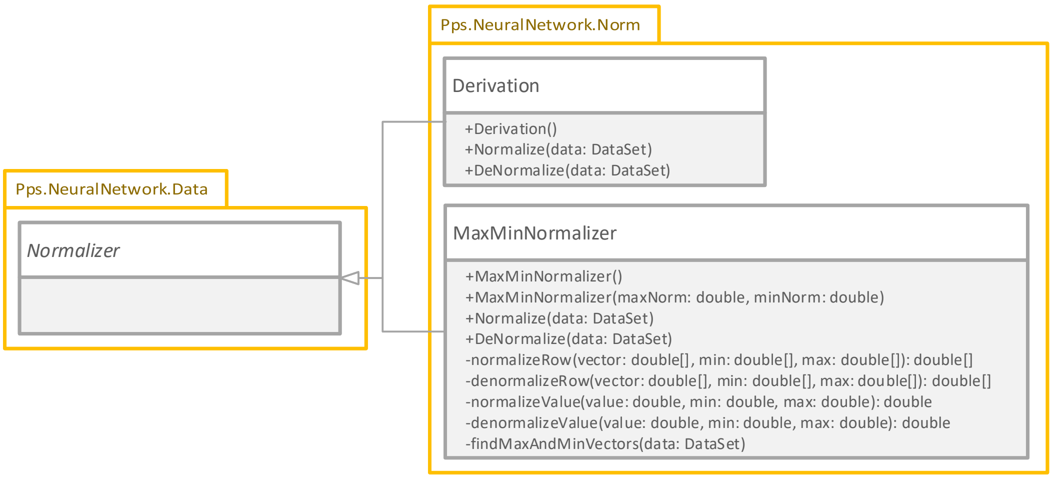
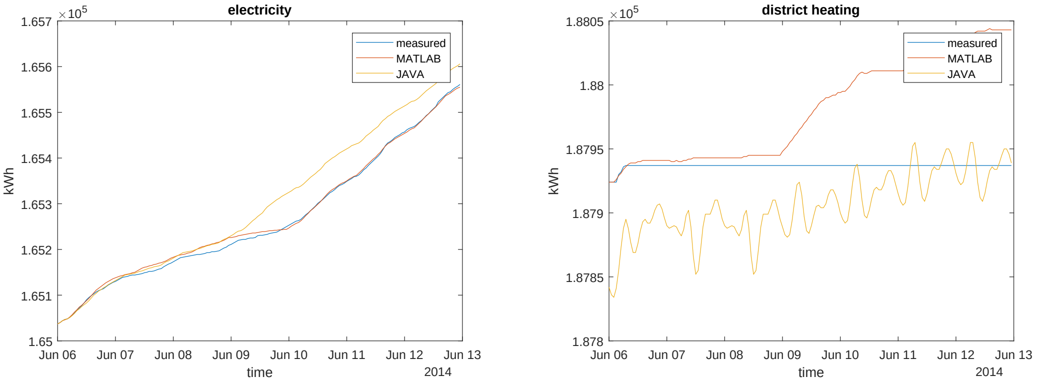

Pps.NeuralNetwork.Norm
Derivation
+Derivation()
+Normalize(data: DataSet)
+DeNormalize(data: DataSet)
Pps.NeuralNetwork.Data
Normalizer
MaxMinNormalizer
+MaxMinNormalizer()
+MaxMinNormalizer(maxNorm: double, minNorm: double)
+Normalize(data: DataSet)
+DeNormalize(data: DataSet)
-normalizeRow(vector: double[], min: double[], max: double[]): double[]
-denormalizeRow(vector: double[], min: double[], max: double[]): double[]
-normalizeValue(value: double, min: double, max: double): double
-denormalizeValue(value: double, min: double, max: double): double
-findMaxAndMinVectors(data: DataSet)

Figure 6.16: UML class diagram for the namespace Pps.NeuralNetwork.Normalizer

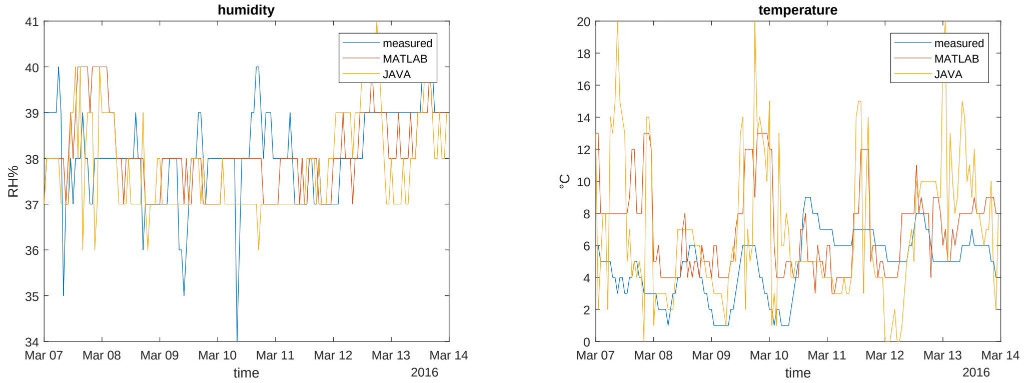
is the random number generator. Random numbers are used either to get an initial weight set for the Nguyen-Widrow algorithm and for choosing the LM training set and the LM validation set. Further research and development is needed to make the output of the JAVA implementation identical to MATLAB.

Table 6.12: Results of the JAVA implementation of the proposed ANN approach.

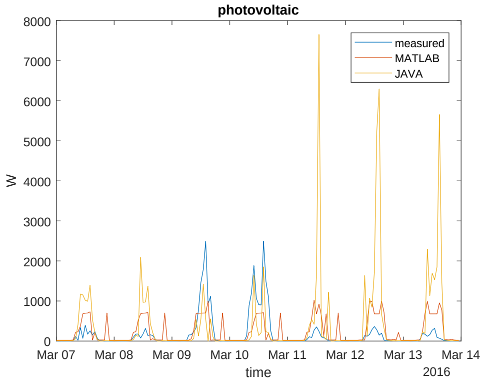
Type	MAE	SMAPE	MASE
electricity	36.309	0.0	0.517
district heating	33.017	0.0	73.142
humidity	1.017	2.839	1.017
temperature	3.732	64.392	2.672
photovoltaic	397.559	124.756	2.041



(a) electricity (b) district heating



(c) humidity (d) temperature



(e) photovoltaic

Figure 6.17: Comparison of the real measured, the MATLAB predicted, and the JAVA predicted values.

CHAPTER 7

Conclusion

7.1 Summary

This thesis investigates proper algorithms to reliable forecast sensor data for utilization in BAS optimization and management tasks. After the literature research, three algorithms were chosen for evaluation. The first presented forecasting algorithm was the ARIMA algorithm, widely used in econometrics to forecasts sales statistics, followed by two machine learning algorithms, namely ANN and SVM. These two algorithms are often used in pattern recognition, but they are also capable to solve regression tasks. The three algorithms were evaluated in a comprehensive performance test resulting in a definite tendency to ANN as the most promising algorithm. Chapter 2 starts with an introduction to time series analysis and related terms. Definitions are provided to understand the behavior of the time series described in Chapter 5. In the next step, these time series were preprocessed. Therefore, every time series is inspected to identify anomalies like missing data, sensor malfunction, or irregularities. Anomalies were removed, and the pchip algorithm interpolated the missing data. After the cleanup of the data, features were generated and used as inputs for the machine learning algorithms. The features are either time or value related. After the preparation of the time series, the best configurations for the individual algorithms were evaluated, as described in Chapter 6. For the ARIMA algorithm, the autocorrelation function and the partial autocorrelation function for a time series were calculated, and then the optimal values for the parameters p, q, and d were selected. In the machine learning approaches, special attention has been paid to the input features. In contrast to the literature, where mostly y_t is provided as input to predict y_{t+1}, this work used y_{t-167} as input. For the ANN algorithm, the number of hidden layers as well as the number of neurons per layer was evaluated based on the publications [13], [14] and [46], and through testing. Different to the literature where often one or two layers are used, the ANN approach performed better with a many layer design. Six hidden layers were used, and the number of nodes was similar to an autoencoder configuration. The SVM was configured to use a linear kernel and the SMO solver based on [43]. Most of the related work tested the algorithms at a special timeframe and for one specific time series. In this work, 30 sensors were selected from two buildings with at least one year of historical data to get a meaningful statement. The sensors were grouped into the categories electricity, district heating, humidity, temperature, and photovoltaic production. Thus, every group consists of six sensors. For every sensor, eight different forecasts were made, two at every season of the year. One forecast is in the middle of the season and one at the end. To evaluate the performance of the algorithms, the MAE, SMAPE, and MASE were calculated. The ANN algorithm was the most accurate in all tested seasons. Finally, the ANN algorithm was implemented in JAVA for further reuse. Therefore, a library was designed to import the historical data, generate all necessary features, configure the ANN, train it, and forecast the data. The library uses the Encog Machine Learning library [45], which provides the neural network and the appropriate training algorithm. The presented thesis shows that it is possible to predict future values without any expert knowledge of the building and its processes. Furthermore, it shows that it is essential to evaluate the forecasting models at different seasons in the year, because depending on the model some seasons were easier to predict than others. Some results of the presented ANN approach and the seasonal testing were already used in a conference paper [47].

7.2 Future work

This thesis offers a basis for further development of the ANN approach as different components can be enhanced or improved. In a first step, the codebase should be moved from MATLAB to the open source language Python[1] for better performance and more flexibility in adapting machine learning algorithms. Furthermore, the change from MATLAB to Python should solve the issue that two codebases can deviate from each other, despite the same algorithms were used. The next component to enhance is the data acquisition and preparation. Therefore, more data is needed from various buildings. This information should be used to generate additional unique features for the ANN algorithm to support the training algorithm to find general relations between the input and output of the network. Besides the data generated by the building also weather data can help to improve the forecasts. Furthermore, particular attention should be paid to the reliable collection of the sensor data. In this context, more research is needed to detect sensor malfunctions and how to handle a breakdown. It turned out that data preprocessing is as crucial as the network structure of the ANN to achieve a high forecasting accuracy. Therefore, the ANN should be enhanced to a deep neural network, and various configurations have to be tested. This testing should result in a further understanding of the inner network structure of an ANN.

Moreover, the developed library is intended to be used with an optimization algorithm to optimize the building efficiency. Possible usage is to optimize the energy consumption

---

[1] `https://www.python.org`

of the building by scheduling dynamic loads in the building so that first the overall energy consumption is lowered, and second the energy consumption is flattened.

APPENDIX A

Detailed forecast results

Table A.1: MAE results for every sensor described in Section 6.3

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
E-1								
ARIMA	19.3	22.2	73.3	8.9	44.3	27.0	81.6	35.3
ANN	17.2	45.7	31.9	37.2	13.4	58.0	4.8	59.0
SVM	17.4	14.6	25.8	24.5	18.8	55.8	6.5	22.1
E-2								
ARIMA	6.5	32.9	136.8	90.5	54.5	17.7	80.0	26.2
ANN	18.3	25.6	77.7	14.9	28.2	32.5	63.7	6.7
SVM	25.2	18.7	119.7	25.1	7.7	20.3	22.2	20.4
E-3								
ARIMA	38.5	27.7	467.6	14.6	26.5	14.8	293.1	14.4
ANN	52.7	180.2	274.7	39.2	29.6	15.8	75.5	49.6
SVM	91.4	46.0	404.9	137.3	163.1	42.3	19.2	325.8
E-4								
ARIMA	68.8	21.7	1014.1	199.0	34.4	341.9	528.4	88.7
ANN	227.1	48.7	853.9	181.6	175.4	377.6	309.4	32.7
SVM	297.7	148.0	749.8	302.2	126.0	496.8	274.3	326.0
E-5								
ARIMA	44.6	11.3	39.4	60.5	31.8	24.1	57.1	23.7
ANN	72.5	17.6	48.6	82.9	9.9	23.3	17.1	17.0
SVM	151.0	71.9	177.6	213.1	36.8	69.2	109.0	197.9
E-6								
ARIMA	3.5	1.6	10.9	3.2	5.3	4.4	10.0	1.0
ANN	2.0	0.4	2.5	4.0	3.4	7.1	1.5	2.8

Continued on next page

Table A.1 – *Continued from previous page*

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
SVM	6.0	8.7	19.2	6.7	4.2	11.4	2.4	6.1
DH-1								
ARIMA	77.8	37.7	18.4	83.5	80.0	475.3	9.0	154.4
ANN	358.8	121.9	9.2	157.0	168.6	142.3	15.0	95.2
SVM	359.7	29.7	24.9	200.3	559.3	221.2	8.4	591.6
DH-2								
ARIMA	469.1	119.1	94.4	2195.2	327.4	845.7	58.6	365.4
ANN	400.7	636.5	270.5	375.5	303.1	391.9	309.6	225.4
SVM	1274.4	655.3	302.3	298.9	2604.8	818.4	34.5	389.1
DH-3								
ARIMA	188.0	463.9	58.3	15551.9	1034.5	1797.4	154.3	1136.9
ANN	3368.7	283.3	142.2	3261.8	470.2	3002.2	501.7	669.5
SVM	5454.5	1921.4	10.3	6082.1	5973.4	1553.4	2.3	4715.9
DH-5								
ARIMA	350.9	0.0	0.0	1224.8	546.6	28.0	11.0	22.7
ANN	29.0	17.0	0.2	205.9	232.0	193.5	12.2	33.4
SVM	66.6	2.4	8.0	1202.4	1173.9	52.0	8.4	22.2
DH-6								
ARIMA	95.8	29.0	21.4	171.9	96.3	45.6	16.9	149.8
ANN	227.2	30.0	1.2	84.3	182.6	271.3	17.5	91.2
SVM	378.3	2.9	172.1	354.9	570.2	239.7	3.4	316.6
DH-4								
ARIMA	139.8	39.6	29.1	72.3	71.2	59.3	31.5	85.9
ANN	538.5	11.0	8.0	137.5	55.4	284.0	81.0	60.3
SVM	592.1	1.5	16.2	340.4	471.6	362.2	3.9	785.7
RH-3								
ARIMA	3.3	2.3	3.7	6.9	3.3	6.6	4.0	5.9
ANN	2.7	2.0	3.6	4.7	3.5	4.3	3.7	6.1
SVM	2.8	1.9	2.9	4.8	3.7	4.7	3.8	5.9
FH-1								
ARIMA	2.2	2.5	2.9	5.6	2.2	6.3	3.0	3.8
ANN	1.7	2.0	3.3	2.6	1.7	5.6	3.9	4.8
SVM	2.0	1.7	2.6	3.6	1.8	5.8	3.3	4.8
FH-2								
ARIMA	2.7	2.7	3.0	6.2	2.2	5.9	2.9	3.5
ANN	1.8	2.2	3.2	2.6	1.4	5.6	3.3	5.3
SVM	2.2	2.0	2.9	3.1	1.6	6.3	3.3	5.4
RH-1								
ARIMA	1.3	5.4	4.3	4.0	2.7	8.3	1.7	3.0
ANN	1.5	4.0	3.7	1.2	1.8	8.0	4.2	3.8

Continued on next page

Table A.1 – *Continued from previous page*

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
SVM	1.5	4.4	2.8	2.2	2.1	8.8	3.8	4.9
				RH-2				
ARIMA	1.2	6.7	4.1	2.0	1.4	10.9	2.0	3.7
ANN	0.6	5.2	4.2	0.8	1.3	9.7	6.0	4.9
SVM	0.7	5.7	2.8	2.9	1.3	10.6	5.3	5.4
				OH-1				
ARIMA	17.0	11.5	23.3	21.3	11.2	18.8	15.4	16.1
ANN	14.1	10.1	11.0	16.6	21.0	11.5	14.5	12.1
SVM	14.2	8.5	12.5	17.0	19.0	9.3	17.2	15.3
				PT-1				
ARIMA	4.1	4.8	10.5	6.3	7.8	10.3	3.4	6.5
ANN	5.1	12.7	7.4	7.2	8.0	10.5	7.1	8.3
SVM	5.8	12.8	8.1	6.2	8.3	10.6	6.7	8.4
				PT-2				
ARIMA	4.1	4.6	10.3	6.4	8.4	10.3	3.5	6.3
ANN	4.8	12.7	7.3	7.3	7.9	10.9	9.9	8.2
SVM	5.9	12.5	8.2	6.3	8.0	10.4	6.5	8.3
				OT-1				
ARIMA	1.6	2.5	7.5	6.3	10.4	5.1	3.9	6.4
ANN	2.4	9.7	5.6	4.6	6.4	8.9	7.4	7.4
SVM	2.5	9.4	5.2	4.2	5.6	9.3	5.2	7.8
				RT-3				
ARIMA	0.6	0.8	2.7	0.7	0.7	1.8	0.5	1.5
ANN	0.8	2.4	1.1	1.5	1.3	1.7	1.4	1.4
SVM	1.0	2.4	1.2	1.3	1.4	1.7	1.8	1.5
				RT-1-M				
ARIMA	0.2	0.2	0.3	0.3	0.4	0.3	0.2	0.5
ANN	0.3	0.8	0.4	0.3	0.9	0.5	0.3	0.3
SVM	0.3	0.9	0.3	0.3	0.8	0.5	0.1	0.2
				RT-4				
ARIMA	0.3	0.4	0.3	0.4	0.3	0.3	0.2	0.5
ANN	0.2	0.8	0.3	0.4	0.6	0.2	0.3	0.2
SVM	0.3	0.7	0.3	0.4	0.6	0.2	0.4	0.2
				PV-1				
ARIMA	585.7	322.4	909.2	178.8	462.1	240.3	385.9	842.8
ANN	384.9	304.5	398.1	293.0	511.3	255.6	385.2	328.9
SVM	434.1	415.8	491.0	89.5	551.8	384.5	184.0	349.1
				PV-2				
ARIMA	219.4	161.0	450.2	94.4	229.0	122.8	120.8	146.3
ANN	177.8	208.9	232.8	136.0	220.2	138.5	168.2	174.5

Continued on next page

Table A.1 – *Continued from previous page*

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
SVM	209.8	183.9	385.8	60.8	225.3	155.0	92.4	175.8
	PV-3							
ARIMA	0.8	1.2	11.9	3.8	1.5	4.2	1.4	1.8
ANN	1.4	1.7	5.7	3.7	1.7	1.2	2.6	1.6
SVM	1.2	0.9	8.8	3.0	1.0	1.0	1.6	1.7
	PV-4							
ARIMA	378.5	347.7	459.5	163.3	386.3	217.8	208.4	262.5
ANN	315.9	349.8	317.1	228.5	417.5	218.0	262.0	258.8
SVM	385.8	335.5	443.0	125.9	390.6	293.9	157.5	313.6
	PV-5							
ARIMA	381.5	270.2	832.5	325.7	384.6	213.9	204.4	264.6
ANN	318.5	356.2	396.7	230.8	377.8	348.9	359.9	258.2
SVM	374.6	308.3	589.2	107.3	412.8	287.8	166.4	308.1
	PV-6							
ARIMA	385.7	351.7	757.5	171.8	389.7	282.8	331.3	305.1
ANN	343.4	366.2	352.4	242.3	384.8	213.1	300.0	297.0
SVM	376.0	318.9	403.1	112.0	388.1	286.9	166.0	316.2

Table A.2: SMAPE results for every sensor described in Section 6.3

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
	E-1							
ARIMA	0.012	0.013	0.054	0.006	0.029	0.017	0.062	0.025
ANN	0.011	0.028	0.023	0.025	0.009	0.036	0.004	0.042
SVM	0.011	0.009	0.019	0.017	0.012	0.035	0.005	0.016
	E-2							
ARIMA	0.004	0.020	0.100	0.062	0.036	0.011	0.061	0.018
ANN	0.012	0.015	0.057	0.010	0.019	0.020	0.049	0.005
SVM	0.016	0.011	0.088	0.017	0.005	0.012	0.017	0.014
	E-3							
ARIMA	0.008	0.005	0.104	0.003	0.005	0.003	0.068	0.003
ANN	0.010	0.033	0.061	0.008	0.006	0.003	0.017	0.011
SVM	0.018	0.009	0.090	0.029	0.033	0.008	0.004	0.071
	E-4							
ARIMA	0.007	0.002	0.120	0.022	0.004	0.035	0.064	0.010
ANN	0.024	0.005	0.101	0.020	0.019	0.038	0.038	0.004
SVM	0.031	0.015	0.088	0.033	0.014	0.050	0.033	0.037
	E-5							
ARIMA	0.054	0.013	0.056	0.079	0.040	0.029	0.083	0.033

Continued on next page

Table A.2 – *Continued from previous page*

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
ANN	0.088	0.021	0.069	0.109	0.012	0.028	0.025	0.023
SVM	0.184	0.085	0.253	0.280	0.046	0.083	0.159	0.273
				E-6				
ARIMA	0.026	0.012	0.100	0.027	0.042	0.031	0.095	0.009
ANN	0.015	0.003	0.023	0.034	0.027	0.051	0.014	0.025
SVM	0.045	0.061	0.176	0.056	0.033	0.081	0.023	0.054
				DH-1				
ARIMA	0.023	0.011	0.007	0.028	0.025	0.136	0.003	0.055
ANN	0.105	0.035	0.003	0.053	0.053	0.041	0.006	0.034
SVM	0.105	0.008	0.009	0.068	0.175	0.063	0.003	0.212
				DH-2				
ARIMA	0.040	0.010	0.010	0.210	0.029	0.069	0.006	0.036
ANN	0.034	0.052	0.027	0.036	0.027	0.032	0.031	0.022
SVM	0.108	0.053	0.031	0.029	0.233	0.067	0.003	0.039
				DH-3				
ARIMA	0.004	0.010	0.002	0.422	0.026	0.041	0.004	0.032
ANN	0.079	0.006	0.004	0.088	0.012	0.069	0.014	0.019
SVM	0.128	0.043	0.000	0.165	0.150	0.035	0.000	0.133
				DH-5				
ARIMA	0.167	0.000	0.000	0.699	0.284	0.013	0.006	0.013
ANN	0.014	0.008	0.000	0.117	0.121	0.091	0.007	0.020
SVM	0.032	0.001	0.005	0.686	0.613	0.025	0.005	0.013
				DH-6				
ARIMA	0.021	0.006	0.005	0.042	0.022	0.010	0.004	0.038
ANN	0.050	0.007	0.000	0.021	0.042	0.059	0.004	0.023
SVM	0.084	0.001	0.044	0.087	0.132	0.052	0.001	0.080
				DH-4				
ARIMA	0.038	0.011	0.010	0.023	0.021	0.016	0.011	0.029
ANN	0.146	0.003	0.003	0.043	0.016	0.076	0.028	0.020
SVM	0.160	0.000	0.006	0.107	0.137	0.096	0.001	0.263
				RH-3				
ARIMA	8.674	6.042	8.547	19.050	9.857	15.640	10.271	19.198
ANN	7.208	5.075	8.178	12.179	10.324	9.991	9.433	19.806
SVM	7.327	4.837	6.650	12.439	10.827	10.911	9.713	19.089
				FH-1				
ARIMA	5.631	6.153	6.296	14.784	5.941	13.393	6.980	11.481
ANN	4.338	4.888	6.987	6.392	4.410	12.127	8.934	14.172
SVM	5.060	4.192	5.593	9.074	4.663	12.663	7.681	14.357
				FH-2				
ARIMA	6.992	6.543	6.432	16.654	5.764	12.419	6.700	10.902

Continued on next page

Table A.2 – *Continued from previous page*

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
ANN	4.722	5.287	6.778	6.657	3.453	12.077	7.451	15.635
SVM	5.796	4.730	6.096	7.811	3.970	13.663	7.418	15.903
RH-1								
ARIMA	3.111	12.845	8.858	10.431	6.927	17.102	3.996	8.430
ANN	3.636	9.176	7.759	2.914	4.364	16.303	9.522	10.687
SVM	3.698	10.199	5.943	5.667	5.123	18.093	8.760	13.543
RH-2								
ARIMA	3.076	16.018	8.220	5.269	3.557	22.480	5.100	11.494
ANN	1.502	12.056	8.446	2.124	3.199	19.384	13.621	15.013
SVM	1.812	13.505	5.769	7.895	3.140	21.611	12.237	16.554
OH-1								
ARIMA	22.845	18.008	37.339	26.578	21.858	31.447	18.606	22.442
ANN	20.087	17.050	16.315	19.777	38.154	21.528	18.585	16.952
SVM	20.285	14.088	18.868	20.253	35.267	17.370	22.841	20.752
PT-1								
ARIMA	186.435	16.359	36.258	79.850	43.849	30.119	28.213	-246.339
ANN	32.876	51.525	28.958	82.195	56.473	36.305	52.287	-986.753
SVM	92.964	52.188	32.373	80.559	56.702	35.907	47.711	-312.928
PT-2								
ARIMA	52.069	15.021	34.525	73.468	42.270	29.330	26.717	624.534
ANN	81.575	49.670	27.959	76.117	50.382	34.990	62.663	453.838
SVM	73.962	49.192	31.302	73.051	51.038	34.029	44.037	-87.951
OT-1								
ARIMA	39.435	10.975	34.168	76.743	61.935	18.833	37.026	195.332
ANN	68.477	53.525	27.779	64.900	60.332	37.597	60.399	536.460
SVM	62.999	52.181	25.993	60.907	51.507	39.746	46.082	866.816
RT-3								
ARIMA	2.452	2.888	9.893	2.779	2.887	6.699	2.171	5.901
ANN	3.081	9.165	4.183	6.090	5.156	6.215	5.823	5.435
SVM	4.102	9.166	4.548	5.455	5.778	6.215	7.147	6.124
RT-1-M								
ARIMA	0.910	0.867	1.224	1.218	1.518	1.162	0.975	2.022
ANN	1.280	3.174	1.471	1.221	3.859	1.845	1.142	1.214
SVM	1.444	3.808	1.381	1.160	3.541	2.107	0.525	1.013
RT-4								
ARIMA	1.122	1.712	1.238	1.520	1.300	1.060	0.874	1.998
ANN	0.999	3.148	1.253	1.649	2.612	0.894	1.235	0.812
SVM	1.083	3.019	1.148	1.566	2.556	0.943	1.577	0.973
PV-1								
ARIMA	-61.724	-124.860	-134.390	49.409	117.204	51.246	-86.516	-179.090

Continued on next page

Table A.2 – *Continued from previous page*

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
ANN	78.168	120.715	143.192	181.524	144.343	115.984	170.205	152.930
SVM	136.099	39.310	109.935	146.035	79.247	36.721	87.508	59.008
	PV-2							
ARIMA	43.563	35.747	-134.894	36.909	60.334	47.785	37.088	34.601
ANN	34.565	82.555	93.371	63.634	57.409	65.986	79.072	36.333
SVM	42.021	52.894	104.410	40.017	72.663	54.181	56.258	43.321
	PV-3							
ARIMA	-70.194	-97.661	115.528	12.024	111.564	130.145	-60.794	-31.358
ANN	94.143	130.672	118.213	73.504	119.150	100.752	90.128	78.630
SVM	94.042	22.182	124.142	68.220	80.269	-22.417	-28.677	79.128
	PV-4							
ARIMA	37.382	63.843	-73.237	33.164	29.432	24.797	40.892	32.763
ANN	35.212	63.240	66.718	51.887	58.420	37.072	55.615	38.150
SVM	52.840	42.417	84.516	55.580	41.689	39.929	50.936	47.164
	PV-5							
ARIMA	112.746	-174.175	-135.341	173.769	100.552	7.211	-98.600	139.600
ANN	105.674	121.147	136.611	169.912	114.821	119.551	160.916	105.680
SVM	54.585	-5.879	120.263	-18.831	-12.338	14.978	41.858	508.121
	PV-6							
ARIMA	46.820	-8.490	-26.162	47.939	42.402	-16.154	108.086	132.007
ANN	56.490	116.381	132.587	80.027	75.676	101.260	107.437	62.130
SVM	47.302	82.404	114.377	84.736	56.529	30.073	266.261	56.520

Table A.3: MASE results for every sensor described in Section 6.3

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
	E-1							
ARIMA	0.2	0.2	0.9	0.1	0.4	0.4	1.0	0.3
ANN	0.2	0.5	0.4	0.3	0.1	0.8	0.1	0.6
SVM	0.2	0.2	0.3	0.2	0.2	0.8	0.1	0.2
	E-2							
ARIMA	0.1	0.3	1.6	0.7	0.4	0.2	0.7	0.2
ANN	0.1	0.2	0.9	0.1	0.2	0.4	0.6	0.1
SVM	0.2	0.2	1.4	0.2	0.1	0.2	0.2	0.2
	E-3							
ARIMA	0.1	0.1	2.1	0.0	0.1	0.0	0.9	0.0
ANN	0.2	0.4	1.2	0.1	0.1	0.1	0.2	0.2
SVM	0.3	0.1	1.8	0.4	0.5	0.1	0.1	1.1
	E-4							

Continued on next page

Table A.3 – *Continued from previous page*

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
ARIMA	0.1	0.0	3.2	0.3	0.1	0.9	1.2	0.2
ANN	0.4	0.1	2.7	0.3	0.3	1.0	0.7	0.1
SVM	0.5	0.3	2.4	0.4	0.2	1.3	0.6	0.6
	E-5							
ARIMA	0.8	0.5	0.8	0.8	0.4	1.1	2.3	0.4
ANN	1.2	0.8	0.9	1.0	0.1	1.1	0.7	0.3
SVM	2.6	3.1	3.5	2.6	0.5	3.2	4.4	3.6
	E-6							
ARIMA	0.2	0.3	1.0	0.2	0.3	0.4	1.4	0.1
ANN	0.1	0.1	0.2	0.3	0.2	0.7	0.2	0.3
SVM	0.4	1.5	1.7	0.5	0.2	1.1	0.3	0.5
	DH-1							
ARIMA	0.2	Inf	2.3	0.2	0.2	3.1	Inf	0.7
ANN	1.0	Inf	1.2	0.3	0.4	0.9	Inf	0.4
SVM	1.0	Inf	3.2	0.4	1.2	1.4	Inf	2.7
	DH-2							
ARIMA	0.4	Inf	0.9	1.7	0.2	1.2	Inf	0.8
ANN	0.3	Inf	2.7	0.3	0.2	0.5	Inf	0.5
SVM	1.0	Inf	3.0	0.2	1.8	1.1	Inf	0.8
	DH-3							
ARIMA	0.0	Inf	96.4	2.3	0.2	0.7	Inf	0.7
ANN	0.8	Inf	235.2	0.5	0.1	1.2	Inf	0.4
SVM	1.3	Inf	17.1	0.9	1.0	0.6	Inf	2.9
	DH-5							
ARIMA	1.2	Inf	Inf	2.9	1.7	13.6	Inf	3.9
ANN	0.1	Inf	Inf	0.5	0.7	93.8	Inf	5.7
SVM	0.2	Inf	Inf	2.8	3.7	25.2	Inf	3.8
	DH-6							
ARIMA	0.3	Inf	352.9	0.3	0.2	Inf	Inf	1.1
ANN	0.8	Inf	20.1	0.2	0.4	Inf	Inf	0.7
SVM	1.3	Inf	2843.7	0.7	1.4	Inf	Inf	2.4
	DH-4							
ARIMA	0.4	Inf	9.1	0.1	0.1	Inf	Inf	0.3
ANN	1.5	Inf	2.5	0.2	0.1	Inf	Inf	0.2
SVM	1.7	Inf	5.0	0.5	0.9	Inf	Inf	2.7
	RH-3							
ARIMA	1.4	1.4	1.0	2.3	0.7	1.8	0.9	1.4
ANN	1.1	1.2	1.0	1.5	0.7	1.2	0.9	1.5
SVM	1.1	1.1	0.8	1.6	0.7	1.3	0.9	1.4
	FH-1							

Continued on next page

Table A.3 – *Continued from previous page*

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
ARIMA	1.4	1.9	0.9	4.4	1.2	2.5	1.4	1.5
ANN	1.1	1.5	1.0	2.0	0.9	2.2	1.8	1.9
SVM	1.3	1.3	0.8	2.8	1.0	2.3	1.5	1.9
				FH-2				
ARIMA	1.4	2.1	1.0	4.8	1.2	2.1	1.5	1.7
ANN	0.9	1.8	1.0	2.0	0.7	2.0	1.7	2.6
SVM	1.2	1.6	0.9	2.4	0.8	2.3	1.7	2.6
				RH-1				
ARIMA	1.1	3.1	1.5	4.3	1.5	2.9	1.2	4.0
ANN	1.3	2.2	1.3	1.3	0.9	2.8	3.1	5.1
SVM	1.3	2.5	1.0	2.4	1.1	3.0	2.8	6.5
				RH-2				
ARIMA	1.3	3.4	1.2	2.9	1.0	4.0	1.8	4.0
ANN	0.6	2.6	1.2	1.2	0.9	3.5	5.3	5.3
SVM	0.8	2.9	0.8	4.3	0.9	3.9	4.7	5.8
				OH-1				
ARIMA	1.6	1.4	1.7	2.0	0.8	2.0	1.3	1.9
ANN	1.3	1.2	0.8	1.6	1.4	1.3	1.2	1.5
SVM	1.3	1.0	0.9	1.6	1.3	1.0	1.4	1.8
				PT-1				
ARIMA	0.8	1.8	1.7	2.1	1.4	2.5	1.0	1.2
ANN	1.0	4.7	1.2	2.4	1.4	2.5	2.1	1.5
SVM	1.1	4.8	1.3	2.0	1.4	2.5	1.9	1.5
				PT-2				
ARIMA	0.8	1.9	1.7	2.1	1.5	2.6	1.0	1.2
ANN	0.9	5.3	1.2	2.3	1.4	2.7	2.9	1.5
SVM	1.1	5.2	1.3	2.0	1.4	2.6	1.9	1.6
				OT-1				
ARIMA	0.8	1.1	2.7	2.7	2.7	2.2	3.6	2.5
ANN	1.2	4.4	2.0	2.0	1.6	3.8	7.0	2.9
SVM	1.3	4.3	1.9	1.8	1.4	4.0	4.9	3.0
				RT-3				
ARIMA	0.7	1.1	3.0	1.4	1.2	2.3	0.8	1.0
ANN	0.9	3.4	1.2	3.1	2.1	2.1	2.2	0.9
SVM	1.2	3.4	1.3	2.8	2.3	2.1	2.7	1.0
				RT-1-M				
ARIMA	1.4	1.2	2.4	1.5	1.8	2.6	1.7	2.9
ANN	2.0	4.4	2.9	1.5	4.6	4.1	2.0	1.8
SVM	2.2	5.2	2.8	1.5	4.2	4.6	0.9	1.5
				RT-4				

Continued on next page

Table A.3 – *Continued from previous page*

	Winter	Spring	Summer	Autumn	Winter*	Spring*	Summer*	Autumn*
ARIMA	1.2	1.1	1.5	1.4	0.9	1.1	0.9	2.1
ANN	1.0	1.9	1.5	1.5	1.8	0.9	1.3	0.9
SVM	1.1	1.9	1.3	1.4	1.8	1.0	1.7	1.0
	PV-1							
ARIMA	1.2	4.2	2.1	2.1	1.2	1.1	1.8	2.3
ANN	0.8	4.0	0.9	3.5	1.3	1.2	1.8	0.9
SVM	0.9	5.4	1.1	1.1	1.5	1.8	0.9	1.0
	PV-2							
ARIMA	0.9	5.4	2.0	1.9	1.2	1.3	1.1	0.8
ANN	0.7	7.1	1.0	2.7	1.2	1.4	1.6	1.0
SVM	0.8	6.2	1.7	1.2	1.2	1.6	0.9	1.0
	PV-3							
ARIMA	2.0	34.8	9.4	1.6	3.2	8.7	1.7	3.1
ANN	3.4	49.1	4.5	1.6	3.8	2.4	3.2	2.8
SVM	2.8	26.2	7.0	1.3	2.2	2.1	2.0	3.0
	PV-4							
ARIMA	0.9	6.8	1.2	1.9	1.2	1.2	1.1	0.9
ANN	0.7	6.9	0.8	2.7	1.3	1.2	1.4	0.8
SVM	0.9	6.6	1.1	1.5	1.2	1.6	0.8	1.0
	PV-5							
ARIMA	0.9	6.3	2.2	3.9	1.2	1.2	1.1	0.9
ANN	0.7	8.2	1.0	2.8	1.2	2.0	2.0	0.8
SVM	0.9	7.1	1.5	1.3	1.3	1.7	0.9	1.0
	PV-6							
ARIMA	0.9	6.9	1.9	2.0	1.2	1.6	1.7	1.0
ANN	0.8	7.2	0.9	2.8	1.2	1.2	1.6	1.0
SVM	0.9	6.2	1.0	1.3	1.2	1.6	0.9	1.0

List of Figures

2.1 Sample time series 8
2.2 Illustration of a time series with trend and seasonality 9
2.3 Autocorrelation of the trend and seasonality sample in Figure 2.2. 11

3.1 Linear filter model, adapted from [17, p. 8] 14
3.2 Autocorrelation and partial autocorrelation functions for the realizations of an AR time series, $x_t = 40 + a_t + 0.8x_{t-1}$, $\sigma_a^2 = 2$ 16
3.3 Autocorrelation and partial autocorrelation functions for the realizations of a MA time series, $x_t = 40 - a_t - 0.8a_{t-1}$, $\sigma_a^2 = 2$ 17

4.1 Perceptron 21
4.2 Exemplary FFNN with two input nodes, three hidden nodes and one output node 23
4.3 Jordan neural network 27
4.4 Unrolled Jordan neural network 27
4.5 Elman neural network 28
4.6 Unrolled Elman neural network 28
4.7 Linear SVM, soft margin visualization, adapted from [31] 30

5.1 Example of the different types of data used in the thesis 33
5.2 Example for invalid data and sensor faults. 37
5.3 Example for misleading interpretation of the correlation. 38
5.4 Correlation-Experiment 1 39
5.5 Correlation-Experiment 2 40
5.6 Correlation-Experiment 3 41
5.7 Correlation-Experiment 4 41
5.8 Correlation-Experiment 5 42
5.9 Electricity consumption grouped by the weekday for the sensor E-7 in the year 2015 from the *Building 2* dataset. The red line shows the mean of the day, and the dashed blue lines the standard deviation of the day. In the plot *Holidays*, only national holidays of Austria are included. 44

6.1 Iterative model design, adapted from [17, p. 16] 50
6.2 Evaluation of the electricity parameters. 51

6.3 Autocorrelation and partial autocorrelation for the categories: district heating, humidity, temperature, and photovoltaic production 52
6.4 One week evaluation forecast for the categories: district heating, humidity, temperature, and photovoltaic production for the ARIMA approach 53
6.5 FFNN used to forecast a time series of the category electricity. There are four input nodes, 27 hidden nodes spread over five hidden layers, and one output node. 55
6.6 Measured (blue) and predicted (red) time series for the categories electricity, district heating, humidity, temperature, and photovoltaic production for the ANN approach 56
6.7 Measured (blue) and predicted (red) time series for the categories electricity, district heating, humidity, temperature, and photovoltaic production for the SVM approach 59
6.8 Electricity forecasts for the sensor E-5 (left) in the season Winter and E-2 (right) in the season Autumn*. 62
6.9 Direct heating forecasts for the sensor DH-3 (left) in the season Summer and DH-5 (right) in the season Winter. 63
6.10 Humidity forecasts for the sensor FH-1 (left) in the season Winter and RH-2 (right) in the season Spring. 64
6.11 Temperature forecasts for the sensor PT-1 (left) in the season Autumn* and RT-1-M (right) in the season Summer. 65
6.12 Photovoltaic production forecasts for the sensor PV-3 (left) in the season Winter and PV-4 (right) in the season Summer. 66
6.13 UML class diagram for the namespace Pps.NeuralNetwork 67
6.14 UML class diagram for the namespace Pps.NeuralNetwork.Data 69
6.15 UML class diagram for the namespace Pps.NeuralNetwork.Features 70
6.16 UML class diagram for the namespace Pps.NeuralNetwork.Normalizer 70
6.17 Comparison of the real measured, the MATLAB predicted, and the JAVA predicted values. 72

List of Tables

4.1 Commonly used activation functions for perceptrons 21
4.2 Common kernels for SVM 29

5.1 Used sensors for performance evaluation. 34
5.2 Results of the augmented Dickey-Fuller test applied to the sample data presented in Figure 5.1 37

6.1 Results of the parameter evaluation for the ARIMA approach. 53
6.2 $SARIMA$ model parameters for the categories defined in Figure 5.1. 54
6.3 Results of the ANN approach. 57
6.4 Results of the SVM approach. 58
6.5 timeframes for the forecasting algorithm evaluation. 60
6.6 Overall results of the forecast accuracy, counted from the MAE result table (see Table A.1) 61
6.7 Mean MAE of the Electricity type of sensors for all seasons. 62
6.8 Mean MAE of the District Heating type of sensors. 63
6.9 Mean MAE of the Humidity type of sensors. 64
6.10 Mean MAE of the Temperature type of sensors. 65
6.11 Mean MAE of the Photovoltaic type of sensors. 66
6.12 Results of the JAVA implementation of the proposed ANN approach. 71

A.1 MAE results for every sensor described in Section 6.3 77
A.2 SMAPE results for every sensor described in Section 6.3 80
A.3 MASE results for every sensor described in Section 6.3 83

List of Acronyms

ANN artificial neural network. ix, 3–6, 19, 20, 31, 42, 43, 45, 49, 54–58, 60–66, 69, 71, 73, 74, 88, 89

AR autoregressive. 5, 6, 13–18, 26, 49, 57, 87

ARIMA autoregressive integrated moving average. ix, 3–6, 13, 18, 28, 31, 47, 49–53, 55, 58, 61–66, 73, 88, 89

ARIMAX autoregressive integrated moving average with exogenous inputs. 5, 13

ARMA autoregressive moving average. 5, 13, 17, 18, 20

BAS building automation system. ix, 1, 2, 32, 37, 38, 66, 73

BMLP bridged multilayer perceptron. 5

EBP error backpropagation algorithm. 5, 22–24, 26

FCC fully connected cascade. 5

FFNN feedforward neural network. 20, 22, 23, 26, 27, 47, 54, 55, 66, 87, 88

iid independent identically distributed. 38

LM Levenberg-Marquardt algorithm. 5, 24, 26, 47, 55, 69

MA moving average. 6, 13–18, 49, 87

MAE mean absolute error. 3, 4, 6, 47, 48, 50, 51, 60–66, 74, 89

MASE mean absolute scaled error. 3, 4, 6, 48, 51, 52, 61–65, 74

MLP multilayer perceptron. 5

NBN neuron by neuron. 5

pchip shape-preserving piecewise cubic interpolation. 36, 73

RNN recurrent neural network. 20, 26, 54

SMAPE symmetric mean absolute percentage error. 3, 4, 6, 48, 51, 60–63, 74

SMO sequential minimal optimization. 30, 57, 74

SSE sum of squared error. 55

SVM support vector machine. ix, 3–6, 19, 28, 29, 31, 43, 45, 47, 49, 57–59, 61–66, 73, 74, 88, 89

SVR support vector regression. 5, 6, 28, 29, 57, 58

Bibliography

[1] P. Domingues, P. Carreira, R. Vieira, and W. Kastner, „Building automation systems: Concepts and technology review“, *Computer Standards & Interfaces*, vol. 45, pp. 1–12, 2016.

[2] G. Gaur, N. Mehta, R. Khanna, and S. Kaur, „Demand side management in a smart grid environment“, in *International Conference on Smart Grid and Smart Cities (ICSGSC)*, IEEE, 2017, pp. 227–231.

[3] P. Palensky and D. Dietrich, „Demand side management: Demand response, intelligent energy systems, and smart loads“, *IEEE Transactions on Industrial Informatics*, 2011.

[4] L. Dannecker, *Energy Time Series Forecasting.* Springer Vieweg, Wiesbaden, 2015.

[5] D. C. Montgomery, C. L. Jennings, and M. Kulahci, *Introduction to time series analysis and forecasting.* John Wiley & Sons, 2015.

[6] *Matlab*, The MathWorks, Inc., Natick, MA, USA, 2015b.

[7] N. I. Sapankevych and R. Sankar, „Time series prediction using support vector machines: A survey“, *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, 2009.

[8] D. Lixing, L. Jinhu, L. Xuemei, and L. Lanlan, „Support vector regression and ant colony optimization for hvac cooling load prediction“, in *Computer Communication Control and Automation (3CA)*, IEEE, vol. 1, 2010, pp. 537–541.

[9] H. Son and C. Kim, „Short-term forecasting of electricity demand for the residential sector using weather and social variables“, *Resources, Conservation and Recycling*, 2016.

[10] L. Xuemei, D. Lixing, D. Yuyuan, and L. Lanlan, „Hybrid support vector machine and arima model in building cooling prediction“, in *International Symposium on Computer, Communication, Control and Automation (3CA)*, vol. 1, 2010, pp. 533–536.

[11] L. Qu, Y. Chen, and Z. Liu, „Time series forecasting model with error correction by structure adaptive rbf neural network“, in *6th World Congress on Intelligent Control and Automation*, vol. 2, 2006, pp. 6831–6835.

[12] L. Jinhu, L. Xuemei, D. Lixing, and J. Liangzhong, „Applying principal component analysis and weighted support vector machine in building cooling load forecasting“, in *Computer and Communication Technologies in Agriculture Engineering (CCTAE)*, IEEE, vol. 1, 2010, pp. 434–437.

[13] D. Hunter, H. Yu, M. S. Pukish III, J. Kolbusz, and B. M. Wilamowski, „Selection of proper neural network sizes and architectures - a comparative study“, *IEEE Transactions on Industrial Informatics*, vol. 8, no. 2, 2012.

[14] G. Escrivá-Escrivá, C. Álvarez-Bel, C. Roldán-Blay, and M. Alcázar-Ortega, „New artificial neural network prediction method for electrical consumption forecasting based on building end-uses“, *Energy and Buildings*, vol. 43, no. 11, pp. 3112–3119, 2011.

[15] P. Mandal, T. Senjyu, N. Urasaki, and T. Funabashi, „A neural network based several-hour-ahead electric load forecasting using similar days approach“, *International Journal of Electrical Power & Energy Systems*, vol. 28, no. 6, pp. 367–373, 2006.

[16] H. Cui and X. Peng, „Short-term city electric load forecasting with considering temperature effects: An improved arimax model“, *Mathematical Problems in Engineering*, 2015.

[17] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis forcasting and control.* John Wiley & Sons, 2016.

[18] B. H. Andrews, M. D. Dean, R. Swain, and C. Cole, „Building arima and arimax models for predicting long-term disability benefit application rates in the public/private sectors“, University of Southern Main, Tech. Rep., 2013.

[19] G. U. Yule, „On a method of investigating periodicities in disturbed series, with special reference to wolfer's sunspot numbers“, *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 226, 1927.

[20] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice.* OTexts, 2014.

[21] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification.* John Wiley & Sons, 2012.

[22] F. Rosenblatt, „The perceptron: A probabilistic model for information storage and organization in the brain“, *Psychological Review*, vol. 65, no. 6, 1958.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, „Learning representations by back-propagating errors“, *Nature*, 1986.

[24] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, New York, 2006.

[25] H. Yu and B. M. Wilamowski, „Levenberg–marquardt training“, *Industrial Electronics Handbook*, vol. 5, no. 12, 2011.

[26] M. T. Hagan and M. B. Menhaj, „Training feedforward networks with the marquardt algorithm“, *IEEE transactions on Neural Networks*, vol. 5, no. 6, 1994.

[27] H. Liu, „On the levenberg-marquardt training method for feed-forward neural networks“, in *Sixth International Conference on Natural Computation*, vol. 1, 2010, pp. 456–460.

[28] M. I. Jordan, „Serial order: A parallel distributed processing approach“, *ICS Report 8604*, 1986.

[29] J. L. Elman, „Finding structure in time“, *Cognitive Science*, vol. 14, 1990.

[30] V. Vapnik, *The nature of statistical learning theory*. Springer, New York, 2013.

[31] A. J. Smola and B. Schölkopf, „A tutorial on support vector regression“, *Statistics and Computing*, vol. 14, no. 3, 2004.

[32] MathWorks. (2018). Understanding support vector machine regression, [Online]. Available: `https://de.mathworks.com/help/stats/understanding-support-vector-machine-regression.html` (visited on 01/26/2018).

[33] J. C. Platt, „Sequential minimal optimization: A fast algorithm for training support vector machines“, Microsoft Research, Tech. Rep., 1998.

[34] R. Isermann, *Fault-diagnosis systems*. Springer, Berlin, Heidelberg, 2006.

[35] G. U. Yule, „Why do we sometimes get nonsense-correlations between time-series?–a study in sampling and the nature of time-series“, *Journal of the Royal Statistical Society*, vol. 89, no. 1, 1926.

[36] S. F. Crone, J. Guajardo, and R. Weber, „The impact of preprocessing on support vector regression and neural networks in time series prediction.“, in *DMIN*, 2006, pp. 37–42.

[37] E. Busseti, I. Osband, and S. Wong, „Deep learning for time series modeling“, Technical report, Stanford University, Tech. Rep., 2012.

[38] F. J. Marin, F. Garcia-Lagos, G. Joya, and F. Sandoval, „Global model for short-term load forecasting using artificial neural networks“, *IEE Proceedings - Generation, Transmission and Distribution*, vol. 149, no. 2, 2002.

[39] R. J. Hyndman and A. B. Koehler, „Another look at measures of forecast accuracy“, *International Journal of Forecasting*, vol. 22, no. 4, 2006.

[40] H. S. Hippert, C. E. Pedreira, and R. C. Souza, „Neural networks for short-term load forecasting: A review and evaluation“, *Power Systems, IEEE Transactions on*, vol. 16, no. 1, 2001.

[41] D. Nguyen and B. Widrow, „Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights“, in *IJCNN International Joint Conference on Neural Networks*, vol. 3, 1990, pp. 21–26.

[42] A. Pavelka and A. Procházka, „Algorithms for initialization of neural network weights“, in *12th Annual Conference, MATLAB*, 2004, pp. 453–459.

[43] S. Rüping, „Svm kernels for time series analysis“, SFB 475, Universität Dortmund, Tech. Rep., 2001.

[44] MathWorks. (2018). Fit a support vector machine regression model, [Online]. Available: `https://de.mathworks.com/help/stats/fitrsvm.html` (visited on 03/03/2018).

[45] J. Heaton, „Encog: Library of interchangeable machine learning models for java and c#“, *Journal of Machine Learning Research*, 2015.

[46] S. F. Crone and N. Kourentzes, „Feature selection for time series prediction – a combined filter and wrapper approach for neural networks“, *Neurocomputing*, vol. 73, no. 10–12, 2010.

[47] D. Schachinger, J. Pannosch, and W. Kastner, „Adaptive learning-based time series prediction framework for building energy management“, in *IEEE International Conference on Industrial Electronics for Sustainable Energy Systems*, Jan. 2018, pp. 453–458.