Master Thesis

Forecasting in the Supply Chain with Machine Learning Techniques

June 5, 2018

MATTHIAS DELFS (Matr. No. 1832258)
Degree Course: Computing in the Humanities (M.Sc.)

Supervisors: Prof. Dr. Ute Schmid, Dr. Martin Kraus
Abstract

This thesis was written in cooperation with Siemens Healthineers. The goal was to predict future sales figures of X-ray systems for the supply chain management. Better knowledge about future sales numbers enables better allocation of resources. The product line consists of five different systems. Sales figures are predicted for all systems as a single quantity and also individually. The historic data supplied covered 15 years and consisted of 180 data points. These data points are a monthly census of sales related figures. Traditional time series modeling techniques from statistics were compared with newly emerging machine learning approaches. The established time series modeling techniques were exponential smoothing and ARIMA. The machine learning techniques consisted of modeling feedforward neural networks and random forests. The performance of all methods was measured with the mean absolute percentage error. The best performance of all implemented methods resulted from an extended ARIMA model (ARIMAX) with a relative error of 9.55%. Moreover, the thesis included implementing a software tool for the supply chain management to make forecasts in practice.
# Contents

1 Introduction

2 Theoretical Background and Prior Work
   2.1 Machine Learning Techniques
       2.1.1 Artificial Neural Nets
       2.1.2 Random Forest Regression
   2.2 Statistical Techniques for Time Series Modeling
       2.2.1 Basic Definitions
       2.2.2 Exponential Smoothing
       2.2.3 ARIMA
   2.3 Prior Work

3 Methods
   3.1 Description of the Forecasting Task
   3.2 Data
       3.2.1 Data Set
       3.2.2 Missing Data
       3.2.3 Correlation of Features
   3.3 Machine Learning Techniques
       3.3.1 Data Preparation
       3.3.2 Setup
       3.3.3 FNN modeling
       3.3.4 Random forest modeling
   3.4 Statistical Techniques
       3.4.1 Setup
       3.4.2 ETS Modeling
       3.4.3 ARIMA Modeling
   3.5 Technical Setup

4 Evaluation
   4.1 Neural Net
       4.1.1 Total
       4.1.2 Single Device
   4.2 Random Forest
       4.2.1 Total
       4.2.2 Single Device
   4.3 ETS

i
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data structure</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>Data augmentation options</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>Hyperparameters NN</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>Hyperparameters random forest</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>Software information</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>Cross-validation result (NN Total)</td>
<td>45</td>
</tr>
<tr>
<td>7</td>
<td>Cross-validation result (NN Single Device)</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>Test result per system type (NN Single Device)</td>
<td>50</td>
</tr>
<tr>
<td>9</td>
<td>Cross-validation result (RF Total)</td>
<td>54</td>
</tr>
<tr>
<td>10</td>
<td>Cross-validation result (RF Single Device)</td>
<td>57</td>
</tr>
<tr>
<td>11</td>
<td>Test result per system type (RF Single Device)</td>
<td>57</td>
</tr>
<tr>
<td>12</td>
<td>Result rolling forecast ETS</td>
<td>57</td>
</tr>
<tr>
<td>13</td>
<td>Result rolling forecast ARIMA</td>
<td>58</td>
</tr>
<tr>
<td>14</td>
<td>Result rolling forecast ARIMAX</td>
<td>59</td>
</tr>
<tr>
<td>15</td>
<td>Result ensemble</td>
<td>60</td>
</tr>
<tr>
<td>16</td>
<td>Final result (Total)</td>
<td>61</td>
</tr>
<tr>
<td>17</td>
<td>Final result (MP)</td>
<td>61</td>
</tr>
<tr>
<td>18</td>
<td>Final result (Zeego)</td>
<td>61</td>
</tr>
<tr>
<td>19</td>
<td>Final result (Biplane)</td>
<td>61</td>
</tr>
<tr>
<td>20</td>
<td>Final result (Ceiling)</td>
<td>62</td>
</tr>
<tr>
<td>21</td>
<td>Final result (Floor)</td>
<td>62</td>
</tr>
<tr>
<td>22</td>
<td>Forecast Supply Chain Management</td>
<td>63</td>
</tr>
<tr>
<td>23</td>
<td>Forecast result for the next month</td>
<td>64</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Single-hidden-layer network</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Single ANN neuron</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Training a neural net</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Decision tree</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>ETS time series</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>ARIMA time series</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>Stationarity and autocorrelation</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>Order lifecycle</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>Forecast interval</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>Bias-variance trade-off</td>
<td>23</td>
</tr>
<tr>
<td>11</td>
<td>Data point</td>
<td>24</td>
</tr>
<tr>
<td>12</td>
<td>Data correlation</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>Moving window</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>Sequence sample</td>
<td>28</td>
</tr>
<tr>
<td>15</td>
<td>ARIMAX data structure</td>
<td>33</td>
</tr>
<tr>
<td>16</td>
<td>Sales data</td>
<td>34</td>
</tr>
<tr>
<td>17</td>
<td>ARIMA modeling</td>
<td>35</td>
</tr>
<tr>
<td>18</td>
<td>Framework architecture</td>
<td>36</td>
</tr>
<tr>
<td>19</td>
<td>Experiments (NN Total)</td>
<td>40</td>
</tr>
<tr>
<td>20</td>
<td>Effect of sequence length (NN Total)</td>
<td>41</td>
</tr>
<tr>
<td>21</td>
<td>Effect of augmentation and dropout (NN Total)</td>
<td>41</td>
</tr>
<tr>
<td>22</td>
<td>Augmentation effect filtered by topologies (NN Total)</td>
<td>42</td>
</tr>
<tr>
<td>23</td>
<td>Dropout effect filtered by topologies (NN Total)</td>
<td>43</td>
</tr>
<tr>
<td>24</td>
<td>Effect of augmentation and dropout in most complex topology (NN Total)</td>
<td>44</td>
</tr>
<tr>
<td>25</td>
<td>Effect of early stopping and L2 regularization (NN Total)</td>
<td>45</td>
</tr>
<tr>
<td>26</td>
<td>Training with and without early stopping (NN Total)</td>
<td>46</td>
</tr>
<tr>
<td>27</td>
<td>Experiments (NN Single Device)</td>
<td>46</td>
</tr>
<tr>
<td>28</td>
<td>Effect of sequence length effect (NN Single Device)</td>
<td>47</td>
</tr>
<tr>
<td>29</td>
<td>Effect of augmentation and dropout (NN Single Device)</td>
<td>47</td>
</tr>
<tr>
<td>30</td>
<td>Experiments without augmentation (NN Single Device)</td>
<td>48</td>
</tr>
<tr>
<td>31</td>
<td>Experiments with augmentation (NN Single Device)</td>
<td>49</td>
</tr>
<tr>
<td>32</td>
<td>Effect of early stopping and L2 regularization (NN Single Device)</td>
<td>49</td>
</tr>
<tr>
<td>33</td>
<td>Training single-hidden layer vs four-hidden layer (NN Single Device)</td>
<td>51</td>
</tr>
<tr>
<td>34</td>
<td>Experiments (RF Total)</td>
<td>51</td>
</tr>
<tr>
<td>35</td>
<td>Effect of sequence length and augmentation (RF Total)</td>
<td>52</td>
</tr>
<tr>
<td>Page</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Effect of number of trees and leaf size (RF Total)</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Percentage of features selected for split (RF Total)</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Experiments (RF Single Device)</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Effect of sequence length and augmentation (RF Single Device)</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Effect of number of trees and leaf size (RF Single Device)</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Percentage of features selected for split (RF Single Device)</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>ETS error distribution over all experiments</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>ARIMA error distribution over all experiments</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>ARIMAX error distribution over all experiments</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Training with early stopping (NN Total)</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Training without early stopping (NN Total)</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Training single-hidden layer (NN Single Device)</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Training four-hidden layer (NN Single Device)</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>Augmentation effect filtered by topologies (NN Single Device)</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Dropout effect filtered by topologies (NN Single Device)</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>Early Stopping effect filtered by topologies (NN Single Device)</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

*Siemens Healthineers Advanced Therapies*¹ is a producer of innovative high tech products for digital X-ray technology with production sites in Germany (Forchheim and Kemnath) and China (Shenzhen and Shanghai). Customers worldwide use their products for diagnosis and therapy in angiography, neurology, cardiology and surgery. The supply chain management (SCM) of *Advanced Therapies* organizes the process from order intake to production, logistics, installation and startup and final handover to the customer. Thus, the SCM needs to structure all internal workflows related to manufacturing the product as bringing raw materials into the organization, assigning production slots, and scheduling the right number of workers. Many functions had been outsourced in the past to focus on core competencies. This provides more flexibility and reduces costs. Yet, this approach implies aligning the workflow with various business partners and therefore potential complications. Altogether, the supply chain has to be regarded as a complete cross-functional system and not as a system of isolated components. Hence, optimizing a supply chain in order to maximize efficiency has to be done across the entire system. That requires a mindful design, thorough planning and execution, and constant monitoring of supply chain activities to detect problems and potentials. The SCM of *Advanced Therapies* offers flexibility to the customers with short lead times of often only a few weeks between order intake and delivery. Therefore, production aligned with the customer may only take between four and seven working days. Allocating all resources in time imposes a great challenge for the strategy team of the supply chain management.

Thus, having reliable forecasts about sales figures two months into the future may optimize the supply chain significantly. Time series analysis studies observations from the past to build models that describe the structure and evolution of the data (Shumway and Stoffer, 2011). These models may then be used to predict future values of the time series. Making forecasts based on historic data is a research area with a large number of applications in many fields as meteorology, economics, business or finance. Today, a planning division of *Advanced Therapies* makes manual forecasts based on experience to predict upcoming sales figures. The goal of this thesis is to improve the accuracy of these predictions compared to the status quo and to reduce the effort by automating the process with a prototypical forecasting system.

Our approach is to address the problem from two different directions. First, we apply established time series modeling techniques from statistics, such as exponential smoothing and ARIMA. Second, we use machine learning techniques and model neural networks and random forests to predict future sales numbers. The supplied historic data is very limited. The data cover the last 15 years and consist of only 180 monthly data points that each contains several

¹https://www.healthcare.siemens.com/
attributes of relevant information. *Advanced Therapies* assembles five different X-ray system types: *MP*, *Zeego*, *Biplane*, *Ceiling*, and *Floor*. That information is also reflected in the data. We will therefore perform two different kinds of experiments. First, we want to predict the total number of X-ray systems regardless of the type. Second, we want to predict future sales figures for each individual system type. So, this thesis explores the following research question:

**How does forecasting with neural networks and random forest regression perform in comparison to the traditional time series modeling techniques exponential smoothing and ARIMA with that particular data set?**

The thesis is structured as follows. Chapter 2 provides a theoretical background about neural networks, random forest regression, and traditional time series modeling with exponential smoothing and ARIMA. Moreover, it contains a short section about prior work on time series modeling. Chapter 3 explains the data set and in detail how each of the theoretical concepts was applied to our specific problem. Moreover, the chapter includes a technical description of our software implementation. Chapter 4 provides the results of all experiments and their analysis. This chapter is followed by a conclusion in Chapter 5.
Chapter 2

Theoretical Background and Prior Work

2.1 Machine Learning Techniques

Mitchell (1997) describes machine learning from a classical artificial intelligence (AI) perspective as a computer’s ability to improve performing a task based on its experience. Conceptually, this high-level definition follows the human learning approach. Similar to human memory, data serves as the computer’s source of experience to perform cognitive tasks. Bishop (2006) by contrast explains the field more from an engineering point of view as pattern recognition, where computer algorithms automatically detect regularities in a dataset. Thus, machine learning algorithms have the capability to discover patterns autonomously in a given dataset to build predictive models. The learning process or training is based on data-driven decisions to continuously improve predictive performance on the data.

In this context supervised learning tasks require training examples consisting of data pairs, both an input and a target vector. Given these pairs the algorithm constructs a model mapping the inputs to corresponding target outputs. Preparing training data by adding the desired target vectors to sample inputs is often called labeling. Labeling e-mails as "spam" or "no spam" and training a spam filter is an example for the supervised learning of a classification task. If the desired output has continuous variables rather than discrete categories we speak of a regression problem. The existing task to forecast values of a time series is a regression problem, and we handle it with supervised learning techniques. Dealing with unlabeled data instead, comprising only of input vectors, addresses unsupervised learning approaches. Clustering samples of unstructured data into different groups with similar qualities would be such an example.

An artificial neural net (ANN) and a random forest regression were applied to predict future figures for the Supply Chain Management. Both machine learning algorithms will be explained thoroughly in this chapter.

2.1.1 Artificial Neural Nets

Feedforward Neural Nets

Complex webs with billions of interconnected neurons allow the human brain to communicate and to process information. The name artificial neural network (ANN) has been inspired by...
attempts to describe this system mathematically (Ripley, 1996). Figure 1 shows the three parts of an ANN: the input layer, a flexible number of hidden layers, and an output layer. Each of the layers is fully connected with the next layer, every single neuron or unit has a connection to all neurons in the following layer. Feedforward neural nets (FNN) have no closed directed cycles unlike other network architectures as recurrent neural nets (RNN).

![Figure 1: Single-hidden-layer network.](image)

We give a short description of feedforward nets. The neural net can basically be described as a function $f$ mapping an input vector $(x_1, \ldots, x_N) \in \mathbb{R}^N$ to an output vector $(y_1, \ldots, y_S) \in \mathbb{R}^S$

$$f(x_1, x_2, \ldots, x_N) = (y_1, \ldots, y_S).$$

![Figure 2: Single ANN neuron.](image)

Figure 1 shows a simple FNN with a single hidden layer, two inputs and one output.

For each input $x_i$, there is a unit in the input layer. It propagates $x_i$ to the neurons in the first hidden layer. The structure of a neuron in a hidden or output layer is as follows (see Figure 2). The neuron has inputs $x_1, \ldots, x_n$, a bias $b$, weights $w_1, \ldots, w_n$ (one for each input) and one output $z$. The output $z$ is computed from the neuron’s activation

$$a = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b,$$
which is the sum of the weighted inputs and the bias, by using the *activation function* $h$

$$z = h(a).$$

For its mathematical properties to being nonlinear and differentiable the function $h$ may be
choosen to be sigmoidal in the hidden layers, for example the logistic sigmoid function

$$a \mapsto \frac{1}{1 + e^{-a}}.$$

In case of a regression problem, the activation function used in the output layer is typi-
cally the identity function, i.e. $z = a$. For classification the output units represent the output
categories and the output layers’ activation functions have to provide a probability distribu-
tion. Consequently, multiclass classification problems use a softmax activation function in
the output layer, whereas multiple binary classification tasks apply logistic sigmoid functions
(Goodfellow et al., 2016).

The input $x_i$ of a neuron originates from the output of another neuron in the previous
layer. Hence, we may consider the weight $w_i$ as being attached to the connection between the
two neurons. The weight of the connection from neuron $i$ in layer $(k - 1)$ to neuron $j$ in layer
$k$ is often denoted by $w_{ji}^{(k)}$, and the bias of neuron $j$ by $w_j^{(k)}$ (it is the weight of an additional
constant input 1). Then the activation $a_j^{(k)}$ of neuron $j$ in layer $k$ is

$$a_j^{(k)} = \sum_{i=1}^{M_{k-1}} w_{ji}^{(k)} z_i^{(k-1)} + w_j^{(k)},$$

where $z_i^{(k-1)}$ is the output of neuron $i$ in layer $(k - 1)$ and $M_k$ is the number of neurons in
layer $k$.

For a single-hidden-layer network with $N$ inputs $x_1, \ldots, x_n$, $M$ hidden units and $S$ outputs
$y_1, \ldots, y_S$ (as, e.g., in Figure 1) the function $f$ that is implemented by the FNN can be
described as

$$y_k(x, w) = g \left( \sum_{j=1}^{M} w_{kj}^{(2)} h \left( \sum_{i=1}^{N} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right),$$

see, e.g., Bishop (2006). Here $h$ is the activation function of the hidden and $g$ the activation
function of the output layer. Altogether, the neural net is a function mapping a set of input
variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by an adjustable vector $w$ containing
all weights and biases. The weight vector and the differentiability of the activation functions
are fundamental for the backpropagation algorithm. Such a net is in theory arbitrarily ex-
pandable in depth in both dimensions, meaning the number of hidden layers and the number
of units in each of the layers. However, computational power might limit the complexity of
the network in practice.

**Backpropagation Algorithm**

Fitting an ANN with $\hat{y} = f(x, w)$ to a dataset consisting of pairs $\langle x, y \rangle$ requires training.
After having learned the right values for the weights and biases the network function’s out-
come $\hat{y}$ for input $x$ should ideally match the actual value $y$.

\footnote{As is common practice, we denote the output of the model with hat and the actual value without hat.} The backpropagation algorithm
solves this optimization problem by incrementally minimizing the error using gradient descent (Goodfellow et al., 2016).

Depending on the task different error (or loss) functions are appropriate. Measuring the quadratic distance between the output values \( \hat{y} \) and the desired values \( y \) with the sum-of-squares error in the form

\[
E(w) = \frac{1}{2} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2
\]

applies for regression problems. The factor \( \frac{1}{2} \) is added to the term to simplify differentiation. In classification nets, however, cross-entropy loss functions are preferred (Bishop, 2006). The variables \( x \) and \( y \) are constants in \( E \), unlike \( w \), which occurs in \( \hat{y} = f(x, w) \). The gradient of \( E \) with respect to the weight vector \( w \) equals the direction of the steepest ascent of the error function. The gradient vector

\[
\text{grad } E(w) = \nabla E(w) = \left( \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_r} \right)
\]

contains the partial derivatives of \( E \) with respect to all weights. Consequently, a step in direction of the negative gradient leads to a maximal reduction of the error \( E(w) \). At this point we have to assume that the activation functions and hence the error function are continuously differentiable. Figure 3 shows the steps of the backpropagation algorithm with stochastic gradient descent (SGD). In the beginning the weights are initialized. Then, repeatedly, a random subset or mini-batch of \( N \) training samples \( (x_n, y_n) \) gets selected and the outputs \( f(x_n, w) \) and the error \( E(w) \) are computed for this batch. By contrast, normal gradient descent would take the whole training set as an input. Computing \( f(x_n, w) \) the direction of motion through the ANN is forward. The next step in line 5 describes the update of the weight vector \( w \) in direction of the negative gradient of \( E(w) \). The parameter \( \eta \) is called learning rate and determines the size of step. Determining the partial derivatives with respect to all weights and biases requires calculating a partial error at each individual unit of the neural net. This works by propagating the error backwards through the net starting with \( E(w) \) at the output layer. That error enables calculating the error terms of the units in the last hidden layer. All errors and with that all derivatives are calculated stepwise from layer to layer in that way to finally update \( w \). A precise mathematical description can be found in Mitchell (1997) and Bishop (2006). The number of training steps to incrementally update \( w \) gets repeated until a termination condition is met. This condition could be a fixed number of training steps or a threshold for the error value.

<table>
<thead>
<tr>
<th>Data: Pairs of input vectors ( {x} ) and corresponding target vectors ( {y} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: ANN with ( \hat{y} = f(x, w) ) that has adapted to the training data</td>
</tr>
<tr>
<td>1 Initialize weight vector ( w ) with random numbers;</td>
</tr>
<tr>
<td>2 repeat</td>
</tr>
<tr>
<td>3 Select mini-batch of ( N ) data samples ( {x_1, \ldots, x_N} ) from training set;</td>
</tr>
<tr>
<td>4 ( E(w) = \frac{1}{2} \sum_{n=1}^{N} (y_n - f(x_n, w))^2 ); // forward propagation</td>
</tr>
<tr>
<td>5 ( w \leftarrow w - \eta \text{grad } E(w) ); // backpropagation</td>
</tr>
<tr>
<td>6 until Termination condition is met;</td>
</tr>
</tbody>
</table>

Figure 3: Training a neural net with SGD for a regression task.
In each iteration we modify the weights such that the error of the current batch becomes smaller. The hope is that in this way we are able to minimize the error for the training data. Of course, this is not guaranteed. We might end up in a local minimum of the error function, far away from the global minimum, or it may happen that the error does not converge. Choosing a mini-batch instead of taking all data samples for a training iteration has proven to be effective. Finding a minimum of $E(w)$ usually requires thousands to millions iterations subject to the task. Saving computational time by calculating the gradients for small subsets allows many more updates of $w$ and therefore quicker convergence. If the size of the subsample is large enough the gradient estimates will approximately still point in the same direction as the ones of the full dataset (Goodfellow et al., 2016). By contrast, on-line gradient descent makes updates based on a single data point for each training step. One epoch has passed if all data samples have been seen for training once. The handling of $\eta$ during training is crucial to find a sufficiently good minimum of $E$ on the error surface. Training an ANN with backpropagation is considered to be a black box algorithm since the learned weights are not intuitively interpretable.

**Regularization**

A trained ANN model should not only perform well on data used for fitting but also on previously unseen samples. In other words, the model must capture the underlying structure without overfitting to the training data. This issue counts particularly for training sets of small size. Several techniques are available to accomplish this ability to generalize.

One possibility would be adding a regularization term, the $L^2$ parameter norm, known as weight decay to the error function in the form

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 + \frac{\lambda}{2} w^T w,$$

where the hyperparameter $\lambda$ determines its importance. That additional term $w^T w = \sum_{i,j,k} (w^{(k)}_{ji})^2$ is the sum-of-squares of all weights and causes a shrinkage of the weights during backpropagation (Goodfellow et al., 2016). Reducing the values of the weight coefficients prevents learning overly complex structures.

Early stopping avoids overfitting by interrupting the training process of updating the weights at a certain point. The stoppage moment can be detected by observing the error progression on the training set and a hold-out validation set while fitting the ANN. According to Bishop (2006) training should be stopped when reaching the smallest error on the out-of-sample validation data.

Srivastava et al. (2014) introduced dropout as an effective regularization method. The term refers to the key idea of randomly dropping neurons of a network during training. Based on a fixed probability each neuron and its connections are either retained or temporarily dropped. Consequently, each training step updates the weights only for a ”thinned” architecture of the original network. After training the entire ”unthinned” network will be considered for inference. Thus, dropout prevents units from adapting too much to the training data and averages a large number of model combinations to increase out-of-sample performance.

Commonly, machine learning models generalize better if they have more data for fitting. Expanding the size of the training set with synthetic data leads to better performance in various scenarios, in particular regarding object recognition (Simard et al., 2003). Adding random
noise as a form of data augmentation also increases the robustness of the model (Goodfellow et al., 2016). Moreover, hand-designed problem specific augmentation schemes using transformation may greatly reduce the generalization error of a machine learning algorithm. For instance in computer vision, applying transformations as flipping or translating the images often improves generalization significantly (Goodfellow et al., 2016).

2.1.2 Random Forest Regression

Decision Trees

Tree based machine learning algorithms divide a feature space into a set of cuboid partitions called regions $R$. Figure 4 shows the general structure of a decision tree with an example. The two-dimensional input space has been recursively partitioned into five regions $R_1, R_2, \ldots, R_5$. Traversing its corresponding tree structure visualizes the sequential splitting of the data. The nodes test the instances for some feature, the branches represent possible values for that particular attribute, and the leaves contain the final regions. To simplify creating, the tree nodes are only split into two branches. Each data point $(x, y)$ consists of an input vector $x$, $x = (x_1, x_2)$ in our example, and an output $y$. Starting from the root the data gets divided into two regions based on split variable $x_1$ depending whether $x_1 \leq 3$. Each of the resulting subregions can subsequently be split again individually. Region $x_1 \leq 3$ is subdivided at $x_2 = 4$ and region $x_1 > 3$ at $x_1 = 5$. Region $x_1 > 5$ is then split again at $x_2 = 6$. Any new input starts at the root and falls into a region according to the given decision path. Thus, as Mitchell (1997) aptly mentions, a decision tree represents a disjunction of conjunctions in the form

$$(x_1 \leq 3 \land x_2 \leq 4) \lor (x_1 \leq 3 \land x_2 > 4) \lor \ldots \lor (x_1 > 3 \land x_1 > 5 \land x_2 > 6)$$

where the conjunctions contain the decision criteria for each region. A decision tree can be used either for a classification or a regression model.

Figure 4: Decision tree based on Hastie et al. (2009).
The output response for a decision tree is modeled as a constant $c_m$ in the form

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

Here $M$ matches the numbers of regions $R_1, R_2, \ldots, R_M$, $x$ is the input feature vector and $I$ is the indicator function with value 1, if $x \in R_m$, and value 0 else. In a regression task the best choice of $c_m$ for one region is simply to average all $y_i$ falling in that partition. By contrast, for classification all instances grouped in one region belong to the same category. Hence, the response $c_m$ can be any substitute value standing for each category. The major advantage of tree based algorithms is the possibility to draw conclusions about the decision making of the resulting models. That key property of being easily interpretable labels decision trees as a white box.

**CART**

The CART algorithm (Classification And Regression Trees) (see, for example, Hastie et al. (2009)) is a method to learn a data-driven tree structure. It is a greedy optimization algorithm that starts with the whole input space at a single root and then recursively splits the data. Hence, growing the tree it makes always the locally optimal choice to split data into two partitions. CART can also handle classification, but we restrict our discussion to a regression task.

Consider a dataset consisting of pairs of input and output $(x_i, y_i)$ with $x_i = (x_{i1}, x_{i2}, \ldots, x_{iN})$. At each step the algorithm chooses an attribute $j$ out of the $N$ input variables as the split criterion and also sets a split threshold $s$. Naturally, $j$ and $s$ should be chosen to minimize the predictive error at that split node. The minimization criterion for regression is the sum-of-squares error in the form

$$\sum (y_i - f(x_i))^2$$

where $f(x_i)$ equals the response $c_m$ of either one of the new partitions. Consequently, $j$ and $s$ are chosen such that the combined sum-of-squares error of both partitions

$$\sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2$$

becomes minimal. As mentioned above, the response $c_1$ or $c_2$ is the average of all $y_i$ where $x_i$ falls in $R_1(j,s)$ or $R_2(j,s)$. Here, finding the optimal choice of $s$ and $j$ can be done very quickly with exhaustive search, whereas finding the overall best binary partition in terms of minimum sum of squares is in general not practical.

A threshold for a minimum number of instances falling in one region terminates further splitting of nodes to avoid overfitting. Finally, the grown tree $T$ with $M$ leaves will be pruned back to reduce complexity costs. Collapsing nodes returns subtrees $\tilde{T}$ of $T$. These subtrees are evaluated according to the criterion

$$C_\alpha(\tilde{T}) = \sum_{m=1}^{M} Q_m + \alpha M$$

where $M$ ist the number of leaves of $\tilde{T}$ and $Q_m$ is the sum-of-squares error of region $R_m$. Thus, the parameter $\alpha$ balances the tree’s predictive error and its complexity measured by
Larger values of \( \alpha \) prefer smaller trees and vice versa. It can be shown that each value of \( \alpha \) provides a unique subtree \( T_\alpha \) that minimizes \( C_\alpha \). A cross-validation for different values of \( \alpha \) determines the final tree \( T_\hat{\alpha} \) based on the sum-of-squares error. For growing classification trees sum-of-squares is substituted by cross-entropy or the Gini index. Descriptions going into more detail can be found in Bishop (2006) and Hastie et al. (2009). Other algorithms as ID3, C4.5, C5.0 use the statistical measure information gain to decide on the splitting criteria. More information and a precise explanation for these algorithms are in Mitchell (1997).

Random Forests

Random forests combine the predictions of several decision trees to make a composite estimate. Breiman (2001) introduced the concept as a collection of \( k \) tree-structured estimators \( h(x, \theta_k) \) taking an input vector \( x \) and a random vector \( \theta_k \) as arguments. Each of the random vectors \( \theta_k \) is generated independently of the past vectors \( \theta_1, \ldots, \theta_{k-1} \) but with the same distribution (iid independent identically distributed). The elements of randomness chosen for a forest are not predetermined. The Python library scikit-learn implemented by Pedregosa et al. (2011) which was used for the practical part of the thesis contains two random factors. Using bootstrap aggregation or bagging the package chooses a random selection of training examples with replacement to grow each individual tree. Additionally, the split criterion picked for each node is the best among a random subset out of all features. The prediction of all trees would then simply be averaged in case of a regression task. A random forest classifier, however, returns the most popular vote of the individual trees. Usually, the composite models are more robust to noise and perform better than single trees because they have a lower variance (Bishop, 2006; Hastie et al., 2009).

2.2 Statistical Techniques for Time Series Modeling

2.2.1 Basic Definitions

Normal Distribution

A continuous real-valued random variable \( X \) with mean \( \mu \) and variance \( \sigma^2 \) and a probability density \( \varphi \) in the form

\[
\varphi(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (x \in \mathbb{R})
\]

is defined to be normal or Gaussian and denoted as \( X \sim \mathcal{N}(\mu, \sigma^2) \). The graph of the density function is shaped symmetrically like a bell with two inflection points at \( \mu \pm \sigma \).

Stationary Time Series

Real-valued data that fluctuate over time in a seemingly random manner yield a time series. A time series can be modeled as a sequence \((X_t)_{t \in T}\) of random variables. Variable \( X_t \) denotes the data value at time \( t \). The family \((X_t)_{t \in T}\) is a stochastic process. In our application \( t \) is discrete and varies over a subset \( T \) of the integers. The observed values \( x_t \) of \( X_t \) are called realizations. Stationarity of a time series describes a form of regularity in its behavior over time. A weakly stationary time series \( X_t \) is a process meeting two conditions. First, evolving over time the mean value \( \mu_t \) of \( X_t \) does not depend on \( t \) and remains constant. Second, the autocovariance \( \gamma(s, t) \) of two time points \( s \) and \( t \) of the series, which is the covariance of \( X_s \) and \( X_t \), depends only on their distance \( d = |s - t| \), meaning any time points being apart the
same distance \( d \) have the same autocovariance. For instance, a series of uncorrelated random variables with constant mean \( \mu = 0 \) and constant variance \( \sigma^2_w \) meets these conditions. Such a process is referred to as \textit{white noise} and denoted as \( X_t \sim wn(0, \sigma^2_w) \). Weak stationarity differs from \textit{strictly stationary} processes. However, for the further purpose from now on using the term stationary means weakly stationary. Variance, standard deviation and covariance of a time series can be estimated from a sufficiently large sample, by applying the formulae for the empirical variance, empirical standard deviation and so on. Shumway and Stoffer (2011) give a more precise explanation with further distinctions.

**Linear Least Squares**

Consider the problem of fitting a linear regression model to a dataset of \( N \) pairs \((x_i, y_i)\) where \( x_i \) has \( m \) attributes \((x_i \in \mathbb{R}^m, y_i \in \mathbb{R})\)

\[
\hat{y}_i = f(x_i) = \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_m x_{im}.
\]

The difference between the actual value \( y_i \) and the model’s estimate is the error \( e_i \). The parameters \( \beta_i \) of the linear model can be estimated by minimizing the sum-of-squares error

\[
E(\beta) = \frac{1}{2} \sum_{i=1}^{N} e_i^2 = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2.
\]

The minimum is determined by setting the gradient \( \nabla E(\beta) \) of the error function with respect to the weight vector \( \beta \) equal to zero. The resulting system of linear equations to calculate the weights can be solved analytically for small \( m \) or else approximated numerically. More information can be found in Bishop (2006) and Hastie et al. (2009).

**Maximum Likelihood**

A powerful and widely used method for estimating unknown parameters of a statistical model is \textit{maximum likelihood estimation}. Consider a random variable \( X \) and assume that the probability distribution \( g_\theta \) or, if \( X \) is continuous, the density function \( g_\theta \) depends on an unknown parameter \( \theta \), which may be a multi-dimensional vector. Given independent observations \( x_i, i = 1, \ldots, N \), of \( X \) the \textit{likelihood function}

\[
L(\theta; x) = \prod_{i=1}^{N} g_\theta(x_i)
\]

can be formed. In other words, \( L(\theta; x) \) measures the probability of the sample dependent on \( \theta \). The idea of maximum likelihood estimation is to choose \( \theta \) in such a way that it maximizes \( L \) for the observed values. This can be achieved by (partially) differentiating the function with respect to \( \theta \) and setting it to zero. Maximizing the natural logarithm of the likelihood function instead,

\[
\log(L(\theta; x)),
\]

which is called the \textit{log-likelihood} function, is frequently more convenient, because \( \log \) turns the product in the definition of \( L(\theta; x) \) into a sum. The log-likelihood function has its maximum at the same points as the original likelihood function since the logarithm increases strictly. The equation system to determine \( \theta \) is solved similarly to linear least squares. For linear models with Gaussian errors minimizing the sum-of-squares error and maximizing the likelihood is the same. More detail and examples are provided in Bishop (2006), Hastie et al. (2009) and Lehn and Wegmann (2006).
2.2. STATISTICAL TECHNIQUES FOR TIME SERIES MODELING

**AIC**

Originating from information theory the *Akaike information criterion* (AIC) estimates the relative quality of a model for a given data set in the form

\[ \text{AIC} = 2k - 2\log(L) \]

where \( k \) is the number of estimated parameters and \( L \) is the maximized likelihood. Thus, AIC assumes that the parameters are estimated with the maximum-likelihood method and balances the complexity of the model \((2k)\) and the goodness of fit \((L)\). Note that \(-\log(L)\) decreases with increasing likelihood \(L\). Preferring the candidate model returning the smallest value the criterion rewards good fits but penalizes the number of parameters used at the same time. Hence, the concept is very similar to the tree pruning criterion \( C_{\alpha}(T) \) in 2.1.2 in trading off goodness of fit against simplicity. The AIC tends to prefer too many parameters \( k \) for small numbers of observations \( N \) and cause overfitting. The corrected criterion \( \text{AIC}_c \) adjusts this bias by adding a second-order parameter penalty \((k^2)\). Hyndman and Athanasopoulos (2014) provide the following corrected formula for a linear model with normally distributed errors,

\[ \text{AIC}_c = \text{AIC} + \frac{2(k + 2)(k + 3)}{N - k - 3}. \]

This in-sample measure for model selection does not evaluate the absolute quality of the model, it compares a model relatively to others of the same kind fitted to the same data set. The model having a lower AIC value has relatively lost less information with respect to the unknown process that generated the data. Further information and alternative criteria as the *Bayesian information criterion* (BIC) are explained in Shumway and Stoffer (2011).

**Prediction Intervals**

Usually, a point estimate \( \hat{y} \) returns a value that does not match with the true future outcome \( y \) of a time series. Two-sided predictive intervals provide lower and upper boundaries such that \( y \) lies within that range with a desired probability \((1 - \alpha)\). The prediction interval for a model with normally distributed errors or *residuals* in the form \( X_{\text{res}} \sim N(0, s^2) \) is given by

\[ \left[ \hat{y} + ts; \hat{y} - ts \right] \]

where \( s \) is the residuals’ empirical standard deviation and \( t \) the standard normal distribution’s \((1 - \frac{\alpha}{2})\)-quantile. For instance, \( t \) equals approximately 1.96 for a 95% prediction interval. Hence, 95% of the residuals are within the range \([-1.96 \cdot s; 1.96 \cdot s]\) which then provides the prediction interval for \( y \). The formula has to be adjusted for point forecasts which are more than one time step ahead. More detail and a distinction from confidence intervals can be found in Shumway and Stoffer (2011), Hyndman and Athanasopoulos (2014), and Hyndman et al. (2008).

2.2.2 Exponential Smoothing

**Basic Concept – Simple Exponential Smoothing**

*Exponential smoothing* (ETS) was introduced by Brown, Holt, and Winters in the late 1950s and turned into one of the most successful forecasting techniques (Hyndman and Athanasopoulos, 2014). ETS is a forecasting method for a time series \( Y \) illustrated in Figure 5 that estimates future values by multiplying past values with relative weights. These weights depend
on a smoothing parameter $\alpha$. The future estimate $\hat{y}_{t+1}$ in the simple exponential smoothing method (SES) is given by

$$\hat{y}_{t+1} = l_t = \alpha y_t + (1 - \alpha) \hat{y}_t.$$ 

The estimate is based on weighting the most recent observation $y_t$ and the most recent forecast $\hat{y}_t$. This weighted sum is also called the level $l_t$. The value $\hat{y}_{t+1}$ at time $t+1$ is expected to be the level $l_t$ which $Y$ has reached at this model at time $t$. The forecast is "flat" and hence SES can only be applied to data without trend and seasonal component. Later we will see how such components are handled. The smoothing parameter $\alpha$ has a value between 0 and 1.

Replacing $\hat{y}$ reveals the recursive structure of the forecast. The equation can be rewritten in the form

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\left[\alpha y_{t-1} + (1 - \alpha) \hat{y}_{t-1}\right] = \alpha y_t + \alpha(1 - \alpha)y_{t-1} + (1 - \alpha)^2 \hat{y}_{t-1}$$

and in a next step by continuously substituting the last term that

$$\hat{y}_{t+1} = \alpha y_t + \alpha(1 - \alpha)y_{t-1} + \alpha(1 - \alpha)^2 y_{t-2} + \ldots + \alpha(1 - \alpha)^{t-1} y_1 + (1 - \alpha)^t \hat{y}_1.$$ 

Thus, $\hat{y}_{t+1}$ is a weighted average of all past observations with exponentially decreasing weights. This property labels the method as exponential smoothing. Recent observations have more weight for larger values of $\alpha$, whereas small $\alpha$ give more influence to time points further in the past. The first value $\hat{y}_1$ and the smoothing parameter $\alpha$ are estimated by minimizing the sum-of-squares error (see 2.2.1)

$$\sum_{t=1}^{T} (y_t - \hat{y}_t)^2.$$ 

Recalling the structure of $\hat{y}_t$ reveals that it contains terms in the form $\alpha^t$ with $t$ greater than one. The yielding non-linear loss function cannot be solved with standard linear least squares anymore but has to be minimized with a non-linear optimizer. Further detail is provided by Hyndman et al. (2008).
Model Variations (ETS)

Decomposing a time series unveils several components. Long term increasing or decreasing of the data is called a trend. This does not necessarily have to be linear and might even reverse from time to time. Seasonal effects influence a series according to a fixed repeating time pattern, for instance the month or the weekday. These properties can also be covered with exponential smoothing after extending the simple model which applies only for data without trend or seasonality. By now we have given forecasts for only one step ahead. In simple exponential smoothing forecasting for a longer horizon does not generate more information.

Since SES forecasts are flat, the value $\hat{y}_{t+h|t}$ forecast for time $t + h$ ($h = 2, 3, \ldots$) at time $t$ is the same as the forecast $\hat{y}_{t+1|t}$ for time $t + 1$ and equals the level at time $t$

$$\hat{y}_{t+h|t} = \hat{y}_{t+1|t} = \hat{l}_t.$$

As mentioned above, SES is not suitable for time series with a trend or seasonal component. This basic model will be enhanced to identify trend so that

$$\hat{y}_{t+h|t} = \hat{l}_t + h\hat{b}_t$$

with $h$ as the forecast horizon. The level component has to be adapted accordingly to the form

$$\hat{l}_t = \alpha y_t + (1 - \alpha)(\hat{l}_{t-1} + \hat{b}_{t-1}).$$

Forecast and level are linearly adjusted with the slope $\hat{b}_t$ of the trend. The trend component $\hat{b}_t$ has the same recursive structure with a new smoothing parameter $\beta$ so that

$$\hat{b}_t = \beta(\hat{l}_t - \hat{l}_{t-1}) + (1 - \beta)\hat{b}_{t-1}.$$

The trend component might be slightly adjusted with a damping factor if it is expected to cease.

Including seasonality requires similar steps. The forecast value $\hat{y}_{t+h|t}$ gets an additional recursive seasonal term in the form

$$\hat{y}_{t+h|t} = \hat{l}_t + h\hat{b}_t + \hat{s}_{t-m+h_m^*}$$

where $m$ is the number of seasons. For example, if the $y_t$ are monthly data with a seasonal pattern that repeats every year, then $m = 12$. The term $h_m^* = [(h - 1) \mod m] + 1$ ensures that the most recent corresponding seasonal observations are used for the forecast. The level component is adjusted so that

$$\hat{l}_t = \alpha(y_t - \hat{s}_{t-m}) + (1 - \alpha)(\hat{l}_{t-1} + \hat{b}_{t-1})$$

while the trend remains the same as

$$\hat{b}_t = \beta(\hat{l}_t - \hat{l}_{t-1}) + (1 - \beta)\hat{b}_{t-1}.$$

The new seasonal component, given as

$$\hat{s}_t = \gamma(y_t - \hat{l}_{t-1} - \hat{b}_{t-1}) + (1 - \gamma)\hat{s}_{t-m},$$

has a similar weighting parameter $\gamma$.

So far, all components had been summed up to model additive properties. Trend, however, could also be non-linear. Think of 5% growth as an example. The trend component would
2.2. STATISTICAL TECHNIQUES FOR TIME SERIES MODELING

then simply be multiplied rather than added up. The same applies for a possibly multiplicative seasonality. Furthermore, considering relative errors $e_t = \frac{y_t - \hat{y}_{t-1}}{\hat{y}_{t-1}}$ instead of additive errors $e_t = y_t - \hat{y}_{t-1}$, also the errors might have multiplicative behavior. As a result, exponential smoothing offers 30 different combinations depending on the problem. The abbreviation (ETS) stands for (Error, Trend, Seasonality) and indicates which kind of model is used for a particular time series. Some examples are:

- (A,N,N) is simple exponential smoothing with additive errors, no trend, and no seasonality,
- (A,A,A) and (A,A,M) are the additive or multiplicative Holt-Winters method with additive errors, additive trend, and the respective seasonal component,
- (M,A_d,N) is the additive damped trend method with multiplicative error and no seasonal component.

Finding the right model will be explained in Section 3.4.2. It should also be pointed out that exponential smoothing is restricted to univariate modeling. Further in-depth analysis of the method can be looked up in Hyndman and Athanasopoulos (2014) and in particular in Hyndman et al. (2008).

2.2.3 ARIMA

Traditional regression models do not have the ability to describe all relevant dynamics of a time series. Box and Jenkins (1970) popularized Autoregressive Integrated Moving Average (ARIMA) as a response. This modeling approach, also known as Box-Jenkins method, combines AR and MA terms for time series forecasting.

![Figure 6: ARIMA time series.](image)

**AR**

As the name already implies, autoregressive models specify that the variable $y_t$ depends linearly on its own previous values in the form

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t$$
2.2. STATISTICAL TECHNIQUES FOR TIME SERIES MODELING

where the constant \( c \) is the bias parameter. Thus, AR terms \( \phi_i \cdot y_t \) consist of a past value \( y_{t-k} \) and a weighting parameter \( \phi_{i-k} \). The error term (or residual) \( e_t \) captures the deviation of the actual value \( y_t \) from the model value \( \hat{y}_t \), which is the forecast estimate

\[
\hat{y}_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p}.
\]

As usual in time series modeling, a basic assumption is that the random errors \( e_t \) are independent and identically normally distributed with mean 0 and a variance \( \sigma^2 \), which is independent of \( t \),

\[
e_t \sim N(0, \sigma^2).
\]

Figure 6 illustrates that \( y_t = \hat{y}_t + e_t \).

MA

A linear moving average model combines weighted previous error terms \( \theta_i \cdot e_i \) in the form

\[
y_t = c + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \ldots + \theta_q e_{t-q} + e_t,
\]

Again, \( c \) is a bias and the error \( e_t \) is the random deviation \( y_t - \hat{y}_t \) of the actual value \( y_t \) from the forecast value

\[
\hat{y}_t = c + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \ldots + \theta_q e_{t-q}
\]

(see Figure 6), and it is assumed that the errors are independent and identically normally distributed. The idea behind this combination of white noise error terms is that these random deviations have lasting impact on future values of the time series. Consequently, for stationary processes, a future estimate \( \hat{y} \) can be based on the previously observed errors.

Differencing

As a prerequisite ARMA modeling expects stationarity of the time series (see 2.2.1). The I (integrated) indicates the process of making a time series stationary with differencing. Trend and seasonal fluctuation affect the value of a time series differently at different times. Differencing may remove these factors to stabilize mean and variance independent of time in the form

\[
y'_t = \Delta y_t = y_t - y_{t-1}.
\]

The differenced time series with new values \( y'_t \) contains the changes between the original data points. Seasonal impact can be adjusted by calculating the seasonal change. For instance, the differenced time series for monthly data with twelve seasons is

\[
y'_t = \Delta y_t = y_t - y_{t-12}.
\]

Differencing may be repeated as required. A process with quadratic growth would have to be differenced twice to obtain stationarity. Furthermore, taking the logarithm is an additional instrument to stabilize the variance. Think of exponentially increasing data. Figure 7(a) shows a time series \( T \) with monthly data points indicating Australia’s electricity consumption over the last fifty years (data taken from Hyndman and Athanasopoulos (2014)) with obvious trend and seasonal patterns. 7(b) shows the stationary time series after taking the logarithm and one round of each seasonal and normal differencing. These steps are simply reversed at the end to rescale the forecast.
2.2. STATISTICAL TECHNIQUES FOR TIME SERIES MODELING

(a) Time series $T$

(b) Stationary time series $T'$

(c) ACF

(d) PACF

Figure 7: Stationarity and autocorrelation of a time series $T$.

Autocorrelation

The autocorrelation function ACF in 7(c) provides insight in the underlying structure of the process by showing the relationship of a time series and its past values. Autocorrelation plots measure the linear relation between a value $y_t$ and lagged values $y_{t-k}$ in the past for different $k$. Correlation is merely a standardized version of the covariance to a scale from $-1$ to $1$. In a time series, $y_t$ and $y_{t-k}$ may be correlated only because $y_t$ is correlated with $y_{t-1}$, $y_{t-1}$ with $y_{t-2}$, and so on. In statistical modeling, we are often interested in the net correlation of $y_t$ and $y_{t-k}$ where the interfering linear effects of the values between them are removed. This quantity is called the partial autocorrelation. More details can be found in Shumway and Stoffer (2011). Figure 7(d) contains the plot of the partial autocorrelation function PACF of the stationarized process. It should be noted that the first spike in Figure 7(c) is at lag 0, whereas in Figure 7(d) at lag 1. A value $y_t$ has a autocorrelation coefficient of 1 with itself but you cannot determine a partial autocorrelation. Accordingly, as expected, spike 2 of the ACF is quantitatively the same as spike 1 of the PACF since they measure the correlation between consecutive values (note the differing scales on the axes in figures c) and d)). The blue lines indicate a critical value whether $y_{t-k}$ is significantly correlated to $y_k$. Correlations below the threshold are considered to be random and therefore nonsignificant. The blue lines indicate the two-sided 95% confidence interval of a white noise process with zero mean. In
other words, uncorrelated terms \( y_{t-k} \) would fall between the thresholds with a probability of 95% as they are white noise. Thus, ACF or respectively PACF values above the line imply significant correlation and should therefore be considered for the model. These boundaries for statistical significance for a time series of length \( N \) are
\[
\pm 1.96 \times \frac{1}{\sqrt{N}}
\]
since the term \( \frac{1}{\sqrt{N}} \) is the ACF’s standard deviation of a white noise process of length \( N \) and 1.96 the quantile containing 97.5% (1 - \( \frac{\alpha}{2} \) where \( \alpha = 1 - 0.95 \)) of the values. Shumway and Stoffer (2011) and Hyndman and Athanasopoulos (2014) evaluate this further. Analyzing the autocorrelations of a stationarized time series allows presumptions about the possible combinations of AR and MA terms of the forecasting model.

**ARIMA**

A mixed model of both autoregressive and moving average terms can be written as
\[
y_t = c + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + \theta_1 e_{t-1} + \ldots + \theta_q e_{t-q} + e_t.
\]
The parameters are estimated with maximum likelihood (see 2.2.1) using the whole time series. An ARIMA model is denoted in the form \((p,d,q)\). The variable \( p \) denotes the number of AR terms, \( d \) the number of differencing steps required, and \( q \) the number of MA terms.

To adapt to seasonal effects, additional seasonal terms can be included. Consider as an example the electricity consumption in Australia, see Figure 7. The following model in the form
\[
y_t = c + \theta_1 e_{t-1} + \theta_{12} e_{t-12} + \theta_{24} e_{t-24} + e_t
\]
turned out to be a good candidate to predict future electricity demand in Australia. Seasonal models contain AR and MA terms with \( m \cdot 1, m \cdot 2, \ldots \) lags, if \( m \) is the number of seasons, i.e., if the seasonal pattern repeats every \( m \) time points. A seasonal ARIMA model gets denoted in the form \((p,d,q)(P,D,Q)[m]\). The second tupel indicates the seasonal elements and the number in brackets tells the corresponding seasonal lag. Recalling the initial differencing to make the time series stationary (one seasonal and one normal differencing) the Australia electricity consumption model would be denoted as ARIMA\((0,1,1)(0,1,2)[12]\). The approach to find the right model will be explained in the next chapter in Section 3.4.3.

**Multi-horizon Forecasts**

Given the necessary input values ARIMA models make point forecasts one time step into the future. For instance, a model consisting of each two AR and MA terms estimates a future value \( y_{t+1} \) in the form
\[
\hat{y}_{t+1} = c + \phi_1 y_t + \phi_2 y_{t+1} + \theta_1 e_t + \theta_2 e_{t-1}.
\]
All estimated parameters \( \phi \) and \( \theta \) stay the same to make a prediction two time steps ahead. However, all indices \( t \) are substituted by \( t + 1 \) so that
\[
\hat{y}_{t+2} = c + \phi_1 y_{t+1} + \phi_2 y_{t+2} + \theta_1 e_{t+1} + \theta_2 e_t.
\]
The unknown value \( y_{t+1} \) gets substituted by the most recent estimation \( \hat{y}_{t+1} \) and the unknown error \( e_{t+1} \) is simply set to zero. This procedure can be repeated as desired (Hyndman and Athanasopoulos, 2014).
ARIMAX

Regular ARIMA models are limited since they only allow to include information from the past observations of the time series. It is not possible to involve further relevant information that correlates with the series. For instance, modeling consumption behavior might correlate with income figures. Dynamic regression or ARIMAX models extend univariate ARIMA models to include external knowledge.

A linear model $y_t$ with $k$ predictor variables in the form

$$y_t = \beta_0 + \beta_1 x_{1,t} + \ldots + \beta_k x_{k,t} + e_t$$

where $e_t$ is white noise is a standard multiple regression. By contrast, ARIMAX allows the error term to have autocorrelation and assumes that the error follows an ARIMA model. The new model is then given by

$$y_t = \beta_0 + \beta_1 x_{1,t} + \ldots + \beta_k x_{k,t} + n_t$$

where $n_t$ denotes the ARIMA model which relates to the output variable. The ARIMA model $n_t$ would then of course contain a white noise error term $e_{t,\text{arima}}$. Model parameters are estimated by minimizing the sum-of-squared $e_{t,\text{arima}}$ errors. The ARIMA error structure can obviously only be calculated when $n_t$ is known. However, $n_t$ can only be determined if the $\beta$ parameters are known. This deadlock is resolved by choosing a preliminary ARIMA model that covers most autocorrelation to estimate the $\beta$ coefficients. The $\beta$ parameters provide new values for $n_t$ which enables selecting a better ARIMA model. Obtaining this new model allows refitting the entire model. All variables have to be stationary and all get differenced if any needs to be stabilized. Thus, ARIMAX allows multivariate forecasting with a combination of predictor variables and the output variable $y$. Further reading is provided by Shumway and Stoffer (2011).

2.3 Prior Work

Regarding the context Chopra and Meindl (2016) and Christopher (2016) provide an insight into supply chain management by giving a thorough overview of the field. Both books cover the topic of forecasting in their domain.

Time series forecasting with exponential smoothing methods goes back to the end of the 1950s to Winters (1960), Brown and Meyer (1961), and Holt (1957). Their techniques are still widely applied in various fields as economics due to their ability to deliver reliable forecasts and being simple and transparent at the same time (Goodwin et al., 2010).

As previously mentioned ARIMA models for time series prediction had been invented by Box and Jenkins (1970). Forecasting with ARIMA models is generally very established for economical and financial time series (Armstrong, 2001). For instance, Conejo et al. (2005) propose a new way to predict future electricity prices based on ARIMA models.

The emergence of neural networks and deep learning offered new possibilities for predictive time series models. Frank et al. (2001) analyzed the performance of feedforward neural networks on time series with traffic data. More recently, applying recurrent networks (RNN) which take continuous sequences as an input has become more popular for time series forecasting. Especially using Long Short-Term Memory (LSTM), a method introduced by Hochreiter and Schmidhuber (1997). Gamboa (2017) discusses RNNs and LSTM as methods to predict time series. Bandara et al. (2017) show that RNNs outperform other state-of-the-art forecasting methods on various data sets. Yet, this approach was not considered for this thesis due to the very small amount of data available (see 3.3.2).
A study by Krauss et al. (2017) compares the performance of neural nets, gradient boosted trees and random forests to predict the stocks in the S&P 500. The random forest returns the best predictions without requiring exhaustive hyperparameter optimization as for the neural networks. However, combining all methods to a simple ensemble by averaging the results outperforms all other approaches. Similarly, Khashei and Bijari (2011) discover an improvement of predictive performance by combining different models.
Chapter 3

Methods

3.1 Description of the Forecasting Task

Operating just-in-time imposes a great challenge for the supply chain management (SCM). Each X-ray system is processed individually and customer specific in parallel alignment with the customer’s preparation of the installation site. The whole lifecycle of an order is depicted by the blue boxes in Figure 8. Each incoming order for a new system goes into a pool of unprocessed items. All these orders have the temporary status backlog. As soon as the construction on the customer’s side has reached a certain progress, the SCM fixes a date to assemble the new X-ray machine and the order’s new status becomes scheduled.

![Order lifecycle diagram](image)

Issues with the preparation of the examination rooms can often cause delays on short notice. The assembly has to be postponed and the order gets put back into the backlog. Eventually,
3.1. DESCRIPTION OF THE FORECASTING TASK

An order gets manufactured and delivered in only a short period of time. Then, after the selling has been finalized the order gets archived and classified in completed sales.

However, the uncertainty when an order has actually to be manufactured makes precise planning in advance impossible. Nevertheless, resource allocation for manufacturing has to be planned thoughtfully ahead to minimize production costs. False estimates could either cause a shortage or an abundance of employees. Materials from suppliers have to be in stock, yet, with minimal storage cost. Moreover, production slots should neither be unused nor lacking. Knowing the number of future sales for a month allows conclusions about the number of starting assemblies since the time period of assembling and completing the deal is usually a fixed period of \( n \) days. Good estimates about the sales figures could potentially optimize the workflow and reduce costs. Thus, the core goal of our work is to design and implement good predictors of future values of sales. The time series that we study consist of monthly data. The numbers \( s_i \) of sales (and the other data like the numbers of backlogged and scheduled systems) are available at the turn of the month. Time point \( i \) represents the beginning of a month (e.g. 01.05.18) and \( s_i \) is the number of sales in the previous month.

The forecast horizon derives from practical reasons. Figure 9 reveals that forecasts have to be made two months into the future. Predicting the number of sales for December as in the red box on December 1 would be too late to allocate resources for that month. Hence, forecasts have to be done one month earlier as shown in the green box. In this example, the sales prediction for December would have to be made on November 1. Consequently, as indicated by the blue line, the forecast would only rely on data at the latest from October.

![Figure 9: Forecast interval.](image)

The four methods explained in the previous chapter, a fully connected neural net, a random forest regression, exponential smoothing, and ARIMA, are applied to make these forecasts. Our approach will be explained in detail in this chapter. Ideally, the resulting forecast models map the underlying process that generates future sales figures. In other words, the models must generalize well for reliable forecast estimates. To verify the methods the data is split into a training set for fitting the models and a hold-out test set to measure out-of-sample performance. The different approaches for the various algorithms to split the data will be explained in the respective upcoming sections. Figure 10 illustrates the trade-off between bias and variance to achieve the best generalization error on the test set. Thus, the dashed vertical line indicates the perfect model complexity. The bias measures the systematic error between prediction and the desired value of the regression function. Variance, however, indicates a model’s sensitivity to different samplings of the training set. High bias and low variance can cause underfitting by missing relevant information in the data. By contrast, low bias and high variance could cause overfitting due to modeling the random noise in the training data. The
regularization measures explained in the previous chapter have been applied to find models with the best ability to generalize. The performance of all algorithms will be measured on a hold-out test data by their relative predictive error MAPE (mean absolute percentage error).

3.2 Data

3.2.1 Data Set

The historic data consists of a monthly census containing all items with status backlog, scheduled, and sales. The data structure for \( N \) data points is illustrated in Table 1. Backlog and sales data consist of one number only, telling the total number of orders in the backlog pool and the number of completed sales in that month. The data concerning the scheduled assemblies is structured differently. Each data point includes the preliminary schedule of upcoming assemblies for the following three months.

<table>
<thead>
<tr>
<th>( t )</th>
<th>backlog</th>
<th>sales</th>
<th>scheduled</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( b_1 )</td>
<td>( s_1 )</td>
<td>( (c_{1,1}, c_{1,2}, c_{1,3}) )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( b_2 )</td>
<td>( s_2 )</td>
<td>( (c_{2,1}, c_{2,2}, c_{2,3}) )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( t_N )</td>
<td>( b_N )</td>
<td>( s_N )</td>
<td>( (c_{N,1}, c_{N,2}, c_{N,3}) )</td>
</tr>
</tbody>
</table>

Table 1: Data structure \( d \).

The example in Figure 11 shows an individual data point with hypothetical numbers. The entry for October 2002 tells that there are 300 unprocessed orders waiting in the backlog and that 100 sales have been completed in that month. Moreover, 80 systems are scheduled to be assembled in November, 50 in December, and 20 in January. Hence, each data point has 5
3.2. DATA

dimensions or features. The supplied dataset covers the last fifteen fiscal years from October 2002 to September 2017. Consequently, the total size $t$ of the dataset used to predict future sales figures is

$$15 \cdot 12 \text{ months} = 180 \text{ datapoints.}$$

Moreover, the data does not only provide the total number of all X-ray systems combined but also reflects specifications about the type of machine in separate tables. The five subtypes are MP, Floor, Ceiling, Zego, and Biplane. Altogether, the available data consists of numbers for all systems, grouped and individual, according to the structure in Table 1. Talking about all systems grouped together as a single quantity will be referred to as Total and individual device types as Single device.

### 3.2.2 Missing Data

Some data points contain missing values that have to be dealt with before modeling. All data points containing missing values could be discarded. Yet, dealing with a continuous time series eliminates that option. Ignoring the missing values by setting them to zero could create models that are heavily influenced by these false outliers. Hence, all missing values are filled with reasonable estimates. This step is called imputation (Goodfellow et al., 2016).

The sales and backlog data is complete, whereas some information misses for the scheduled assemblies. Yet, the percentage of missing data is very small with 0.019% regarding the Total data and 0.02% of all Single device data. The imputation algorithm bases on the assumption of a linear dependence between the scheduled assemblies and the actual sales figures and the simple tactic to impute missing values with the mean of existing values. A scheduled assembly refers either 1, 2, or 3 time steps into the future. The average ratio between scheduled and sales for each forecast horizon is given by

$$\text{ratio}_j = \left( \frac{\sum_{t=1}^{n} c_{t,j}}{s_{t+j}} \right) \cdot \frac{1}{n}$$

where $j \in [1, 2, 3]$ denotes the number of corresponding time steps ahead. The variable $n$ is the number of all available data pairs of scheduled assemblies $c_{t,j}$ at time $t$ and actual sales $s_{t+j}$ at time $t + j$ where $s_{t+j} \neq 0$. Accordingly, all missing values $c_{t,j,\text{missing}}$ at time $i$ are replaced in the form

$$c_{t,j,\text{missing}} = s_{t+j} \cdot \text{ratio}_j.$$  

This procedure does not work to replace missing values where $t + j$ exceeds the boundaries of the time series. However, these border cases do not occur in the data. As of now, after imputation, all missing values are treated like real data as if they had been observed. Using average as a component of imputation is proposed, among other methods, by Hastie et al. (2009).
3.2.3 Correlation of Features

Analyzing the correlation between the five features backlog $b_t$, sales $s_t$, and scheduled $(c_{t,1}, c_{t,2}, c_{t,3})$ provides insight about the data (Total). Correlation coefficients close to 1 or $-1$ imply high correlation whereas values close to zero suggest small correlation. The scatter plots in Figure 12 show the linear correlation between the output variable sales and the other predictor variables. Plot 12(a) reveals no linear dependency between backlog and sales at time $t$. Still, the distribution indicates a positive correlation. In fact, the correlation coefficient between the two variables is 0.47. However, the backlog might affect the sales with some lag since orders might remain in the backlog pool for an uncertain period of time, but the analysis of $b_{t-x} \sim s_t$ for $x = 1, 2, \ldots, 12$ delivers similar plots and correlation coefficients. This makes actually sense as all orders are not assembled and delivered exactly $x$ months after being ordered. Some are immediately manufactured whereas others remain unprocessed in the waiting line for a long time. Yet, lagging the backlog for two months, $b_{t-2} \sim s_t$, has the highest correlation 0.52. The other plots in Figure 12 compare scheduled assemblies with actual sales figures. Plot (b) unveils an almost linear relationship between the scheduled assemblies for the next month $c_{t,1}$ and the real sales numbers of that month $s_{t+1}$. Accordingly, the correlation coefficient 0.96 has a value very close to one. Thus, a linear regression would deliver an almost optimal solution for a forecast model. However, recalling the required forecast horizon of two...
months eliminates this option. The plots (c) and (d) reveal increasing uncertainty about total sales if relying only on the schedule with respective correlation coefficients of 0.70 and 0.19. The sudden drop between (c) and (d) implies that systems get scheduled only shortly before manufacturing (≤ 2 months ahead).

### 3.3 Machine Learning Techniques

In this section we discuss our way of applying the machine learning methods, ANN and random forests, to the forecasting problem. So all details on data preparation and so on refer only to the machine learning approach. The statistical approach is then described in Section 3.4.

#### 3.3.1 Data Preparation

**Additional features**

The data structure from Table 1 gets extended with two additional features for the neural net and the random forest. First, each data point receives a **month** tag $m_1, m_2, \ldots, m_{12}$. The reason will become evident in the upcoming Section 3.3.2 about the algorithms’ input and output. Moreover, when dealing with different system types in the **Single device** data each data point gets a **type** ID $t_1, t_2, \ldots, t_5$. These IDs are implemented with dummy variables as a one-hot vector because they are categorical rather than numerical. One-hot vectors have $n$ bits of ones and zeros that are mutually exclusive ($n = 5$ in our case). In other words, only one of them can be active. This conversion is better than simple numbering because it avoids the effects of numerical ordering. More information about encoding can be found in Goodfellow et al. (2016) and Hastie et al. (2009).

**Data augmentation**

The training set can been be expanded by creating synthetic data structures based on the existing data. Therefore, the original data $d_{org}$ gets diversified according to the following augmentation schemes to obtain new data $d_{new}$. These newly obtained data sets are then used in addition to the original to train the machine learning algorithms. Of course, this option would not be applicable for the statistical forecasting methods.

First, adding random white noise to the time series creates new samples. Each data point’s values get multiplied with a random number following a white noise distribution with mean 1 and standard deviation 0.03. The values were chosen to remain in a realistic range considering the error deviation. Adding noisy training data makes the model more robust.

Second, masking the series with a trend produces new synthetic data. That trend $g$ for a new time series given by

$$g_i = g_0 \cdot (1 + b)^i$$

is exponential to simulate natural positive or negative growth. The parameter $i$ indicates the time series’ index. The growth rate $b$ is randomly chosen between $-0.004$ and $0.004$ from a uniform distribution. That range was chosen to remain in a plausible range for 180 data points after adding the trend. Each value of the original time series gets multiplied with the corresponding trend according to its index (remark: $t_1$ has index 0 here). $g_0$ is chosen to be 1 so that the first value of the time series remains the same. **Scheduled** features $c_{i,j}$ at index $i$ reference either 1, 2, or 3 time steps into the future. This is considered by multiplying $c_{i,j}$ with $g_{i+j}$. 

26
Third, decomposing the original time series in its components allows seasonal augmentation. The seasonal-trend method STL by Cleveland et al. (1990) decomposes a time series into a trend, a seasonal, and a remainder component. The R forecast package by Hyndman et al. (2017) provides that functionality and returns the average impact for each season. We distinguish 12 seasons, one for each month. Then, this seasonal component gets subtracted from the series, shifted, and added up again. This is done separately in parallel for backlog, sales, and each scheduled feature. Thus, the new data structure retains the original underlying process but simulates shifted seasonal impact.

The noise and trend augmentation can be repeated as desired. By contrast, seasonal augmentation is limited to shifting eleven times before repetition since the year has only twelve months. The new features month and type remain obviously untouched in the new synthetic data structures. Enlarging the training data might improve the models’ generalization ability. We considered three options to augment our data set for the machine learning experiments which can be seen in table 2.

<table>
<thead>
<tr>
<th>Noise augmentations</th>
<th>Trend augmentations</th>
<th>Seasonal augmentations</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Middle</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>High</td>
<td>5</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 2: Data augmentation options.

Normalization

Among many machine learning algorithms, a neural network requires standardization as a preprocessing step (Pyle, 1999). Accordingly, all features get rescaled to obtain the properties of a standard normal distribution with zero mean and unit variance \( X \sim \mathcal{N}(0,1) \) as a preprocessing step. Each feature will be normalized separately due to their partly huge difference in range. That can be achieved by subtracting the variable’s mean \( \overline{x} \) and division by the standard deviation \( \sigma \) in the form

\[
x' = \frac{x - \overline{x}}{\sigma}.
\]

Basically, feature scaling prevents the neural net to use one feature as its main predictor only because it is bigger. Normalization ensures that all variables get treated equally when training a model. Weights of a neural network are initialized to the same range. Dominating variables would cause dominating weights during training and consequently a biased error surface of the loss function. That distortion prevents direct steps towards a minimum and slows down convergence. Thus, data normalization speeds up backpropagation significantly. Moreover, large values would cause gradients getting very close to zero and with that also slower training. Further reading in particular regarding deep networks is provided by Ioffe and Szegedy (2015).

Sample Generation

Training a FNN and a random forest requires many data samples consisting of pairs of inputs and outputs. The data structure is split into sequences by sliding over the time line according to the principle of a sliding window. This technique has also been applied by Le Guennec
et al. (2016). Figure 13 as an example illustrates that method for a window of size 6. Sliding over the time line creates sequences of length 6 which are incrementally shifted by one data point.

![Figure 13: Moving window.](image)

Each of the sequences is then split into two parts. The first four months are the input for predicting the fifth and sixth month. Thus, instead of using the time series as a whole it gets split into many overlaying subsamples. The obtained set of sequences serves as the data set for the neural net and the random forest.

### 3.3.2 Setup

#### Input and Output

The actual sample length of the moving window is a hyperparameter and set either to 4 or 14. Figure 14 shows an example for one sequence which is split into an input part, representing the past, and a future output. The window length in the example is 14. In this case, the size of the input interval is chosen to be 12 months to cover the cycle of a whole year. Two time points as the output meets the requirement of predicting two months into the future.

Yet, not all information of the newly created sequences is of interest. The first 12 data points for the input keep the entire backlog $b_1, b_2, \ldots, b_{12}$ and sales $s_1, s_2, \ldots, s_{12}$. However, they retain only the most recent scheduled information $c_{12,1}, c_{12,2}, c_{12,3}$ which references the forecast interval. Moreover, one 12 dimensional one-hot vector $m_{12,1}, m_{12,2}, \ldots, m_{12,12}$ suffices to distinguish exactly which months are going to be predicted. Dealing with Single Device data would require keeping an additional one-hot vector containing the encoding of the system type information. The output contains only the variables sales $s_{13}$ and $s_{14}$.

![Figure 14: Sequence sample (Total data).](image)
The example shows that only two variables would be kept from January to November compared to 17 from December to predict the February sales of the next year. For a sequence length of 14, the neural net and the random forest map a 39 dimensional input to two output variables in the form

\[ f(b_1, b_2, \ldots, b_{12}, s_1, s_2, \ldots, s_{12}, c_{12,1}, c_{12,2}, c_{12,3}, m_{12,1}, m_{12,2}, \ldots, m_{12,12}) = (s_{13}, s_{14}). \]

In fact, all 39 variables get zipped together as a single input vector. The same applies for both output variables.

By contrast, having a window length of 4 simply means a shrinkage of the input regarding backlog \( b \) and sales \( s \). Instead of considering 12 months for the input and 2 for the output we take only 2 for both input and output. Thus, we map a 19 dimensional input to two output variables in the form

\[ f(b_{11}, b_{12}, s_{11}, s_{12}, c_{12,1}, c_{12,2}, c_{12,3}, m_{12,1}, m_{12,2}, \ldots, m_{12,12}) = (s_{13}, s_{14}). \]

Hence, in our example in Figure 14 we would make a forecast for February 13 only with the November and December data from the previous year.

**Training and Testing Split**

After augmentation the dataset consists of the original time series data \( d_{org} \) and potentially many synthetic data frames \( d_{new} \) of the same structure. Each data structure consists of a time series containing the backlog, sales, and scheduled information as shown in Table 3.2.1. The last 20% of the data points of all data frames are cut off and used for testing. However, only the data points of the original data \( d_{org} \) are preserved for a test set whereas the rest gets discarded. Thus, the most recent three years of the original time series will be taken to evaluate the forecast performance of the algorithms. The remaining 80% of the possibly augmented data are then used to perform a 5-fold cross-validation. Five times the data gets split into non-overlapping training and validation sets so that all data points will have been used for validation once. Each of the time series (backlog, sales, scheduled systems) is split at the same time points for each fold to prevent temporal intersections. The validation sets will not contain any augmented data either by reducing them to the original data as well. In order to determine the validation error, the performance on the different validation partitions is then simply averaged. We create the previously described input sequences for the algorithms with the moving window approach (see 3.3.1 and 3.3.2) always after splitting the data to avoid leaks between training, validation, and test data. The cross-validation performance was taken to determine the hyperparameter settings of the algorithms and the test error to measure its actual performance. The small amount of original data (180 data points) made this approach with a cross-validation and an extra hold-out test set necessary. A single test set without cross-validation would have implied a high statistical uncertainty about the estimated error considering the small amount of data. Also, tuning the algorithms to get the best cross-validation performance returns a biased model that might underestimate the generalization error. Hence, the models have to be evaluated with another set of completely unseen test data for a realistic evaluation (see Goodfellow et al. (2016)). The approach explains also the chosen sequence length of 14 as an upper boundary. The hold-out test set consisting of 36 data points allows generating 23 sequences for this window size. The number of validation samples for each cross-validation fold is already only 15. Depending on the validation partition, training samples vary between 73 and 102 (data without augmentation). Hence, longer sequences would shrink the data sets even more and render this approach impracticable. Reducing the

\[ 29 \]
3.3. MACHINE LEARNING TECHNIQUES

window size to 4 to generate sequences increases the training, validation, and test set by 10 samples.

Dealing with the Single Device data follows the same pattern. The idea is to develop one model with the neural net and the random forest that can be used to make sales predictions regardless of the type. We assume that the sales figures of all five system types follow a similar underlying process. In other words, we train one model with all the data of the different types. Omitting a distinction between the types provides five times more training data. The data of each type may be augmented separately and then split according to the previously described approach. The respective type-specific training, validation, and test splits are then combined for further proceeding. Consequently, this model allows forecasts regardless of the system type. Training five different models for each type would have also been possible but was not considered for this thesis.

In all cases, assuming normally distributed errors allows providing a prediction interval for future estimates by taking the averaged standard deviation of the mean absolute percentage test error (MAPE).

3.3.3 FNN modeling

The settings to control the behavior of a learning algorithm are called hyperparameters. As previously mentioned a cross-validation is used to determine these settings. In other words, the hyperparameters of the neural net get adjusted after training to find the lowest error on the unobserved validation set examples. All activation functions in the network are sigmoid functions. The error function to be minimized during training is given by

$$\frac{1}{2} \sum_{i}^{N} (y_i - \hat{y}_i)^2 + \frac{\lambda}{2} w^T w$$

where \(N\) is the batch size and \(w\) the network’s weight vector (see 2.1.1). The batch size of training samples for each update of the weight vector was set to 50. The number of hidden layers was varied with varying numbers of hidden units to find the optimal complexity. Our notation \([20, 10, 5]\) means, for example, 3 hidden layers with 20 units in the first hidden layer, 10, in the second, and 5 in the third. The subsequent parameters in Table 3 were optimized with a grid search over the hyperparameter space for each network topology. Thus, a model was trained for each possible hyperparameter combination and ranked by its validation error. Performing a grid search is a common approach for a small set of variables. Increasing the number of layers and units increases the network’s capacity. Adding regularization as L2 loss or dropout has the opposite effect. Often, large neural nets with appropriate regularization return the best models considering the generalization error (Goodfellow et al., 2016). The exponential decay hyperparameter \(\beta\) refers to repeatedly reducing the learning rate \(\mu\) after each \(m\) steps for quicker convergence in the form

$$\mu_d = \mu \cdot \beta^m,$$

where \(r\) is the number of executed steps (i.e. \(r = m, 2m, \ldots\)). The parameter \(m\) was chosen to be the maximum number of training steps divided by 5. The maximum number of training steps was set 7500 for the Total data scenario and to 20000 for the Single device data scenario. Moreover, dropout has always been applied to all hidden layers with the same rate. Tuning these parameters required observing the evolution of the error during training for both the training and the validation set. Ideally, as Figure 10 shows, the training process should
3.3. MACHINE LEARNING TECHNIQUES

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate $\mu$</td>
<td>[0.0001, 0.0005, 0.001]</td>
</tr>
<tr>
<td>Exponential decay $\beta$</td>
<td>[0.86, 1.0]</td>
</tr>
<tr>
<td>Dropout</td>
<td>[0.0, 0.2, 0.4]</td>
</tr>
<tr>
<td>L2 regularization $\lambda$</td>
<td>[0.0, 0.0005, 0.001]</td>
</tr>
<tr>
<td>Augmentations</td>
<td>[none, middle, high]</td>
</tr>
<tr>
<td>Early stopping</td>
<td>[True, False]</td>
</tr>
<tr>
<td>Sequence length</td>
<td>[4, 14]</td>
</tr>
<tr>
<td>Topology</td>
<td>[[20], [20, 10], [20, 10, 5], [20, 15, 10, 5]]</td>
</tr>
</tbody>
</table>

Table 3: Hyperparameters NN.

return the lowest error on the validation data before overfitting to the training samples. Monitoring this evolution for the Total experiments showed quick overfitting regardless of the network’s complexity. Hence, one option was to stop the training process automatically (early stopping) when the relative error on the training data reached $\pi = 8\%$ (Total) respectively $\pi = 15\%$ (Single Device). These thresholds were chosen after analyzing experiments for various parameter settings and refer to the combined relative error of both future outputs. Choosing the training error instead of the validation error reduces the models’ bias.

Finally, the hyperparameters for each network topology get selected based on the lowest relative validation error (MAPE). Ultimately, the test set errors (MAPE) for the final settings get evaluated in order to measure the generalization abilities of the models. This grid search was performed twice, for the Total and Single Device scenario.

3.3.4 Random forest modeling

The parameter settings for the random forest are significantly more limited. Nevertheless, finding the best settings from Table 4 with a grid search based on a cross-validation provides again a slightly biased model. Thus, the random forest performance gets evaluated in the exact same way as the neural net with an extra hold-out test set. All trees of the forest are grown with the CART algorithm (see Section 2.1.2). The size of the bootstrap subsample is the same as the original input size. Despite retaining the original size the bootstrap subsample is different every time since the training examples are drawn randomly with replacement. Thus, a training example maybe drawn multiple times in one bootstrap. The tree with the lowest validation error gets selected and its corresponding test error serves for the evaluation. Again, the grid search is performed twice, for the Total and for the Single Device scenario.
3.4 Statistical Techniques

In this Section we discuss our approach of applying the statistical techniques, exponential smoothing and ARIMA, to the forecasting problem.

3.4.1 Setup

Input
Both univariate statistical methods, ETS and ARIMA, require choosing a model in the first place. Then, either algorithm takes the whole consecutive time series with all past observations of the output variable $s_1, s_2, \ldots, s_t$ as a single input vector to estimate the model parameters.

By contrast, applying dynamic regression (ARIMAX, see Subsection 2.2.3) involves several variables and consequently multidimensional input. Our previous analysis of correlations between the sales $s_t$, backlog items ($b_t$) and scheduled systems ($c_{t,1}, c_{t,2}, c_{t,3}$) (see Subsection 3.2.3) suggests to augment the ARIMA estimate with the following regression on $c_{t-1,2}$ and $b_{t-3}$:
\[
\hat{s}_{t+1} = \beta_0 + \beta_1 b_{t-3} + \beta_2 c_{t-1,2} + n_t,
\]
with $n_t$ the ARIMA part of the estimate
\[
n_t = \phi_0 s_t + \phi_1 s_{t-1} + \ldots + \phi_{p-1} s_{t-(p-1)} + \theta_0 e_t + \theta_1 e_{t-1} + \ldots + \theta_{q-1} e_{t-(q-1)} + e_t.
\]

The extra regression for our time series with $p$ data points is illustrated in Figure 15. The values of $c_{t-1,2}$ refer to the sales $s_{t+1}$ two time points later. In fact, our previous analysis has revealed a correlation coefficient of 0.70 between the two variables. Furthermore, individual orders on average remain 4.3 months in the backlog. Hence, we assume that both of these variables influence sales with a lag of 2 respectively 4. The value $\hat{s}_{t+2}$, which is also forecast at time $t$, is estimated in the same way, the actual values $s_{t+1}$ and $e_{t+1}$, which are not known at time $t$, are replaced by the estimate $\hat{s}_{t+1}$ and 0.

Training and Testing Split

Time series models resulting from ETS or ARIMA have certainly to be further evaluated with out-of-sample test data to assess their performance. Yet, the approach is different to the cross-validation explained in Section 3.3.2. We use a rolling forecast (see Hyndman and Athanasopoulos (2014)) to obtain a reliable estimate about the goodness of each method for our particular forecasting problem. We start with the first 121 data points of our time series and take 119 points $s_{t-118}, s_{t-117}, \ldots, s_t$ to estimate the parameters of an ETS or an ARIMA.
3.4. STATISTICAL TECHNIQUES

model. Then, we compare the models’ prediction for \( \hat{s}_{t+1} \) and \( \hat{s}_{t+2} \) with the actual values \( s_{t+1} \) and \( s_{t+2} \). Again, in a next step we take 122 data points, estimate a model and calculate the loss. We repeat that until we have rolled to the end of the time series. The forecast accuracy will then be computed by averaging the errors of our 60 samples. Thus, we do not evaluate a single model but rather how appropriate either method is for the given time series data. Also, we estimate separate models for each system type regarding the Single Device data unlike the machine learning algorithms.

3.4.2 ETS Modeling

Forecasting with statistical methods begins with visualizing the given data. Figure 16 (a) plots the historic sales data \( s_t \) (Total) from October 2002 to September 2017. Plots (b) and (c) show the corresponding ACF and PACF. Similarly to the previous chapter, plot (b) of the ACF starts at lag 0 whereas plot (c) of the PACF starts at lag 1. Plotting the time series allows conclusions about possible ETS models. Figure 16 (a) indicates a ceasing trend component. Moreover, the partial autocorrelation function in (c) shows a significant spike at lag 12. That suggests an annual influence based on the month of year. Other lags which do not match with the year are unlikely to make interpretable sense and should be ignored regarding seasonality (Hyndman and Athanasopoulos, 2014). Thus, the model candidates should contain a trend and a seasonal component, probably with additive behavior. However, usually, the relatively small number of 30 possible model combinations (see Section 2.2.2) and software support allow estimating the parameters for all combinations. The final method gets selected among the group of fitted models according to the best AIC\(_c\) value (see 2.2.1).

3.4.3 ARIMA Modeling

Building an ARIMA model \((p, d, q)\) consists of several steps. Figure 17 shows the algorithm to obtain an appropriate model. The data has to be stationarized in a first step. Then, a promising number of either \( p \) AR and \( q \) MA terms has to be chosen. Hyndman and Athanasopoulos (2014) provide the following rule to determine these parameters:

- Exponential or sinusoidal decay of the ACF and a significant spike at lag \( p \) but none beyond in the PACF suggest a model with \( p \) AR terms.

- Exponential or sinusoidal decay of the PACF and a significant spike at lag \( q \) in the ACF but none after imply a MA model with \( q \) MA terms.

This recommended procedure provides a pure AR or MA model for a start. Choosing the right terms is also thoroughly explained by Shumway and Stoffer (2011). The parameters of the
chosen model will then be estimated with maximum likelihood (see 2.2.1) using all observed values of the time series. Subsequently, the model gets evaluated with the AICₖ (see 2.2.1). Ultimately, the residuals of the model are checked to be white noise. Otherwise there would still be information left to be included in the model. Usually, it is not possible to determine
3.5. TECHNICAL SETUP

Data: Time series consisting of \( t \) observations \( \{y_0, y_1, y_2, \ldots, y_{t-1}\} \)

Result: ARIMA \((p, d, q)\) model

1. Remove trend and seasonality of time series by differencing \( d \) times (stationarity);
2. Choose number of \( p \) AR or \( q \) MA terms ← Analyze autocorrelation (ACF & PACF) of time series;
3. repeat
   4. Estimate candidate model parameters \((\phi, \theta)\) with maximum likelihood;
   5. Calculate AIC;
   6. if Residuals are white noise then
      7. Store model
   8. else
      9. Discard Model
   10. end
11. Adjust \( p \) and \( q \) to find variations of the model with lower AIC;
12. until Sufficient number of models have been evaluated;
13. Pick model \((p, d, q)\) with lowest AIC value

Figure 17: ARIMA modeling approach.

the best model by analyzing the plot. Thus, the process gets repeated with variations of \( p \) and \( q \) to determine the best model heuristically. These variations may then be mixed models of AR and MA terms.

Plot 16 (d) of our sales data results after making the process stationary with constant mean and constant autocovariance between any two time points apart the same distance (see again 2.2.1). Among others, the Augmented Dickey-Fuller Test (ADF) checks stationarity and indicates with an estimate if a time series meets the standards or requires further differencing. A precise explanation of ADF can be found in Hyndman and Athanasopoulos (2014). The significant spike of the partial autocorrelation function of the original series at lag 12 in Figure 16 (c) indicates significant annual correlation. Thus, our time series has been differenced once seasonally with \( m = 12 \) beside normally differencing it once. The two plots of Figure 16 (e) and (f) show the ACF and PACF of the stationary series. Our sales time series suggests choosing 1 regular MA term to begin with. The PACF decays exponentially and the last significant spike in the ACF before it drops is at lag 1. Moreover, the significant spike at lag 12 in the ACF suggests adding this MA term. Choosing seasonal terms applies to the same rules as choosing regular terms (see Hyndman and Athanasopoulos (2014)). Thus, our initial model would be an ARIMA(0,1,1)(0,1,1)[12]. Finally, the residuals have to be checked again to be white noise. Eventually, after varying the number of terms, an ARIMA(0,1,1)(0,1,2)[12] had the lowest AIC value and would have been chosen as the most appropriate model for the whole sales time series.

3.5 Technical Setup

3.5.1 Implementation

All steps to conduct an experiment with either method have been realized with a single software framework. Figure 18 shows a diagram of the program architecture. The software is a Python script that automatically executes all steps from the raw data to storing the result.
The most important software packages that were used in the implementation can be looked up in Table 5.

First, the script parses the monthly Excel sheets and saves the relevant backlog, sales, and scheduled information in a Pandas dataframe. Furthermore, Numpy arrays have been widely used. Both data structures are high-performance data analysis tools for scientific computing in Python and interact optimally with all other packages. Second, all missing values are filled according to the pattern described in Section 3.2.2. Third, a parameter selects the desired method to train a forecasting model. As already explained in this chapter, each of the algorithms requires specific further data preparation steps. Depending on the parameter settings, the data gets augmented (see 3.3.1) for the neural net and the random forest regression. In case of modeling a feedforward net the data gets additionally normalized and rescaled after training (see 3.3.1). Next, the data gets split for training and testing and
then the respective input sequences get automatically created (FNN and random forest: 3.3.2, 3.3.1, 3.3.2; ETS and ARIMA: 3.4.1, 3.4.1) and stored in a Numpy data structure.

Modeling the neural net was implemented in TensorFlow. This Python library is an open source software package provided by Google for high performance computation with CPUs and GPUs. The framework is widely used to train neural networks due to its efficiency and the large variety of options to make low-level adjustments for individual purposes.

The random forest implementation uses the scikit-learn package. This library offers a wide range of simple machine learning tools. Our Python script calls the RandomForestRegressor for training.

By contrast, the statistical algorithms require software packages only available for R. The Python package rpy2 is an interface between both languages. Thus, R code and all of its functionality may be included directly into Python. All statistical forecast methods rely on Robert Hyndman’s forecast package. Extensive information on this library is provided in Hyndman and Khandakar (2008) and Hyndman et al. (2017). We estimate ARIMA and ARIMAX model parameter with the auto.arima() function and use the default hyperparameter space of possible models \((p(\text{max})=5, q(\text{max})=5, P(\text{max})=2, Q(\text{max})=2)\) and set seasonal differencing \(D=1\) (see Section 2.2.3 and 3.4.3). ETS uses the ets() method. Both methods estimate parameters for all possible model combinations and return the one with lowest AICc value. Also, the package’s stl() method gets called to obtain the seasonal information for data augmentation for the machine learning algorithms (see 3.3.1).

As indicated with the red box, the script verifies the performance of each algorithm with the corresponding cross-validation by fitting multiple models. Finally all results and plots get immediately evaluated and stored in a file system. All graphs are plotted with either matplotlib (Python) or ggplot2 (R).

The orange circle in Figure 18 points at the general parameter settings of the Python script. These parameters are controlled with the sacred package. Using this library allows calling the script with many different parameter settings and also keeping conveniently track of these experiments by saving the configurations.

Moreover, we implemented a tool based on the results of the thesis that can be used by the supply chain management to make forecasts each month after adding new data. The tool loads stored models of the neural net and the random forest that have already been trained to make a forecast based on the newest data available. We stored the best models resulting from the evaluation in the upcoming chapter. Also, the software estimates new ETS, ARIMA, and ARIMAX models to make a prediction with the newly added data. Furthermore, the tool returns the mean of all estimators as an ensemble forecast. All forecasts are displayed on the console and saved to a file.
3.5. TECHNICAL SETUP

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
<th>Last access date of all websites: May 12, 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>3.6.1</td>
<td><a href="https://www.python.org/downloads/">https://www.python.org/downloads/</a></td>
</tr>
<tr>
<td>Pandas</td>
<td>0.20.1</td>
<td><a href="https://pandas.pydata.org/">https://pandas.pydata.org/</a></td>
</tr>
<tr>
<td>Numpy</td>
<td>1.12.1</td>
<td><a href="http://www.numpy.org/">http://www.numpy.org/</a></td>
</tr>
<tr>
<td>TensorFlow</td>
<td>1.1</td>
<td><a href="https://www.tensorflow.org/versions/r1.1">https://www.tensorflow.org/versions/r1.1</a></td>
</tr>
<tr>
<td>scikit-learn</td>
<td>0.18.1</td>
<td><a href="http://scikit-learn.org/dev/">http://scikit-learn.org/dev/</a></td>
</tr>
<tr>
<td>R</td>
<td>3.4.2</td>
<td><a href="https://cran.r-project.org/bin/windows/base/old/3.4.2/">https://cran.r-project.org/bin/windows/base/old/3.4.2/</a></td>
</tr>
<tr>
<td>rpy2</td>
<td>2.8.6</td>
<td><a href="https://rpy2.readthedocs.io/en/version_2.8.x/">https://rpy2.readthedocs.io/en/version_2.8.x/</a></td>
</tr>
<tr>
<td>forecast</td>
<td>8.2</td>
<td><a href="https://github.com/robjhyndman/forecast">https://github.com/robjhyndman/forecast</a></td>
</tr>
<tr>
<td>matplotlib</td>
<td>2.2.2</td>
<td><a href="https://matplotlib.org/">https://matplotlib.org/</a></td>
</tr>
<tr>
<td>ggplot2</td>
<td>2.2.0</td>
<td><a href="http://ggplot2.tidyverse.org/reference/">http://ggplot2.tidyverse.org/reference/</a></td>
</tr>
<tr>
<td>sacred</td>
<td>0.7.3</td>
<td><a href="http://sacred.readthedocs.io/">http://sacred.readthedocs.io/</a></td>
</tr>
</tbody>
</table>

Table 5: Software information.
Chapter 4

Evaluation

This chapter provides a thorough evaluation and analysis of the results derived from the experiments conducted according to the methods in the previous chapter. First, we will give and analyze the results for the neural networks. Second, we evaluate the outcome of the random forest models. Then, we provide the results for ETS, ARIMA, and ARIMAX. In the final section we discuss and compare the different methods. Comparing the output of each modeling technique and the desired forecast horizon (see Figure 9) reveals a mismatch. All models return the sales forecast for the next two months \( \hat{s}_{t+1} \) and \( \hat{s}_{t+2} \) at time \( t \). However, during evaluation we focus on the second month into the future and leave \( \hat{s}_{t+1} \) aside. Thus, we assess all experiments according only to the relative error of \( \hat{s}_{t+2} \). All tables, figures, and numbers refer only to the relative error of \( \hat{s}_{t+2} \). The box-whiskers plots used throughout the evaluation, as for instance in Figure 20, provide information about the distribution of the errors. The lower boundary of the box indicates the first quartile, the band inside the box is the median (second quartile), and the upper boundary is giving the third quartile. The range of the whiskers goes from the minimum to the maximum value. Also, all errors are mean absolute percentage errors (MAPE). In our result sections we also provide the standard deviation (SD).

4.1 Neural Net

As explained in Section 3.3.3, the neural nets were trained to minimize a cost function which essentially measures the quadratic distance between the net’s output \( \hat{y} = (\hat{s}_{t+1}, \hat{s}_{t+2}) \) and the actual future values \( y = (s_{t+1}, s_{t+2}) \). Yet, for our purposes we rank the trained models according to the final relative error (MAPE) between \( \hat{s}_{t+2} \) and \( s_{t+2} \) after training. All errors given (training, validation, test) are average figures resulting from a 5 fold cross-validation. The grid search for the best hyperparameter settings contained 2592 experiments for both \textit{Total} and \textit{Single Device} scenario. However, after running all experiments we filter the results. Basically, we conducted 864 experiments with three different learning rates \((864 \cdot 3 = 2592)\) and we want to keep the ones with the most appropriate learning rate. The lowest training error indicates the best convergence properties of the learning rate. Hence, we compare the training error for each experiment for the three learning rates and keep the experiment with the lowest training error.
4.1.1 Total Analysis

In this section we analyze the predictive performance of the feedforward neural nets which we trained regarding the total sales number of X-ray systems. Figure 19 shows the results of all 864 experiments sorted by the validation error in descending order. The left diagram reveals the influence of different hyperparameter settings on the final training set and validation set errors. Mostly towards the left side, the black and blue graph contain experiments with settings for networks with an insufficient low capacity. Here, the training error remains high and does not converge to a satisfactory minimum while having a high validation loss at the same time. Also, many experiments show clear signs of overfitting to the training data set with the final training loss going down towards zero combined with fairly high validation losses. Furthermore, the lowest validation errors appear towards the right with a steadily decreasing gap between training and validation loss. The influences of certain hyperparameters causing this outcome will be analyzed in this section. The right diagram compares the losses on the trained networks for validation and test set. Ideally, averaging the differences between validation and test error of all experiments of our grid should give zero. That is because both errors measure out-of-sample performance of the model and should therefore have similar numbers on average. Yet, the diagram reveals a slight bias towards lower test errors. This can be attributed to the very small test and validation sets (see Section 3.3.2) which might contain different levels of noise. Moreover, these graphs show that the lowest validation errors underestimate the true generalization error. Hence, after selecting the best model according to the lowest validation error, we have to take the test error as the final measurement to rate the model’s performance.

Different lengths of the moving window (see 3.3.1) to create samples from the data set gave different outcomes. As Figure 20 reveals, longer sequences containing sales $s$ and backlog $b$ data going further into the past resulted in a generally better performance regarding the validation error. The best model trained with short sequences had only a MAPE of 12.78 %, whereas longer sequences lowered this optimum to 10.82 %. Consequently, training the network with longer cohesive parts of the time series allowed better capturing of the underlying pattern of the process.

We applied data augmentation and dropout to increase the generalization performance of our trained networks. Figure 21 shows the effect of both methods on all experiments. The
left subfigure reveals a significant increase of the minimum training error if the data set was augmented from 0.76% (none) to 5.51% (middle) and 5.33% (high). In reverse, increasing augmentation lowered the minimum validation error from 11.69% (none) to 11.20% (middle) and 10.82% (high). Figure 22 provides insight on the effect of augmentation for the different kind of used network topologies. Interestingly, networks trained with augmented data deliver the best validation result for all complexities and reduce overfitting decisively.

Dropout, however, as can be seen in the right plot of Figure 21 did not cause lower errors on the validation data. In fact, the lowest validation error went up from 10.82% (0.0) to 11.14% (0.2) and 11.69% (0.4). The regularization method reduced overfitting by lifting the minimum training error from 0.76% (0.0) to 5.28% (0.2) and 6.64% (0.4). Analyzing dropout in more detail in Figure 23 shows that dropping a certain percentage of neurons during training did not improve the minimum validation error in any scenario. The networks in this figure with more than one hidden layer ([20, 10], [20, 10, 5], and [20, 15, 10, 5]) show the opposite. Validation and training error go up simultaneously. Concerning the single-hidden layer architecture the validation error variance gets lower with higher dropout and not as many experiments overfit. Yet, the lowest validation error is still given without dropout with 10.82%.
4.1. NEURAL NET

Figure 22: Augmentation effect filtered by topologies (NN Total).
4.1. NEURAL NET

Figure 23: Dropout effect filtered by topologies (NN Total).
Picking 9 experiments with fixed hyperparameter settings for the most complex network with four hidden layers ([20,15,10,5]) demonstrates the interaction between augmentation and dropout more vividly. All learning rates are 0.001, the exponential decay of the learning rate is 0.86, the length of the moving window is 14, L2 loss is 0, and we train without early stopping. The heatmaps of Figure 24 show the training and validation results for this hyperparameter setting when changing only augmentation and dropout rate. No augmentation and dropout result in heavy overfitting to the training data. Applying data augmentation or dropout decreases the gap between training and validation error. Looking at the validation errors reveals that high augmentation without dropout delivers the best result. However, the effect of augmentation may also be seen with a dropout rate of 20% where the error drops from over 21% to under 18% with augmentation.

Ultimately, Figure 25 shows the impact of early stopping and L2 regularization. Obviously, early stopping prevents overfitting to the training data as the threshold \( \pi \) which stops the training is linked to the training error (see 3.3.3). Also, the minimum validation error improves from 11.32% to 10.82% after applying early stopping. Yet, as the boxplots show, the overall improvement of the validation errors is only marginally. The effect of early stopping depending on the complexity of the network can be found in the appendix (see Figure 51). The right subplot analyzes the effect of using L2 regularization during training. The plots indicate that the parameter values were not chosen to be large enough for having a real impact. The training error should go up when L2 is applied.

Results
Table 6 gives the best models based on the validation error for each network topology to predict future sales figures for the total number of systems. The validation errors and their respective standard deviation of all four hyperparameter settings are in a close range. The best model’s results with a MAPE validation error of 10.82% are highlighted. The diagram on the left in Figure 26 shows the history of the validation and training loss during training for our best model. The training process gets stopped fairly early after only
4.1. NEURAL NET

Figure 25: Effect of early stopping and L2 regularization (NN Total).

<table>
<thead>
<tr>
<th></th>
<th>[20]</th>
<th>[20,10]</th>
<th>[20,10,5]</th>
<th>[20,15,10,5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Augmentation</td>
<td>High</td>
<td>High</td>
<td>Middle</td>
<td>Middle</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>L2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.001</td>
<td>0.0005</td>
</tr>
<tr>
<td>Early stopping</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>Sequence length</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Learning rate (LR)</td>
<td>0.001</td>
<td>0.0005</td>
<td>0.0005</td>
<td>0.001</td>
</tr>
<tr>
<td>Exponential decay LR</td>
<td>0.86</td>
<td>0.86</td>
<td>0.86</td>
<td>1.0</td>
</tr>
<tr>
<td>Training MAPE</td>
<td>9.20%</td>
<td>9.54%</td>
<td>9.43%</td>
<td>9.36%</td>
</tr>
<tr>
<td>Training SD</td>
<td>7.67%</td>
<td>7.58%</td>
<td>8.17%</td>
<td>7.84%</td>
</tr>
<tr>
<td>Validation MAPE</td>
<td>10.82%</td>
<td>11.08%</td>
<td>11.36%</td>
<td>11.64%</td>
</tr>
<tr>
<td>Validation SD</td>
<td>8.83%</td>
<td>8.79%</td>
<td>9.00%</td>
<td>8.25%</td>
</tr>
<tr>
<td>Test MAPE</td>
<td>13.70%</td>
<td>13.34%</td>
<td>14.01%</td>
<td>13.42%</td>
</tr>
<tr>
<td>Test SD</td>
<td>9.36%</td>
<td>9.26%</td>
<td>9.63%</td>
<td>9.51%</td>
</tr>
</tbody>
</table>

Table 6: Cross-validation result (NN Total).

200 iterations. Considering the size of the training set (with high augmentation) of approximately 5000 samples (depending on the fold of the cross-validation) and a batch size of 50 for each iteration the training stops already after a little more than 200 iterations (or after approximately 2 epochs). Yet, the right plot indicates that longer training would have resulted in overfitting and a poorer generalization performance. The plot shows training a model with the same hyperparameter settings except that early stopping was turned off. The remaining respective four plots of the cross-validation which belong to either setting can be found in the appendix (see figures 45 and 46).
4.1. NEURAL NET

4.1.2 Single Device Analysis

This section analyzes the forecast performance of the trained neural nets to predict the sales numbers for each system type (MP, Floor, Ceiling, Zeego, and Biplane) individually. We trained one model with all the data of all individual system types in each experiment (see section 3.3.2). 14 training runs diverged which resulted in very high losses. These experiments were dropped for the evaluation in order to avoid a distortion of the plots. Thus, Figure 27 provides the results of 850 experiments sorted by the validation error in descending order. The diagram on the left compares the final training and validation error for all experiments.

The relative training loss indicated by the blue line oscillates approximately between 50% and 150% and remains almost always above the validation loss for the first 400 experiments. This phenomenon will be evaluated when we analyze the influence of single hyperparameters. The subsequent about 50 experiments have the lowest training errors with a minimum of 13.06%. Regarding the rest of the experiments, the training and validation errors decrease simultaneously with training errors being slightly higher on average. The fact that no trained model among all experiments managed to overfit heavily with a training error close to zero.

Figure 26: Training with and without early stopping (NN Total).

Figure 27: Experiments (NN Single Device).
4.1. NEURAL NET

by basically memorizing the training data allows two implications. First, we did not train for
enough iterations, or second, the complexity of our network was not sufficient with a maximum
of 4 hidden layers ([20, 15, 10, 5]). The right diagram compares the experiments’ validation
and test set losses. Unlike the Total experiments the diagram reveals a bias towards lower
validation errors. This may again be attributed to the very small test and validation sets. Yet
again, these graphs show that the lowest validation errors underestimate the generalization
error.

![Figure 28: Effect of sequence length on validation error (NN Single Device).](image)

Changing the length of the moving window (see 3.3.1) for sample generation provides
different outcomes. Figure 28 compares training with shorter and longer sequences of sales $s$
and backlog $b$ data. The best model trained with short sequences had a MAPE of 23.70 %,
whereas longer sequences reduced the minimum to 23.20 %. The mean validation error for a
window length of 4 is 31.52% compared to 30.08% for the longer sampling size. Hence, longer
sequences deliver slightly better results regarding the validation error. The long whiskers for
a window length of 4 indicate outliers.

Once again, we applied data augmentation and dropout intending to increase the out-of-
sample performance of our trained networks. Figure 29 shows the effect of both methods on
all experiments. Its right subfigure shows the effect of applying dropout at different rates

![Figure 29: Effect of augmentation and dropout (NN Single Device).](image)
during training. The best validation error for each setting remains in close range with the others from 23.70% (0.0) to 23.20% (0.2) and 25.51% (0.4). Also, the regularization method increases the minimum training error from 13.06% (0.0) to 18.92% (0.2) and 22.08% (0.4). Moreover, the training and validation error variance increases with higher dropout. Analyzing the effect on dropout depending on the number of hidden layers reveals similar results as with the Total experiments. The minimum validation errors increase with higher dropout with the exception of the single-hidden layer networks. These plots can be found in the appendix (see Figure 50).

The right plot of Figure 29 reveals an significant effect on the training error if the data set was augmented. The relative training loss rises from 13.06% (none) to 26.72% (middle) and 20.77% (high). Also the mean of the errors increases decisively from 27.10% (none) to 67.27% (middle) and 48.30% (high). By contrast, the validation error’s key figures remain almost exactly the same for either setting (none, middle, high) with a minimum of about 23%, a maximum of 89%, and a mean of 31%. Filtering all results and plotting only the

![Figure 30: Experiments without augmentation (NN Single Device).](image)

experiments without augmentation reveals some insight about these unexpected numbers. The left plot in Figure 30 compares the training and test loss after removing experiments involving augmentation. Suddenly, the graph of training losses drops below validation. The left side of the graph shows high training errors in the same range with their corresponding validation errors. Dropout causes these models to have low capacity. This may be seen in the right plot after removing all experiments with dropout. Also, Figure 31 plotting only experiments involving augmentation shows that their validation loss is almost always a lower boundary of the training loss and often significantly lower. Altogether, data augmentation did not have the same effect for the Single Device experiments compared to their successful implementation in the Total experiments. This may have two possible reasons. First, the data augmentation scheme (see 3.3.1) does not suit the Single Device data. Second, the amount of data requires much longer training time or more complex networks. Compared to the original training set size of around 500 (depending on the cross-validation partition) an augmented data set contained 13000 (middle) respectively 28000 (high) samples. This huge difference might require more training steps than 20000 (about 77 (middle) and 36 (high) epochs).

Figure 32 analyzes the effects of early stopping and L2 regularization. Again, early stopping rises the minimum relative error on the training data as we set the threshold $\pi$ (see 3.3.3) to stop training. However, the mean of the training (47%) and validation errors (30%) remain
4.1. NEURAL NET

almost exactly the same for both options. Moreover, the minimum of 23% and the maximum of 89% of the validation error stay the same. This is due to the fact that the threshold to stop training is almost never reached. Hence, early stopping does not provide any improvement regarding the generalization error. The right figure shows the effect of using L2 regularization during training. Similarly to the Total experiments the plots reveal that the parameter values were not large enough to make a difference. The training error should go up when L2 is applied. However, the mean of the training error remains the same at about 46% regardless of the L2 parameter value.

Results

Table 7 shows the best models based on the validation error to forecast future sales numbers for individual systems. The validation errors and their respective standard deviation for each network topology are again in a very close range. The best model’s results with a MAPE validation error of 23.20% are highlighted. The test performance of this model maybe distinguished into the predictive performance for each individual system type. These results are given in table 8. The forecast error and the standard deviation of the errors differs

![Figure 31: Experiments with augmentation (NN Single Device).](image)

![Figure 32: Effect of early stopping and L2 regularization (NN Single Device).](image)
significantly between the different systems. The systems Zeego, Biplane, and Floor show similar results whereas the forecast for Ceiling is much better and for MP much much worse.

<table>
<thead>
<tr>
<th>Augmentation</th>
<th>[20]</th>
<th>[20,10]</th>
<th>[20,10,5]</th>
<th>[20,15,10,5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dropout</td>
<td>None</td>
<td>Middle</td>
<td>Middle</td>
<td>Middle</td>
</tr>
<tr>
<td>L2</td>
<td>0.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Early stopping</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Sequence length</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>Learning rate (LR)</td>
<td>0.001</td>
<td>0.0001</td>
<td>0.0005</td>
<td>0.0005</td>
</tr>
<tr>
<td>Exponential decay LR</td>
<td>1.0</td>
<td>1.0</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td>Training MAPE</td>
<td><strong>19.48%</strong></td>
<td>32.83%</td>
<td>31.03%</td>
<td>37.75%</td>
</tr>
<tr>
<td>Training SD</td>
<td>25.57%</td>
<td>177.16%</td>
<td>180.43%</td>
<td>208.71%</td>
</tr>
<tr>
<td>Validation MAPE</td>
<td><strong>23.20%</strong></td>
<td>23.98%</td>
<td>23.80%</td>
<td>23.70%</td>
</tr>
<tr>
<td>Validation SD</td>
<td>26.01%</td>
<td>26.97%</td>
<td>26.89%</td>
<td>25.03%</td>
</tr>
<tr>
<td>Test MAPE</td>
<td><strong>27.79%</strong></td>
<td>26.57%</td>
<td>26.89%</td>
<td>28.34%</td>
</tr>
<tr>
<td>Test SD</td>
<td><strong>37.53%</strong></td>
<td>32.70%</td>
<td>35.56%</td>
<td>41.51%</td>
</tr>
</tbody>
</table>

Table 7: Cross-validation result (NN Single Device).

<table>
<thead>
<tr>
<th>MP</th>
<th>Zeego</th>
<th>Biplane</th>
<th>Ceiling</th>
<th>Floor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test MAPE</td>
<td>41.90%</td>
<td>29.91%</td>
<td>25.73%</td>
<td>14.15%</td>
</tr>
<tr>
<td>Test SD</td>
<td>106.16%</td>
<td>20.95%</td>
<td>22.85%</td>
<td>16.60%</td>
</tr>
</tbody>
</table>

Table 8: Test result per system type (NN Single Device).

Finally, the plot on the left in Figure 33 shows the evolution of the validation and training loss during training for our best model which has only one hidden layer. The training process does not get interrupted by early stopping although this hyperparameter was set to True and runs for 20000 iterations. Yet, the training loss starts improving only very slowly after approximately 7500 iterations. This may indicate that this single-hidden layer network has not enough capacity. The plot on the right shows the training and validation loss history of the best network with four hidden-layers. Training stops improving again after approximately 7500 iterations similar to the single-hidden layer architecture. The respective four plots of the cross-validation which belong to either topology may again be found in the appendix (see figures 47 and 48). This result implies that longer training may not necessarily deliver better models. Problems with the data, normalization, or the learning rate could have influenced the performance. Also, as suggested before, more complex models may have given better results.
4.2 Random Forest

This section evaluates the random forest models (RF) which were trained with the CART algorithm as explained in Section 3.3.4. Again, we rank the trained models according to the final relative error (MAPE) between $\hat{s}_{t+2}$ and $s_{t+2}$ and ignore the first prediction $\hat{s}_{t+1}$. The errors given (training, validation, test) are average numbers resulting from a 5 fold cross-validation. The grid search for the best hyperparameter settings contained 216 experiments for either Total and Single Device data.

4.2.1 Total

Analysis

In this section we analyze the forecast performance of the random forests we modeled regarding the total sales figures of X-ray systems. Figure 34 provides the results of all 216 experiments sorted by the validation error in descending order. The left plot shows the impact of different hyperparameter settings on the final training set and validation set error. The blue and black graph tell that training error and validation errors have a tendency to go down in parallel. Yet, the training error fluctuates depending on the hyperparameter setting. Generally, the
significant gap between training and validation loss is a sign that the random forests overfit to the training data. We will analyze in this section how different hyperparameter settings affect the outcome of the experiments. Again, the right diagram compares the validation and test losses of our models. Test and validation loss are not evenly balanced as test errors tend to be lower on average. This observation supports our assumption from the neural networks that the test data might not be as noisy as the validation data (see 4.1.1). Equally to the experiments with the neural networks, the graphs reveal that the lowest validation errors underestimate the true generalization error. Consequently, we have to take the test error as the final measurement to rate a model’s performance again when we choose the one with the lowest validation error.

Once again, changing the length of the window (see 3.3.1) to create sequences made a huge difference. The left boxplot in Figure 35 reveals that longer sequences containing more past sales $s$ and backlog $b$ data provided much better results regarding the validation error. The best random forest modeled with short sequences had a MAPE of 14.72%. Longer sequences reduced the minimum validation error to 12.28%. Thus, modeling the decision trees with longer parts of the time series delivers more accurate forecasts.

![Figure 35: Effect of sequence length and augmentation (RF Total).](image)

Applying data augmentation yielded similar improvements of the validation loss as for the neural networks. The right diagram in Figure 21 shows the effect of this method on all experiments. Increasing augmentation reduced the minimum validation error from 15.59% (none) to 12.28% (middle) and 12.65% (high). Surprisingly, the training error gets also lower for the augmented data sets from 4.87% (none) to 2.28% (middle) and 1.28% (high). This phenomenon is caused by the CART algorithm (see 2.1.2). The algorithm builds decision trees according to the minimum leaf size. Allowing a minimum of one sample per leaf would imply a perfect score of zero on the training set. However, the CART algorithm prunes the tree by collapsing nodes to reduce complexity. Trees modeled with more data will also get pruned. However, if the additional data is similar (augmentation is only a distortion of the original data) samples falling into one region (leaf) will be closer together in terms of range and therefore closer to the average which will be the output. Hence, having more data in this case results in lower training errors.

Choosing a different number of decision trees for the random forest models did not have any effect on the validation loss as may be seen in Figure 36. For either 10, 50, or 100 trees the validation loss mean remained the same with about 12%. The minimum 16% and maximum
validation error 24% did not change as well. Minimum and maximum training error and its mean did not change significantly either.

The right plot of Figure 36 shows the impact of the decision trees’ leaf size. Setting different thresholds for the minimum number of samples that had to fall into one leaf had a considerable impact on the models. As presumed the training error increases if the minimum leaf size gets bigger. Bigger leaves simply imply less precise regression. The minimum training error goes up from 1.28% (1) to 2.45% (5), to 3.74% (50) and 5.60% (100). Thus, the models do not overfit too much anymore on the training data. Concerning the validation error the minimum stagnates between 12% and 13%. Yet, the variance of the errors goes up resulting in a larger mean and maximum of the validation errors. Another method to prevent overfitting that may have been applied is to limit the maximum depth of the decision trees.

Choosing different percentages for the number of features considered at each split when the trees are constructed by the algorithm did not improve the generalization ability of the models either. Figure 37 shows a small average improvement of the training error from from 9.13% (1) to 7.53% (0.5) and 7.92% (0.3). However, the figures of the validation loss remained the same with about 16% for the mean, 13% for the minimum and 24% for the maximum.
4.2. RANDOM FOREST

Results

Table 9 provides the best models to predict future sales numbers for the total number of systems for all different forest sizes based on the validation error. All validation errors and their respective standard deviation are in a very close range. The best model’s results with a MAPE validation error of 12.28% are highlighted.

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Augmentation</td>
<td>High</td>
<td>Middle</td>
<td>Middle</td>
</tr>
<tr>
<td>Leaf size</td>
<td>20</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Split feature percentage</td>
<td>0.5</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td>Sequence length</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Training MAPE</td>
<td>6.13%</td>
<td>4.40%</td>
<td>2.37%</td>
</tr>
<tr>
<td>Training SD</td>
<td>5.73%</td>
<td>4.54%</td>
<td>2.49%</td>
</tr>
<tr>
<td>Validation MAPE</td>
<td>12.65%</td>
<td>12.28%</td>
<td>12.39%</td>
</tr>
<tr>
<td>Validation SD</td>
<td>10.83%</td>
<td>10.53%</td>
<td>10.34%</td>
</tr>
<tr>
<td>Test MAPE</td>
<td>14.81%</td>
<td>15.13%</td>
<td>14.42%</td>
</tr>
<tr>
<td>Test SD</td>
<td>11.99%</td>
<td>12.14%</td>
<td>11.68%</td>
</tr>
</tbody>
</table>

Table 9: Cross-validation result (RF Total).

4.2.2 Single Device

Analysis

This section analyzes the predictive performance of the random forest models to forecast the sales numbers for each system type (MP, Floor, Ceiling, Zeego, and Biplane) individually. We modeled one random forest with the data of all systems in each experiment. Figure 38 shows
4.2. RANDOM FOREST

the results of all 216 experiments sorted by the validation error in descending order. The left subfigure reveals the impact of different hyperparameter settings on the final training set and validation set error. The training and test loss curves show again that the best validation results might be achieved if the training loss is low as well. As previously with the Total experiments most models seem to overfit to the training data, in particular the ones giving the lowest validation errors. The right subfigure which compares the test and corresponding validation loss yields the same phenomenon as observed with the neural networks. Validation errors are significantly lower than test errors in almost all experiments. This bias may again be explained with the characteristics of time series data and the small size of the data set.

Varying the length of the moving window that generates the samples confirms our previous results. The left boxplot in Figure 39 shows that longer sequences with more sales and backlog data from the past provided marginally better results regarding the validation error. The mean validation error decreases only from 31.73% to 30.78%. The best model based on the short sequence length of 4 had a MAPE of 27.47%. Modeling with longer sequences lowered this minimum to 27.22%.

![Figure 39: Effect of sequence length and augmentation (RF Single Device).](image)

Unlike the neural net, random forests modeled with augmented data sets provided the best validation error. The right diagram in plot 39 shows that increasing augmentation reduced the minimum validation error from 29.49% (none) to 27.24% (middle) and 27.22% (high). Additionally, the mean goes down from 34.29% (none) to 30.04% (middle) and 29.43% (high). As with the Total random forest experiments, the minimum training error and its mean get considerably lower for the augmented data sets for the same reason as previously explained.

Varying the number of trees did not have a significant impact similar to the Total experiments. Figure 40 shows that validation loss mean (31%), minimum (28%), and maximum (50%) stayed about the same for either 10, 50, or 100 trees.

The right plot of Figure 40 shows that the impact of the decision trees’ leaf size did not change to the previous experiments. Increasing the thresholds for the minimum number of samples that need to fall into one region increases the training error. That error goes up from 2.47% (1) to 5.37% (5), to 8.11% (50) and 11.66% (100). The lowest validation error, however, stays at about 27%.

Figure 41 shows a significant improvement of the validation error regarding the percentage of features selected for each split. The minimum validation error goes down from from 30.43% (1) to 27.22% (0.5) and 27.30% (0.3). Similarly, the figures of the training loss decrease on average as may easily be seen in the plot.

Figure 40: Effect of number of trees (RF Single Device).

Figure 41: Effect of feature selection ratio (RF Single Device).
4.2. RANDOM FOREST

Results

Table 10 shows the best random forest models based on the validation error for the three different forest sizes. All validation errors and their respective standard deviation are once again in a very close range. The best model’s results with a MAPE validation error of 27.22% are highlighted. The test performance distinguished by the individual systems can be found in table 11. The prediction error and the standard deviation differs again considerably. Moreover, the results match with neural networks. As for the neural net, the systems Zeego, Biplane, and Floor show results in similar range. Also, the forecast for Ceiling is by far the most reliable. The prediction accuracy for MP systems is again far behind the others.
4.3 ETS

Table 10: Cross-validation result (RF Single Device).

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Augmentation</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Leaf size</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Split feature percentage</td>
<td>0.3</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Sequence length</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Training MAPE</td>
<td>9.94%</td>
<td>5.48%</td>
<td><strong>5.37%</strong></td>
</tr>
<tr>
<td>Training SD</td>
<td>6.15%</td>
<td>3.89%</td>
<td><strong>3.97%</strong></td>
</tr>
<tr>
<td>Validation MAPE</td>
<td>27.60%</td>
<td>27.32%</td>
<td><strong>27.22%</strong></td>
</tr>
<tr>
<td>Validation SD</td>
<td>34.84%</td>
<td>32.09%</td>
<td><strong>32.00%</strong></td>
</tr>
<tr>
<td>Test MAPE</td>
<td>36.50%</td>
<td>33.52%</td>
<td><strong>33.37%</strong></td>
</tr>
<tr>
<td>Test SD</td>
<td>60.57%</td>
<td>56.17%</td>
<td><strong>55.63%</strong></td>
</tr>
</tbody>
</table>

The ETS models which, based on Akaike’s information criterion, were selected for prediction varied, even within one time series of sales data, e.g. for Total. We evaluate ETS with a rolling forecast starting with 121 observations (see 2.2.2). With more observed values becoming available for each experiment other models might become more appropriate according to the AICc. Regarding the experiments to predict the Total number of future sales 57 \((M,A,M)\) and 3 \((M,A_d,M)\) were chosen. These models differ only in one component as three models expect a ceasing trend. Similarly, individual system types were predicted with varying models.
as the input length of time series increased. However, model choices were not so homogeneous for the single systems MP, Floor, Ceiling, Zeego, and Biplane. The number of models varied between three and four for all types. Yet, as for instance the forecast for Floor systems shows, the models vary significantly with 25 (A.N.A), 11 (M.A.A), 3 (M.A.M), and 21 (M.N.A) models. The divergence between the model types was similar for all other types. These univariate forecasts base exclusively on the sales data. The Floor example shows that additional observations change how the ETS algorithm assesses the characteristics of the time series. However, these changes of model choice are not in a strict chronological order. This implies that it is ambiguous for the ETS algorithm if the observations show additive or multiplicative errors, no trend or additive trend, additive or multiplicative seasonal properties. Different models give AICc values in a very close range without one model being clearly superior and cause this ambiguity of models. The boxplots in Figure 42 show the distribution of the relative errors for the different forecast scenarios.

Figure 42: ETS error distribution over all experiments.

4.4 ARIMA

Table 13 provides the results of ARIMA time series modeling. The results rely on a rolling forecast with each 60 experiments. Forecasting the sum of all systems (Total) delivered the best results with a MAPE of 13.81%. Forecasting single systems was again much less solid with relative errors between 25.12% (Ceiling) and 64.10% (MP).

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>MP</th>
<th>Zeego</th>
<th>Biplane</th>
<th>Ceiling</th>
<th>Floor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPE</td>
<td>13.81%</td>
<td>64.10%</td>
<td>44.84%</td>
<td>30.05%</td>
<td>25.12%</td>
<td>30.14%</td>
</tr>
<tr>
<td>SD</td>
<td>11.51%</td>
<td>133.24%</td>
<td>56.38%</td>
<td>30.08%</td>
<td>24.63%</td>
<td>31.51%</td>
</tr>
</tbody>
</table>

Table 13: Result rolling forecast ARIMA.

Similar to ETS, different ARIMA models were picked as the length of the time series increases during the rolling forecast. Once again, the Total experiments delivered the most uniform results in terms of model variability with 19 (0,0,0)(0,1,1)[12], 25 (0,1,1)(0,1,1)[12], and 17 (1,0,0)(0,1,1)[12] models. All models contain one seasonal moving average term at lag 12.
4.5. ARIMAX

Selecting models for the single systems MP, Floor, Ceiling, Zeego, and Biplane was not that stable. The number of models varied between five and fifteen for all types. For instance, the forecast of Zeego systems contains 6 (0,1,1)(2,1,1)[12], 1 (1,0,2)(0,1,1)[12], 24 (1,0,3)(0,1,1)[12], 12 (1,1,2)(0,1,1)[12], 7 (2,0,2)(0,1,1)[12], and 10 (2,1,2)(0,1,1)[12] models. Again, these changes of model selection were not in a strict chronological order implying ambiguity about the properties of the respective time series. Figure 43 shows the distribution of the relative errors for all forecast scenarios.

![Figure 43: ARIMA error distribution over all experiments.](image)

### 4.5 ARIMAX

The results for time series modeling with ARIMAX are shown in table 14. As before, the results base on a rolling forecast with 60 experiments. The most accurate ARIMAX forecast is also for all systems combined (Total) with a relative error of 9.55%. Modeling the time series for each individual system provided relative errors between 19.44% (Ceiling) and 42.72% (MP).

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>MP</th>
<th>Zeego</th>
<th>Biplane</th>
<th>Ceiling</th>
<th>Floor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPE</td>
<td>9.55%</td>
<td>42.72%</td>
<td>35.53%</td>
<td>24.00%</td>
<td>19.44%</td>
<td>25.16%</td>
</tr>
<tr>
<td>SD</td>
<td>8.28%</td>
<td>60.33%</td>
<td>38.44%</td>
<td>26.95%</td>
<td>20.16%</td>
<td>23.35%</td>
</tr>
</tbody>
</table>

Table 14: Result rolling forecast ARIMAX.

ARIMAX selected different terms during the rolling forecast in analogy to ETS and ARIMA. The multiple regression part of the models, as given in Section 3.4.1, stayed the same in each experiment. The Total experiments have 11 (0,0,0)(0,1,1)[12], 13 (0,0,0)(2,1,0)[12], 8 (1,0,2)(1,1,0)[12], 14 (2,0,0)(0,1,1)[12], and 10 (4,0,0)(0,1,1)[12] terms for the ARIMA part. The number of models for MP, Floor, Ceiling, Zeego, and Biplane varied between five and thirteen. Also here, similar models were not picked in consecutive order. Figure 44 shows the distribution of the relative errors for all forecast scenarios. This method may deliver better results if the backlog predictor variable was tuned for each system. However, no distinction was made and it was assumed that these systems remain in the backlog for the same amount of time as all systems together (Total) with 4.3 months (see 3.4.1).
4.6 Ensemble

The results of various studies as Krauss et al. (2017) and Khashei and Bijari (2011) suggest to combine different models to form an ensemble by averaging the forecasts (see Section 2.3). We averaged the results of our best neural network and random forest models with the forecasts of ETS, ARIMA, and ARIMAX. The test set of the random forest and the neural net contains only 23 samples. Hence, the results in table 15 show the ensemble’s relative error for 23 forecasts.

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>MP</th>
<th>Zeego</th>
<th>Biplane</th>
<th>Ceiling</th>
<th>Floor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPE</td>
<td>11.94%</td>
<td>56.48%</td>
<td>27.22%</td>
<td>29.04%</td>
<td>17.231%</td>
<td>31.36%</td>
</tr>
<tr>
<td>SD</td>
<td>7.66%</td>
<td>105.17%</td>
<td>30.63%</td>
<td>29.01%</td>
<td>18.06%</td>
<td>27.33%</td>
</tr>
</tbody>
</table>

Table 15: Result ensemble.

4.7 Overall Results

As an overview, we provide the results how well each method performed to predict all systems and each individual system type. Additionally, we provide a two-sided 95% prediction interval (PI) as a boundary to measure the confidence in our forecasts (see 2.2.1). These intervals assume that all forecast errors are normally distributed. These intervals indicate the range around an estimate \( \hat{s}_{t+2} \) in which the actual value \( s_{t+2} \) lies with a 95% probability. Moreover, we provide the forecasting result for a very naive approach (Naive). We say that sales \( s_{t+2} \) is the same as \( s_t \). In other words, we compare the forecast quality if we assume that the number of sales does not change. This method serves as a benchmark for the more sophisticated forecast approaches.
### 4.7. Overall Results

#### Total

<table>
<thead>
<tr>
<th></th>
<th>Neural net</th>
<th>Rand. forest</th>
<th>ETS</th>
<th>ARIMA</th>
<th>ARIMAX</th>
<th>Ensemble</th>
<th>Naive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAPE</strong></td>
<td>13.70%</td>
<td>15.13%</td>
<td>13.38%</td>
<td>13.81%</td>
<td>9.55%</td>
<td>11.94%</td>
<td>25.79%</td>
</tr>
<tr>
<td><strong>SD</strong></td>
<td>9.36%</td>
<td>12.14%</td>
<td>9.37%</td>
<td>11.51%</td>
<td>8.28%</td>
<td>7.66%</td>
<td>22.41%</td>
</tr>
<tr>
<td><strong>95 % PI</strong></td>
<td>±18.35%</td>
<td>±23.79%</td>
<td>±18.37%</td>
<td>±22.54%</td>
<td>±16.23%</td>
<td>±15.01%</td>
<td>±43.92%</td>
</tr>
</tbody>
</table>

Table 16: Final result (Total).

#### MP

<table>
<thead>
<tr>
<th></th>
<th>Neural net</th>
<th>Rand. forest</th>
<th>ETS</th>
<th>ARIMA</th>
<th>ARIMAX</th>
<th>Ensemble</th>
<th>Naive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAPE</strong></td>
<td>41.90%</td>
<td>48.07%</td>
<td>59.44%</td>
<td>64.10%</td>
<td>42.72%</td>
<td>56.48%</td>
<td>59.44%</td>
</tr>
<tr>
<td><strong>SD</strong></td>
<td>106.16%</td>
<td>79.53%</td>
<td>125.51%</td>
<td>133.24%</td>
<td>60.33%</td>
<td>105.17%</td>
<td>101.69%</td>
</tr>
<tr>
<td><strong>95 % PI</strong></td>
<td>±208.13%</td>
<td>±155.88%</td>
<td>±245.99%</td>
<td>±261.16%</td>
<td>±118.25%</td>
<td>±206.13%</td>
<td>±199.31%</td>
</tr>
</tbody>
</table>

Table 17: Final result (MP).

#### Zeego

<table>
<thead>
<tr>
<th></th>
<th>Neural net</th>
<th>Rand. forest</th>
<th>ETS</th>
<th>ARIMA</th>
<th>ARIMAX</th>
<th>Ensemble</th>
<th>Naive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAPE</strong></td>
<td>29.91%</td>
<td>31.65%</td>
<td>45.45%</td>
<td>44.84%</td>
<td>35.53%</td>
<td>27.22%</td>
<td>65.55%</td>
</tr>
<tr>
<td><strong>SD</strong></td>
<td>20.95%</td>
<td>42.02%</td>
<td>58.36%</td>
<td>56.38%</td>
<td>38.44%</td>
<td>30.63%</td>
<td>74.31%</td>
</tr>
<tr>
<td><strong>95 % PI</strong></td>
<td>±41.06%</td>
<td>±82.36%</td>
<td>±114.38%</td>
<td>±110.50%</td>
<td>±75.36%</td>
<td>±60.03%</td>
<td>±143.41%</td>
</tr>
</tbody>
</table>

Table 18: Final result (Zeego).

#### Biplane

<table>
<thead>
<tr>
<th></th>
<th>Neural net</th>
<th>Rand. forest</th>
<th>ETS</th>
<th>ARIMA</th>
<th>ARIMAX</th>
<th>Ensemble</th>
<th>Naive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAPE</strong></td>
<td>25.73%</td>
<td>28.89%</td>
<td>30.47%</td>
<td>30.05%</td>
<td>24.00%</td>
<td>29.04%</td>
<td>40.65%</td>
</tr>
<tr>
<td><strong>SD</strong></td>
<td>22.85%</td>
<td>34.83%</td>
<td>32.56%</td>
<td>30.08%</td>
<td>26.95%</td>
<td>29.01%</td>
<td>35.04%</td>
</tr>
<tr>
<td><strong>95 % PI</strong></td>
<td>±44.78%</td>
<td>±68.27%</td>
<td>±63.81%</td>
<td>±58.97%</td>
<td>±52.83%</td>
<td>±56.85%</td>
<td>±68.67%</td>
</tr>
</tbody>
</table>

Table 19: Final result (Biplane).
In order to verify the results of the neural nets, the best five models for both Total and Single Device experiments regarding the validation error were repeated each 5 times with the same settings but with a different seed for the pseudo random number generator. On average the result deviated 0.58% (Total) and 0.42% (Single Device). This is within a reasonable range and validates the results.

The various methods we applied for the machine learning algorithms differed considerably in their effect. Respective regularization for a better generalization ability did not improve the out-of-sample error of neither neural network nor random forest models with one exception. Our data augmentation scheme to extend the data set by simulating different trends and seasons lowered the validation error and made our models more robust. Yet, the forecast of Single Device figures with the neural net has to be excluded from this statement (see Figure 31). Training both algorithms with longer sequences and thereby with more observations of the past had a positive impact regarding the Total forecast. By contrast, the complexity of the neural net and the random forest did not influence the predictive error significantly. The best validation results for the neural network models with one, two, three, and four hidden layers are in absolute numbers within the range of 1% for Total and Single Device (see tables 6 and 7). The same applies for the random forest regression with 10, 50, and 100 decision trees (see tables 9 and 10).

The final results in Section 4.7 show very different forecast accuracies depending on the task. Predicting the sum of all system subtypes (Total) is much more reliable with a relative error of 9.55% compared to forecasting individual system types with a MAPE range from 14.15% (Ceiling) to 24.00% (Biplane), to 25.16% (Floor), to 29.91% (Zeego), and to 42.72%\(^1\)

\(^1\)ARIMAX was chosen over the neural net for the much smaller standard deviation of the errors.
Similarly, the standard deviation of the errors increases from 8.28% for Total to 16.60% (Ceiling), to 20.95% (Zeego), to 23.35% (Floor), to 26.95% (Biplane), and even to 60.33% (MP). Yet, analyzing the results the fact that

\[ MP_t + Zeego_t + Biplane_t + Ceiling_t + Floor_t = Total_t \]

has to be taken into consideration. Taking the sum of various variables instead of predicting them individually reduces the variance. Thus, the collective time series is more stable and therefore easier to forecast. Moreover, the mere quantity of the sales figures for each scenario has an influence on accuracy. In fact, there is a fairly high negative correlation of \(-0.72\) between the number of systems to predict and the relative error. In other words, smaller sales numbers imply a higher forecast error. Hence, predicting the Total number of X-ray machines is more accurate than predicting only MP systems as they differ hugely in quantity. However, this is also connected to our measurement criterion. Comparing the forecasts with a percentage error does not take the absolute error into account. On average each of the five system types makes 20% of all sold systems. Hence, the same absolute error in this example would cause the relative error to be five times higher for a single system type than for all combined.

Currently, the supply chain management makes a manual forecast (SCM forecast, for short) every month on the 15th for the upcoming month. Their average relative error over four years until September 2017 can be seen in table 22. The numbers for Single Device are averaged over all system types. The table shows that the human forecast is still more precise for both

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Single Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPE</td>
<td>7.5%</td>
<td>15.5%</td>
</tr>
<tr>
<td>SD</td>
<td>5.6%</td>
<td>5.9%</td>
</tr>
</tbody>
</table>

Table 22: Forecast Supply Chain Management.

scenarios compared to the applied methods in this thesis. The gap of the relative errors in the Total scenario is not so huge. Yet, the difference is significant, especially regarding the standard deviation. Looking at the forecast for individual system types reveals a much bigger discrepancy. Combining the relative error of our most accurate models provides an average relative error of 27.19% and a standard deviation of 29.64%. Both values are far behind the precision of the current approach. In particular the small deviation of the errors makes the forecast of the supply chain management much more reliable. However, the SCM forecast has a two weeks time advantage to make their prediction. Our algorithms make a sales forecast at time \( t \) for \( s_t + 2 \) (see Figure 9). The human forecast does not depend on the monthly census at the end of each month which allows them to make their prediction for \( s_t + 2 \) at time \( t + \frac{1}{2} \) with additional knowledge. That difference makes our ARIMAX forecasts for all systems to be manufactured (Total) very interesting to use in practice since the errors are within a comparable range and the numbers are two weeks earlier available. Our models to predict future sales figures for individual system types, however, provide no gain for the SCM in the daily business.

Therefore, the following assessment to rank the different forecasting techniques applied in this thesis bases exclusively on the Total results. The comparison with the Naive method justifies the utilization of each approach as all algorithms improve this benchmark result. The ensemble approach to average the forecast of all methods did not provide the best forecast. Yet,
the ensemble forecast has the lowest standard deviation. The random forest regression provides
the model with highest relative error. Univariate time series modeling with ETS and ARIMA
provides similar results to the neural network despite having only the sales data available.
Giving the ARIMA model additional information to make multivariate forecasts (ARIMAX)
outperforms all other methods by a large margin. This outcome results from the very small
data set of 180 data points. Random forest and neural network models do apparently not
generalize well enough when trained with this limited amount of data. Traditional time series
modeling with ARIMAX seems to handle this problem much better and is therefore the best
option to make a forecast for this specific task.
Throughout the evaluation we did not consider the forecast for $s_{t+1}$. For completion, table
23 provides these numbers for the Total scenario. The figures for the neural network and the
random forest were given by the same models we chose in the final results section according to
the lowest validation error for $s_{t+2}$. Thus, there might be models from our grid search which
have a lower relative error for $s_{t+1}$. All errors are again test set errors. Interestingly, ARIMA,

<table>
<thead>
<tr>
<th></th>
<th>Neural net</th>
<th>Rand. forest</th>
<th>ETS</th>
<th>ARIMA</th>
<th>ARIMAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPE</td>
<td>7.67%</td>
<td>8.21%</td>
<td>13.69%</td>
<td>14.68%</td>
<td>9.59%</td>
</tr>
<tr>
<td>SD</td>
<td>8.36%</td>
<td>8.38%</td>
<td>9.86%</td>
<td>11.78%</td>
<td>8.36%</td>
</tr>
</tbody>
</table>

Table 23: Forecast result for the next month.

ETS, and ARIMAX do not improve their predictive accuracy having basically the same errors
and standard deviations. The neural network and the random forest regression, however, do
significantly better. ARIMAX may again outperform all other methods with different settings
if we change the regressor variable for scheduled assemblies $c_{t-1,2}$ to $c_{t-1,1}$ in order to relate
to the immediate next month (see Section 3.4.1).
Chapter 5

Conclusion

Summary
This master thesis has explored the applicability of different algorithms for time series forecasting. In Chapter 2 we provided the fundamental theory to approach the topic. The first step in Chapter 3 was to analyze the very small data set of only 180 data points. Then, we applied the necessary pre-processing steps to prepare the data for the neural network and random forest models. In a next step we performed an experimental grid search for both algorithms to find the most appropriate hyperparameter settings. Similarly, we prepared the data for the time series analysis techniques exponential smoothing, ARIMA, and its variation ARIMAX, and developed forecast models. In Chapter 4 we analyzed the outcome of all experiments. The evaluation showed that the forecast of sales figures for all systems combined (Total) worked significantly better than predicting numbers for individual systems (Single Device) according to our measurement criterion, the mean absolute percentage error. Regarding the forecast of all systems combined (Total) ARIMAX outperformed all other methods with the lowest forecast error of 9.55%. The neural network models, exponential smoothing, and ARIMA provided results in a similar error range between 13% and 14%. The random forest however, had the highest predictive error of 15.13%. Predicting future sales figures for individual system types (Single Device) resulted on average in errors well above 25% for all methods. Thus, the research question of this thesis may be answered as follows. Considering our specific problem the classical statistical approaches to model time series are superior to the machine learning algorithms for two reasons. First, they provide similar or much better results (ARIMAX) than the neural net. Second, the models of ETS, ARIMA, and ARIMAX have fewer variables than the neural networks (think of all the weights and biases) and are therefore simpler which makes them less vulnerable to overfitting. Altogether, the very limited amount of data prevented the machine learning models from delivering better results. However, all methods would benefit if more data was available to make more robust forecasts.

Real World Contribution
Our best forecast with ARIMAX to predict the sales figure for all systems (Total) has a real value for the supply chain management (SCM) of Advanced Therapies. The relative error of 9.55% is within range of the human forecast (7.5%) but the forecast numbers are two weeks earlier available. Using the tool we implemented allows convenient forecasting without any effort. It requires only adding the newly available data to the folder structure. The forecasting software may also provide future estimates for individual systems. However, this functionality
is switched off due to the much better human forecast with a relative error of 15.5% and a standard deviation of only 5.9%.

**Future Work**

Regarding the approach of this thesis several methods could be adjusted to rerun the experiments. Regarding the neural network models to predict individual system types more complex topologies would be a good option. Also, increasing L2 regularization might have an effect to lower the out-of-sample error. Moreover, training individual models for each system type is an option. Yet, the problem of having only a very small data set remains. Another way to improve the forecast performance of both ARIMAX and the machine learning techniques is to find further external factors that correlate with the sales figures. Also, there is more data on individual orders available from the SCM. This information could be included in future approaches. Increasing the dimensionality of the data points could be countered with a principal component analysis. There might also be potential in trying more sophisticated ensembles to make a forecast. Finally, the approach of modeling recurrent neural networks with long short-term memory may be evaluated despite having such a small data set.
Bibliography


BIBLIOGRAPHY


Appendix A

Experiments

Figure 45: Training with early stopping (NN Total)
Figure 46: Training without early stopping (NN Total)
Figure 47: Training single-hidden layer (NN Single Device).
Figure 48: Training four-hidden layer (NN Single Device).
Figure 49: Augmentation effect filtered by topologies (NN Single Device).
Figure 50: Dropout effect filtered by topologies (NN Single Device).
Figure 51: Early Stopping effect by topologies (NN Total).
Index

ACF, see Autocorrelation function
Activation function, 5
AIC, see Akaike information criterion
Akaike information criterion, 12
ANN, see Feedforward neural net
AR, see Autoregressive terms
ARIMA, 15, 33
ARIMAX, 19, 32
Augmented Dickey-Fuller Test, 35
Autocorrelation function, 17
Autoregressive terms, 15

Backlog, 23
Backpropagation algorithm, 5
Bias, 4
Bias-variance trade-off, 22
Bootstrap, 10, 31
Box-whiskers plot, 39

CART, 9
Classification, 3
Confidence interval, 17
Correlation, 25
Cross-validation, 29

Data augmentation, 7, 26
Decision tree, 8
Differencing, 16
Dropout, 7

Early stopping, 7
Ensemble, 20
Error, 6
ETS, see Exponential smoothing
Exponential decay, 30
Exponential smoothing, 12, 33

Feedforward neural net, 3, 30
FNN, see Feedforward neural net
Gradient descent, 6
Hyperparameters, 30
Imputation, 24
Indicator, 9
Least squares, 11
Log likelihood, 11
Loss, 6

MA, see Moving average terms
Machine learning, 3
MAPE, 23
Maximum likelihood, 11
Missing data, 24
Moving average terms, 16
Multiple regression, 19

Naive forecast, 60
Neuron, 4
NN, see Feedforward neural net
Normal distribution, 10
Normalization, 27
One-hot vector, 26

PACF, see Partial autocorrelation function
Partial autocorrelation function, 17
Prediction interval, 12

Random forest, 10, 31
Regression, 3
Regularization, 7
Rolling forecast, 32

Sales, 23
Scheduled, 23
Simple exponential smoothing, 13
Single device, 24, 46, 54
Affirmation

Ich erkläre hiermit gemäß § 17 Abs. 2 APO, dass ich die vorstehende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Bamberg, den 5. Juni 2018

____________________________
Unterschrift