Time series based anomaly detection in an event driven environment by means of machine learning

Zeitseriengestützte Anomaliedetektion in einer ereignisgesteuerten Umgebung mittels maschinellem Lernen

Teresa Auerbach

A thesis presented for the degree of
Bachelor of Science

Supervised by:
Professor Sarah Brockhaus

Hochschule für angewandte Wissenschaften München, Deutschland

August 2023

I, AUTHORNAME confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

The influence of artificial intelligence has increased significantly in many areas of our lives. We constantly encounter machine learning in our everyday lives. Be it image recognition, autonomous driving or the prediction of stock prices, for example.

Due to the growing presence of this topic, anomaly detection is increasingly coming into scientific focus. Anomaly detection is used to report abnormal behavior because anomalies show that something different is happening than expected. These anomalies are not necessarily good or bad, but they should be detected to assess whether or not any aytion needs to be taken.

The focus in this bachelor thesis will be on how three different machine learning models can be trained with time series based data and how well they are suited for anomaly detection. The aim of this research is to provide an overview of these three models. They were trained and analyzed regarding their results in detecting anomalies.

This thesis is structured as follows: The first chapter provides the theoretical knowledge to the Autoregressive Integrated Moving Average model

and how to use it in Python. Afterwards in the second and third chapter, the Feedforward Neural Network and the Long Short Term Memory model are explained and the usage in Python is also shown. These chapters are followed by a chapter on the comparison of the three models in terms of their accuracy in predictions. Thereafter, the anomaly detection of these models is analysed. The last chapter of the thesis, a conclusion, contains a summary and ideas for future work on that topic.

Acknowledgements

Table of Contents

List of Figures

List of Tables

Abbreviations

ARIMA	**A**uto- **R**egressive **I**ntegrated **M**oving **A**verage
FFNN	**F**eed**F**orward **N**eural **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory
MSE	**M**ean **S**quared **E**rror

Chapter 1

The ARIMA model

The ARIMA model consists of three different models combined into one and can be used to make predictions.

1.1 AR component

The AR part of ARIMA stands for “Auto-Regression”. Values from previous time stamps are used to predict the value for the next time stamp. The parameter **p** determines how many past values are used. Thus, the formula for the AR model can look like this (Hyndman & Athanasopoulos 2018a):

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \quad + \phi_p y_{t-p} + \varepsilon_t \tag{1.1}$$

The component c is a constant and can be used to shift the outcome of the prediction. If not desired, it can be set to 0 and be left out.

The component ε_t is white noise.

1.2 I component

The I part stands for "Integrated" and is described by the integration of the differentiation terms of the time series to establish data stationarity. The paramter **d** indicates how often this operation is performed. If the time series is not stationary i.e. its statistical properties such as mean and variance vary over time, they must be stabilized by differentiation to allow stable prediction.

This part can be described with the formula (Schaffer et al. 2021):

$$y'_t = y_t - y_{t-1} \tag{1.2}$$

1.3 MA component

The MA part stands for "Moving Average". Previous forecast errors are used to predict the next value. The parameter **q** determines how many past forecast errors are used. Thus, the formula for the MA model can look like this (Hyndman & Athanasopoulos 2018b):

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ + \theta_q \varepsilon_{t-q} \tag{1.3}$$

As in the AR model, the component c is a constant to shift the outcome and the component ε_t is white noise.

1.4 Determing the parameters

There are various ways to determine the parameters. One way would be to use random numbers in a certain range.

Another way is to perform a grid search. That means after defining a certain range for each parameter, an error value for comparison is calculated from all possible combinations and the combination with the lowest error value will be used.

In the following, one mathematical approach for each paramter will be described.

1.4.1 Parameters $\phi_1, ..., \phi_p, \theta_1, ..., \theta_q$

The parameters $\phi_1, ..., \phi_p, \theta_1, ..., \theta_q$ can be determined by using maximum likelihood estimation (MLE). This method tries to find the parameter values that minimize the distance between the observed values and the predicted values.

1.4.2 Parameter p

To choose a suitable value for the parameter **p** it is possible to choose the most significant lags in the partial autocorrelation function plot. This function is estimated by the partial correlation of values after controlling for the effects of other variables (Zvornicanin 2023).

1.4.3 Parameter d

The paramter **d** is the value at which the time series becomes stationary and the ACF plot and PACF plot of the differentiated time series no longer show significant autocorrelations.It can be determined by checking test statistics such as the ADF test (Augmented Dickey-Fuller Test). When the value returned from that test is greater than 0.5 the time series is non-stationary and more differenciating is needed.

1.4.4 Parameter q

Similar to the parameter p, the parameter **q** can be chosen by the most significant lags in the autocorrelation function plot. This plot shows the correlation of the values in a time series (Zvornicanin 2023).

1.5 ARIMA library method

The library method of ARIMA can be used by importing:

```
from statsmodels.tsa.arima.model import ARIMA
```

To generate an ARIMA model, it is necessary to pass the data which should be used for the training and the parameters p, d and q to the method.

Therefore, a simple model with data called train_data and the parameters p, d and q as 1,1 and 1 can be generated by a code looking like in Listing 1.1:

Listing 1.1 ARIMA example

```
model = ARIMA(train_data, order=(1,1,1))
model_fit = model.fit()
forecast = model_fit.predict(start=100, end=200)
```

This code snippet also fits the model by calling the *fit()* method and calculated a forecast. This prediction can be made by using the *predict()* method. To indicate the start and end of the desired prediction, these values can be passed into the method.

There are also optional parameters that can be passed into the *ARIMA()* method to modify the model. For example, the parameter **enforce_stationarity** indicates whether the model should be set to produce stable and predictable results by adjusting the autoregressive parameters to match a stationary process (Fulton 2023).

Chapter 2

The feedforward neural network

The FFNN, also known as Multilayer Perceptron, is a type of an artificial neural network that consist of multiple layers of neurons connected by weighted connections.

2.1 Structure of the model

The authors of the article "An Introductory Review of Deep Learning for Prediction Models With Big Data" explain the FFNN as follows (Emmert-Streib et al. 2020):

In this neural network, all perceptrons are arranged in layers, with the input layer receiving the input and the output layer producing the output. The layers between the input and the output layer which have no connection to the outside are called hidden layers. The FFNN doesn´t have feedback to the previous layers or perceptrons in the same layer. The number of layers is called depth and the number of neurons width. The concept of an FFNN is graphically shown in Figure 2.1.

An FFNN has at least one hidden layer (Ko 2017).

2.2 Training mechanism

According to Terry-Jack, the perceptrons works as follows (Terry-Jack 2019):

- a perceptron takes some values as input
- each of these values are multiplied by values between 0 and 1 which are called weights
- these weighted values are added up
- a bias is added to the sum
- depending on which activation function was chosen, the output value of the perceptron equals the result of the activation funtion with the sum as an input

The output value can be an input value for another perceptron.

2.3 FFNN library method

The library method of a FFNN can be used by importing:

```python
from tensorflow import keras
```

The following coding part is an example of how to use this library method.

Listing 2.1 FFNN example

```python
input_shape = (X_train.shape[1],)

model = keras.Sequential()
model.add(keras.Dense(500, activation='relu', input_shape=input_shape))
model.add(keras.Dense(units=500, name = "HiddenLayer1", activation="relu"))
model.add(keras.Dense(units=96))

model.compile(optimizer='adam', loss='mse')

model.fit(x = X_train, y = Y_train, epochs=1000, batch_size=512)

y_pred = model.predict(X_test)
```

In the first line of Listing 2.1, the input shape that equals the numbers of columns is returned. That number represents the number of features that will be integrated into the model.

After that, the input layer of the model is created. In the example above, that layer has 500 neurons and as an activation *relu* is chosen.

This model has one hidden layer which also has 500 neurons. As an activation, *relu* is chosen.

The output layer has 96 units. 96 is number of values that will be predicted.

After the layer definition, the model is compiled with *adam* as an optimizer and *mse* as the loss parameter.

Then, the model can be fitted. The x parameter represents the data based on which the model should predict the y parameter. Parameters as the number of epochs and the batch size can also be passed on into the fit()-method. In this case, the epochs are set to 1000 and the batch size is 512.

Finally, a prediction can be made about the test data which can be compared to the expected results to determine the accuracy.

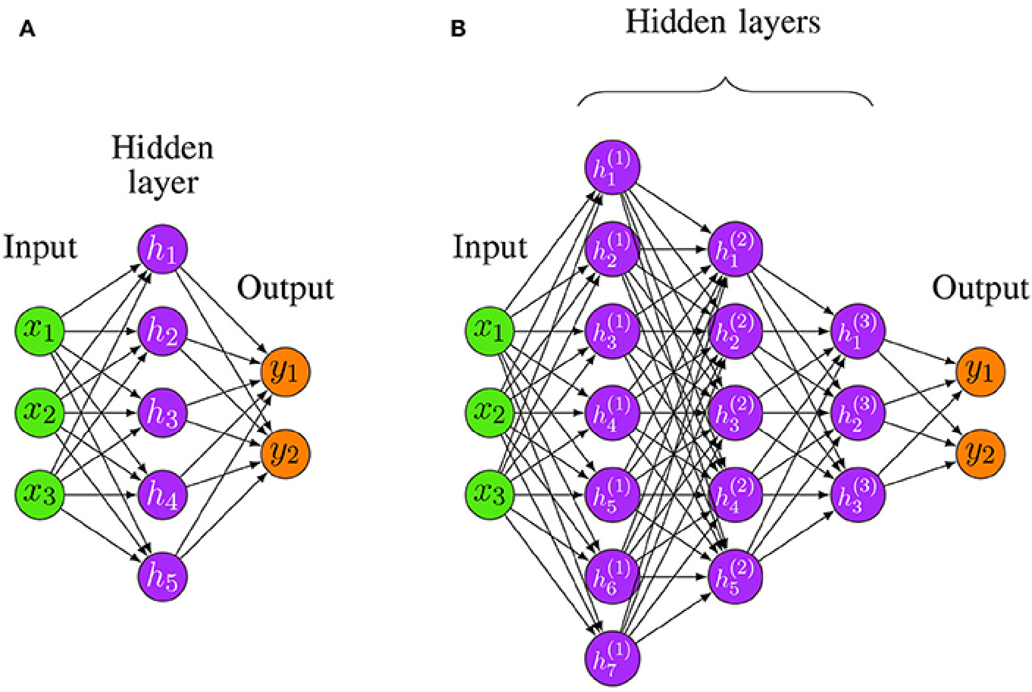

Figure 2.1: Exemplary illustration of two FFNN structures

Chapter 3

The long short-term memory neural network

An LSTM is a type of recurrent neural network designed specifically for sequential data. Compared to an FFNN, which has only a fixed number of inputs and outputs, an LSTM can handle a variable number of inputs and outputs.

3.1 Idea

Luber and Litzel desribed the LSTM as followed (Luber & Litzel 2018):

The LSTM is comparable to a human brain. Recurrent neural networks have the possibility to fall back on already gained knowledge. They have so called feedback loops. These store the already learned information in the neural network. The more layers and complexity a neural network has and the longer it has been trained, the more interlinked feedback loops are created. But this leads to the fact that certain information and experiences in deep layers can no longer be retrieved efficiently.

A LSTM solves this problem. It creates a long-lasting memory of previous experiences that is easily retrievable in the neural network. That makes it is particularly good at detecting and learning long-term dependencies.

3.2 FFNN library method

The needed import to use the library method is the same as the one mentioned in Section 2.3.

The following coding part is an example of how to use this library method.

Listing 3.1 LSTM example

```
input_shape = (X_train.shape[1], 1)

model = keras.Sequential()
model.add(keras.layers.LSTM(5, input_shape=input_shape))
model.add(keras.layers.LSTM(5))
model.add(keras.layers.Dense(units=96))

model.compile(optimizer='adam', loss='mse')

model.fit(x = X_train, y = Y_train, epochs=1000, batch_size=128)

y_pred = model.predict(X_test)
```

In the first line of Listing 3.1, the input shape is defined. An LSTM expects the input shape to be in the following form: (number of samples, number of time steps, number of features).

After that, the input layer of the model is created. Its type is a LSTM layer. In this example, that layer has 5 neurons.

The next LSTM layer of the model has 5 neurons.

The output layer has 96 units for the 96 values that will be predicted.

Then, the model is compiled with *adam* as an optimizer and *mse* as the loss parameter.

After that, the model can be fitted similarly to the FFNN and the predict() method is also used in the same way.

Chapter 4

Comparison of the three methods

4.0.1 MSE

As a method to compare the results of the different models, I will use the MSE. The formula looks as follows (Glen n.d.):

$$MSE = (\frac{1}{n}) * \Sigma(x_i - y_i)^2 \tag{4.1}$$

This formula sums the squared deviations between the actual values and the forecasted ones and dividing that by the number of values.

Due to the squaring, the result is always positive.

The MSE decreases when the model better predicts the observed data. So a result as low as possible is desired.

To use the library method to calculate the MSE, the following import is needed:

```
from sklearn.metrics import mean_squared_error
```

The MSE can be easily calculated by passing the actual data and the forecasted data into the method:

Listing 4.1 MSE calculation

```
mse = mean_squared_error(test_data, forecast)
```

4.1 Data

The data that I used to train the models with are temperature values. Each temperature value has its time stamp.

The first time stamp is the 28th of February 2023 19:57:15 and the last one is the 25th of May 2023 09:36:16.

The time stamps are mostly spaced at a distance of 5 minutes but there are irregularities like bigger gaps of missing data or more than one value within five minutes.

In total, the data set contains 17321 data points.

Plotting the data without any preprocessing steps delivers the plot shown in Figure 4.1.

To work with the data, I used a dataframe that contains the temperature values in a column and the time stamps as the index.

4.2 Data preprocessing

Looking at Figure 4.1 and the data points itself, it is clear to see that there is an odd temperature value with below negative five degrees. Another noticable thing are some gaps which appear as horizontal lines on the graph.

To improve these two issues, I removed all data points with a temperature below ten degrees. Furthermore, I seperated the dataframe into a list of dataframes for each day. From this list, I removed all the days that have less than 150 datapoints to make sure I only make predictions based on days with more than 50 % of the perfect data. The ideal number of data points would be 288, 12 every hour for the whole day.

To assure that each of the dataframes contains the same count of temperature values, I used the resample() method with an interval of five minutes. After that, I used the fillna() method with the argument *ffill* to get rid of all of the remaining gaps. *ffill* is a forward fill which means that the last valid temperature value is updated forward to the next valid temperature value.

Finally, I divided the list of dataframes so that 80 % of it are forming the dataset training and the rest belongs to the test dataset.

4.3 ARIMA results

First of all, I performed a grid search to find the best values for the parameters p, d und q. As an error value that allows me to compare all of the possible combinations I used the MSE (Equation 4.1).

The lowest MSE value was calculated for the numbers 0, 2 and 1. Therefore I used these to pass into the ARIMA model generation.

To make sure that the time series is stationary and no more differenciating is needed as described in Section 1.4.3, the Augmented Dickey-Fuller Test is performed. The result was less than 0.5 which means that the parameter 0 is suitable for the data.

To predict the temperature developement, I passed the preprocessed dataframes into the ARIMA method and started predicting after two hours. The prediction is a rolling forecast which means that besides the two hours, all of the previous values are also included.

The plots shown in Figure 4.2 and Figure 4.3 are the predictions compared to the actual values for two days. Figure 4.2 has the highest measured MSE value and Figure 4.3 the lowest.

The make the results comparable to the other models, I added all of the MSE values and also the average error value for one day. The added up MSE value is 0.08378079892276652 which makes it an average of 0.0013963466487127753 a day.

As a summary, ARIMA predictions were really accurate with MSEs less than 0.01.

4.4 FFNN results

My first approach to make predictions using an FFNN was to split the data similar to the ARIMA approach. So I created two columns in the dataframe. One had the datapoints from before 4 p.m. and the other one the ones after 4 p.m. . My intention was to predict the temperature after 4 p.m. based on the previous values for each day.

The results are shown in Figure 4.4. The red lines are the actual values, the blue ones the predicted ones. The added up MSE value is 16.726520374860815 and the average per day 1.286655413450832.

Since these results were not satisfying loooking at the huge difference in the graph as well as an MSE value above 1, I tried a different approach and seperated the data differently. This time, complete previous days were used to predict a whole day.

The results for taking 3 previous days into the prediction are plotted in Figure 4.5. As in the previous plot, the red lines are the actual values and the blue ones are the predicted ones. This time, the added up MSE value is 8.557747080099956 and the average per day 0.8557747080099956.

This is definitly an improvement compared to the first approach.

In the third approach, I added other features to the neural network.

One feature that I chose is the calendar week. I added the number depending on which week of the calendar the temperature value was recorded. The intention was to add an influence over the year into the prediction to consider the yearly temperature.

Adding the calendar feature didn´t bring any improvement regarding the MSE value. Against my expectations, the results were 38.077699477978896 for the added up MSE value and 3.8077699477978895 for the average.

As seen in Figure 4.6, the blue predicted lines even reach the 25 degrees. That is roughly 2 degrees more than the red lines reach as a peek.

Another feature that I chose to try out was the time of the day. The intention there was to add something that gives structure troughout the day.

The results for that can be seen in Figure 4.7.

The calculated error values were as followed: the added up MSE was 13.077899275853396 and the average MSE value 1.3077899275853395.

4.5 LSTM results

For the LSTM neural network, I used the same four options as for the FFNN.

The first implemented way was the split into before and after 4 p.m. . addedMse 130.9609446831818 counter 13 average 10.073918821783215

The next prediction was based on the 3 previous days.

addedMse 12.028280919440851 counter 10 average 1.202828091944085

The third prediction has the calendar weeks as a additional feature.

The last prediction has the time of the day as a second feature.

4.6 Key comparison points

not always same results

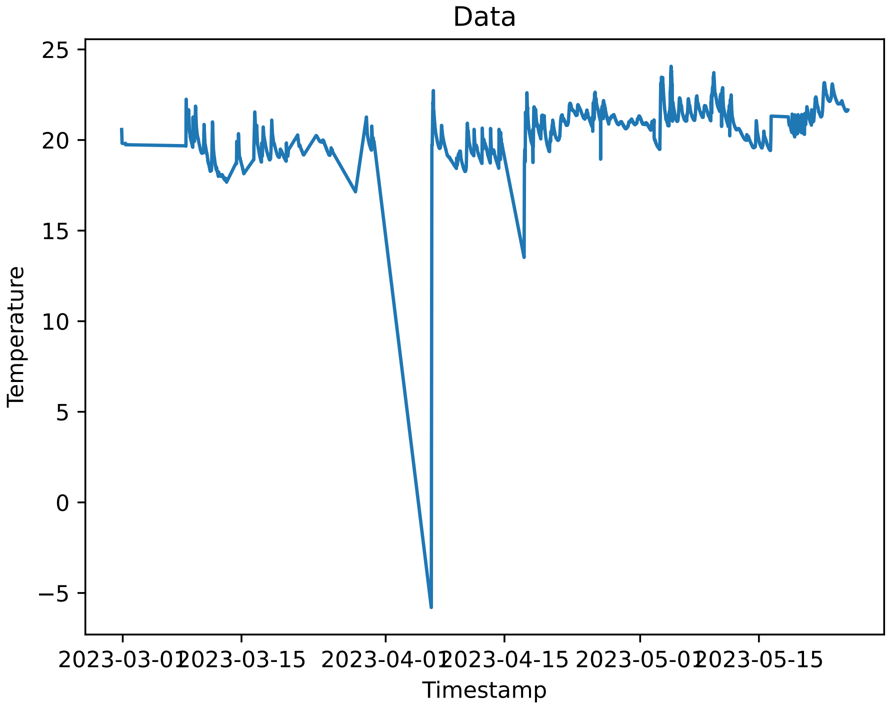

Figure 4.1: Plot given data

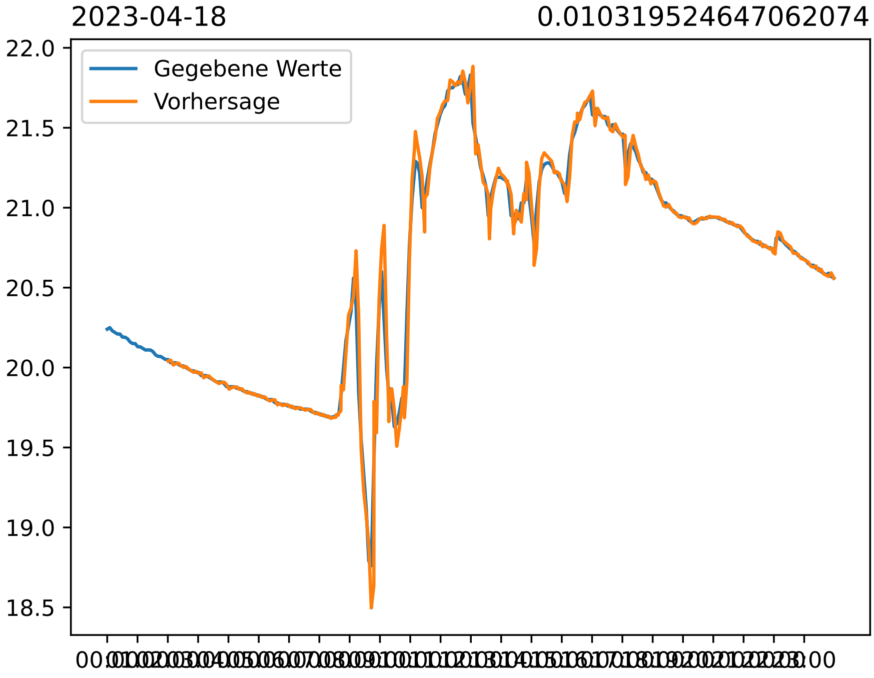

Figure 4.2: ARIMA prediction for 2023-04-18

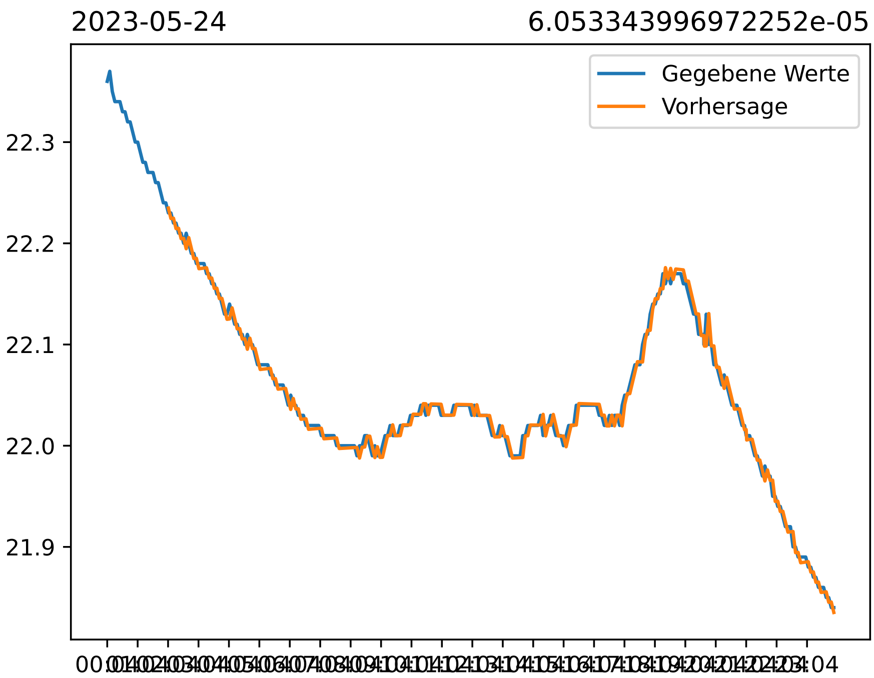

Figure 4.3: ARIMA prediction for 2023-05-24

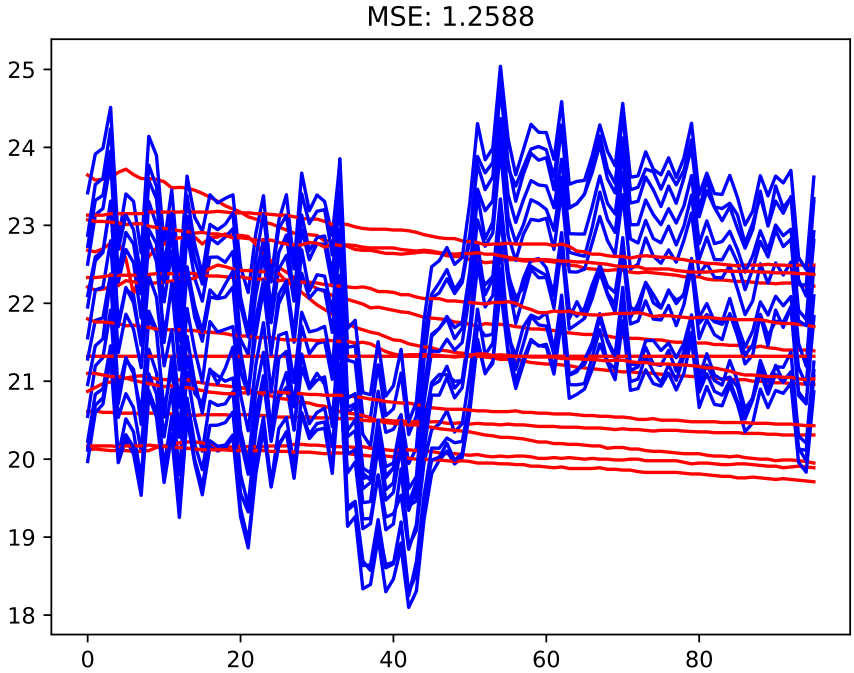

Figure 4.4: FFNN prediction from 4 p.m.

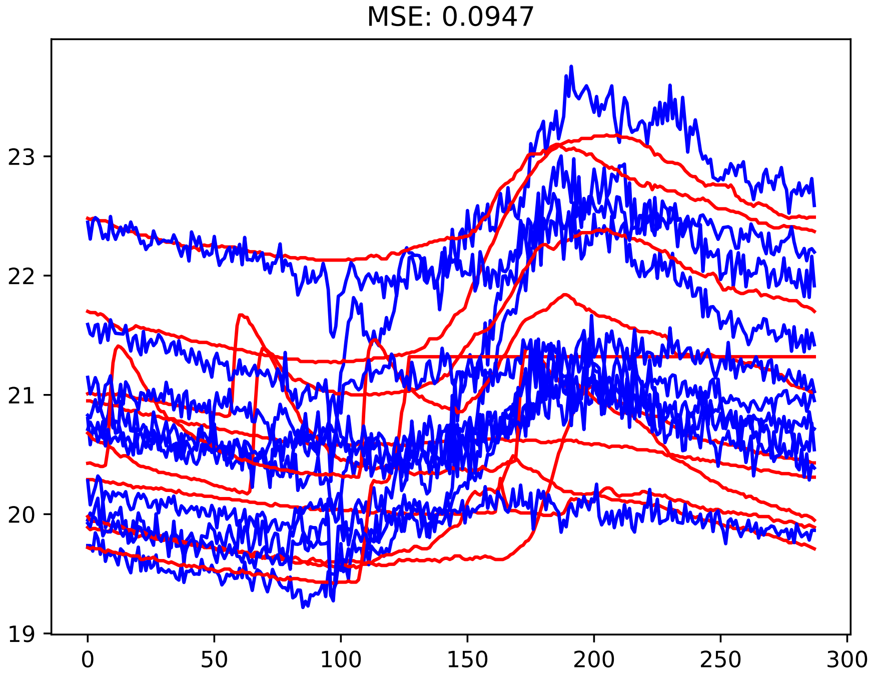

Figure 4.5: FFNN prediction 3 previous days

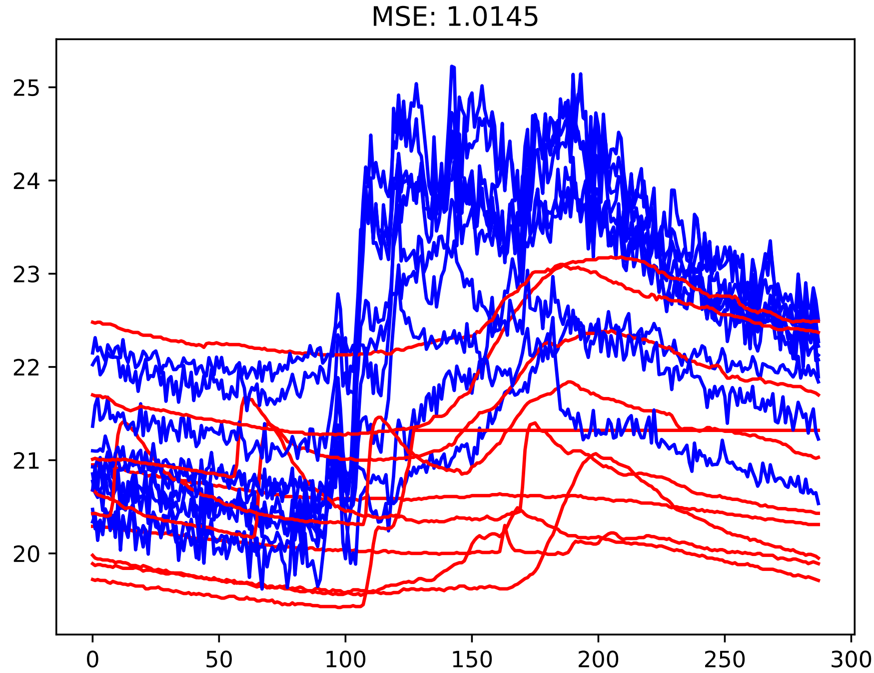

Figure 4.6: FFNN prediction with calendar weeks

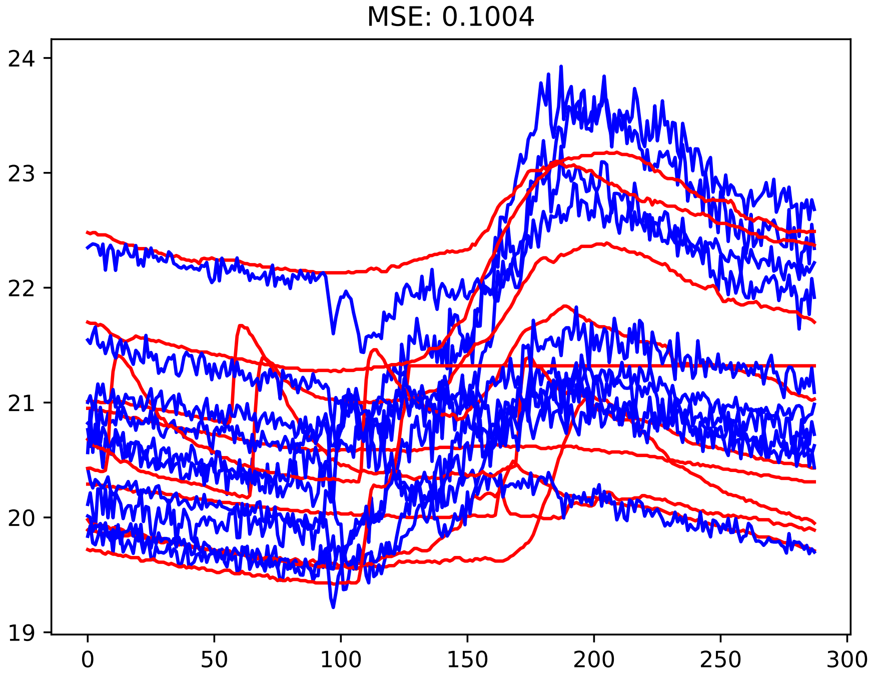

Figure 4.7: FFNN prediction with time of the day

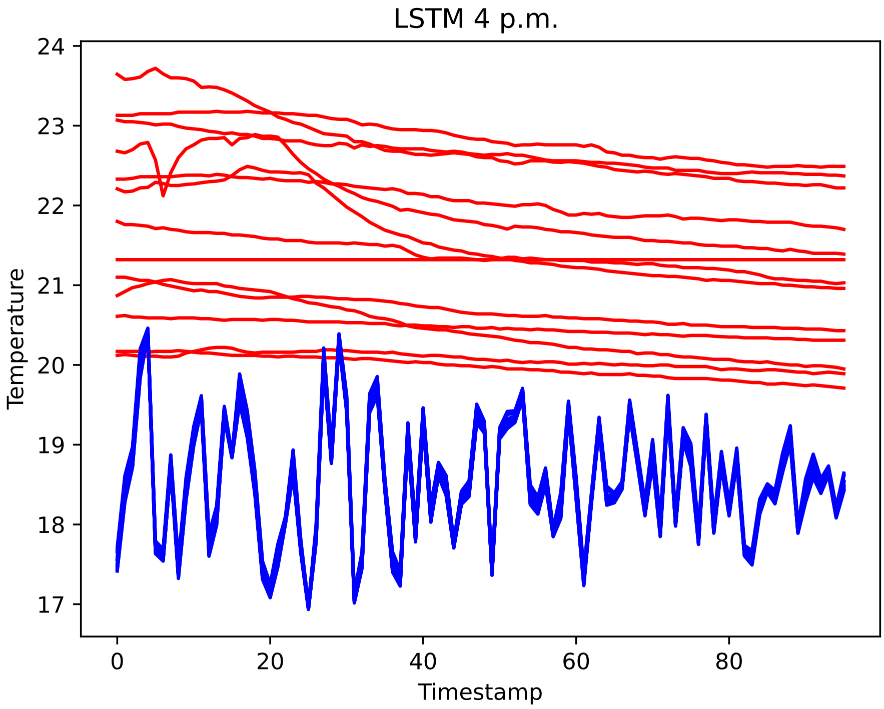

Figure 4.8: LSTM prediction from 4 p.m.

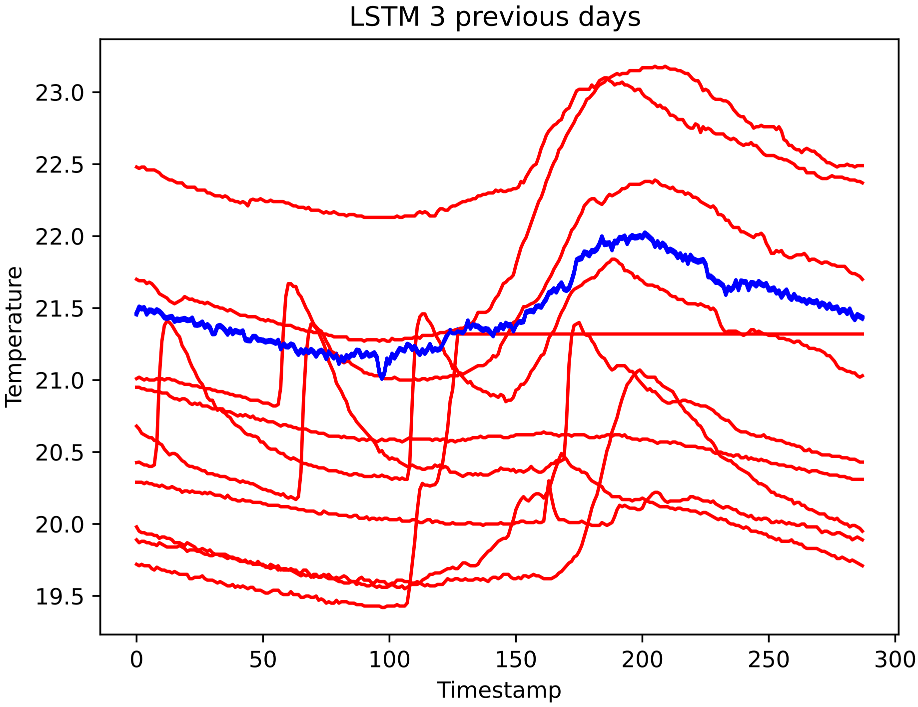

Figure 4.9: LSTM prediction 3 previous days

Chapter 5

Anomaly detection

5.1 Anomaly definition

5.2 ARIMA

5.3 FFNN

5.4 LSTM

Außreiser vs Anomalie

Chapter 6

Conclusion

Appendix 1: Some extra stuff

References

Emmert-Streib, F. et al., 2020. An introductory review of deep learning for prediction models with big data. *Frontiers in Artificial Intelligence*, 3. Available at: https://www.frontiersin.org/articles/10.3389/frai.2020.00004 [Accessed July 4, 2023].

Fulton, C., 2023. Class ARIMA(sarimax.SARIMAX). Available at: https://github.com/statsmodels/statsmodels/tree/56e9c569fadb2805eeb3c64989ea122ea85a95af/statsmodels/tsa/arima [Accessed July 2, 2023].

Glen, S., Mean squared error: Definition and example. StatisticsHowTo.com. Available at: https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-squared-error/ [Accessed June 28, 2023].

Hyndman, R.J. & Athanasopoulos, G., 2018a. 8.3 autoregressive models. In *Forecasting: Principles and practice*. Melbourne, Australia: OTexts. Available at: OTexts.com/fpp2 [Accessed May 28, 2023].

Hyndman, R.J. & Athanasopoulos, G., 2018b. 8.4 moving average models. In *Forecasting: Principles and practice*. Melbourne, Australia: OTexts. Available at: OTexts.com/fpp2 [Accessed May 28, 2023].

Ko, B., 2017. Feed-forward neural network (FFNN). Https://kobiso.github.io//. Available at: https://kobiso.github.io//research/research-ffnn/ [Accessed July 3, 2023].

Luber, S. & Litzel, N., 2018. Was ist ein long short-term memory? BIGDATAINSIDER. Available at: https://www.bigdata-insider.de/was-ist-ein-long-short-term-memory-a-774848/ [Accessed July 18, 2023].

Schaffer, A.L., Dobbins, T.A. & Pearson, S., 2021. Interrupted time series analysis using autoregressive integrated moving average (ARIMA) models: A guide for evaluating large-scale health interventions. *BMC Med Res Methodol*, 21(58).

Terry-Jack, M., 2019. Deep learning: Feed forward neural networks (FFNNs). Medium. Available at: https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a309d [Accessed July 13, 2023].

Zvornicanin, E., 2023. Choosing the best q and p from ACF and PACF plots in ARMA-type modeling. Baeldung. Available at: https://www.baeldung.com/cs/acf-pacf-plots-arma-modeling [Accessed May 28, 2023].