Uncertainity in Renewable Energy Time Series Prediction using Neural Networks

Phil Aupke

Faculty of Health, Science and Technology
Computer Science Department
Master Thesis: 30 ECTS
Supervisor: Dr. Andreas Kassler
Examiner: Dr. Johan Garcia
Karlstad, 26th January 2021

I Table of Contents

II List of Figures

III List of Tables

IV Table of Listings

V List of Abbreviations

AnEn	**Analog Ensemble**
ANN	**Artificial Neural Network**
ARIMA	**Autoregressive Integrated Moving Average**
FFNN	**Feed Forward Neural Network**
GBR	**Gradient Boosting Regression**
GBRT	**Gradient Boosted Regression Trees**
ICT	**Information and Communications Technology**
IQR	**Interquartile Range**
JSON	**JavaScript Object Notation**
LSTMN	**Long Short-Term Memory Network**
MLP	**Multilayer Perceptron**
NaN	**Not a Number**
NWP	**Numerical Weather Prediction**
PCA	**Principal Components Analysis**
PV	**Photovoltaics**
RE	**Renewable Energy**
RFE	**Recursive Feature Elimination**
RQ	**Rolling Quantile**
RT	**Regression Trees**
SSL	**Secure Sockets Layer**
SVM	**Support Vector Machines**
TDNN	**Time Delayed Neural Network**

Abstract

With the increasing demand for solar energy, the forecast of the PV station energy production has to be as precisely as possible. To make the prediction more robust, also correlated information about the weather can be added to the previous energy production of the PV station. This thesis is part of a project, which has the goal to build an energy marketplace for a smart energy grid between households. To make the decisions of the prosumer more accurate, a forecast for the PV station energy production has to be as accurate as possible. Because not every household or even some smart grids will contain a weather station, also interpolated weather information has to be considered. The objective of this work is the evaluation of the accuracy difference between precise weather information, located directly at the PV station and interpolated weather data.

The errors of the data were recorded due to misfunctions in the sensors and were cleared with the usage of winsorization. The unnecessary weather features have been detected with several feature selection methods. For the forecast of the energy production three established machine learning algorithms were used: Random Forest, LSTM and Facebook Prophet. For the comparison of the performance different performance metrics were used. The validation of the three models was carried out by a walk-forward cross validation with unseen data. Furthermore, for each of the two datasets one of the three machine learning model were trained. For the performance measurement i.e., the LSTM model trained on precise weather information also received the interpolated data as an input for the prediction and vice versa. As a conclusion, the Random Forest model performed better than the other two model types, with an average normalized error of 0.15. Whereas the LSTM model received an error of 0.37 and the Prophet model 0.58. For the difference between interpolated and actual weather information the results prove, that the uncertainity in those variables also affects the prediction of the PV station energy outcome. The LSTM model MSE increased by 14 percent and the Random Forest results with an increasement of 16 percent. The end of the thesis includes a discussion about the results and possible tasks for future work takes place.

1 Introduction

Due to the increasing usage of renewable energy sources around the globe, the development of *photovoltaic panels* (PV) has intensified significantly over the last few years. Because of the increased development the costs of the PV systems dropped as well. This is presented in the IRENA report of 2017 which concludes, that the levelized cost of large-scale PV systems dropped by 73% from the years 2010 – 2017 [IR18]. Because of the increasing usage of PV systems, it is also necessary to predict the upcoming energy outcome of them as precise as possible. But due to the fact, that the PV systems are depending on the weather which has different random parameter, like: irradiance, relative humidity and ambient temperature which can affect the power outcome of the PV.
To predict the weather and even the corresponding power outcome of the PV stations there are two approaches, the deterministic or probabilistic concept or by using machine learning for time series data. Both concepts have their positive and negative characteristics for this kind of usage.

1.1 Description of the Thesis

This thesis is going to be an introductory work for an upcoming project which is going to integrate *renewable energy* (RE) into power grids and also use the advancements of *information and communication technologies* (ICT) like the cloud or edge computing. Those technologies have high potential in the development of smart energy grids. In those smart energy grids, a customer has the overview of all steps like the production, consumption and also the storage capacities. In this system the prosumer has an active role in the reduction of CO2 for this smart energy grid. Furthermore, multiple machine learning techniques will be evaluated, which will optimize the energy management as well as give the prosumer future predictions of the weather and also the estimated PV system energy outcome. Those techniques will help the prosumer to make decisions for the energy usage of himself and also the distribution of his surplus energy to another prosumer.

1.2 Objectives of the Thesis

One of the objectives of this work is the evaluation of different machine learning methodologies like *LSTM*, *Random Forest* and the *Facebook Prophet Library* for the two use-cases of weather forecasting and also the corresponding PV system energy outcome. In Addition to that, an evaluation of the accuracy comparison between the usage of interpolated and precise weather information takes place. This evaluation will indicate how much better a machine learning model can perform, if the different weather information are interpolated or received from the same location as the PV system itself.

1.3 Research Question

- **RQ1** How can the PV system energy outcome be proactively determined using machine learning model and weather information?
- **RQ2** Do interpolated weather information affect the future prediction of PV energy outcome?

1.4 Ethics and Sustainability

In regard to the scope of this thesis, there are no ethical concerns. With reference to any economic sustainability the results of this work will contribute to a project, which aims toward a more sustainable energy exchange between prosumer within a smart energy grid. The goal of smart energy grids is a smarter energy exchange between different households which uses renewable energy productions within the grid, so they do not need that much energy from the main grid, which produces only some parts with renewable energy. For Sweden the amount of renewable energy produced for the main grid is 54 percent and for some other countries even less. [@Swe20]

1.5 Structure of the Thesis

This thesis is divided in seven subareas. The second chapter describes the state of the art of current research in the field of weather forecasting as well as predicting the PV system power outcome. Furthermore, this chapter presents current techniques for weather and PV energy outcome. One method is to use different machine learning model for the prediction and the other one is the usage of numerical weather forecasting, which uses deterministic or probabilistic methodologies.
Afterwards in the third chapter all used methodologies are going to be explained in detail. This includes the data preparation, feature selection and the actual ML-Model which were used. The fourth chapter describes the implementation of the different stages of the machine learning pipeline, which includes the data preparation, feature selection and the training and hyperparameter tuning of each model type.
The fifth chapter concludes the evaluations of the machine learning model in the different use-cases. These use-cases involve the comparison between model trained with interpolated

and actual data. The last chapter describes a discussion about the collected results from the previous chapter.

2 State of the Art

The field of predicting time series data with machine learning is a field of research in which many contributors work on. Furthermore, the need for predicting the weather and the corresponding supply of renewable energy with it, is crucial. This chapter concludes the research in the field of predicting the energy outcome of PV Systems and the handling of time series data in general.

2.1 General Research

The paper [KP11] concludes a summary of techniques for the prediction of PV power outcome and also states, that there should be an industrial standard for this kind of technology. Furthermore, the authors and some other studies found out, that the use of NWP can help the long-term prediction of weather forecasting and accompanying the prediction of PV power outcome. In addition to that the thesis states, that the predictions can be improved when the machine learning model is trained with local weather situations and not simultaneously with different situations around the globe. This scheme can also be applied to different weather situations over the year. The model performs better if it is just trained for one specific season.
The conclusion from [KP11] is, that both techniques, the classic time series techniques and also the machine learning models have been widely used for weather forecast and also the prediction of the PV power outcome. For this thesis only the machine learning model is going to be examine in depth. But for the overall view of the current research on weather and PV system output forecast also the traditional time series techniques are going to be presented.
Besides weather forecasting, the prediction of time series data is also useful in many other applications like the financial sector [K03] or even for the event detection [GS99]. There are many other use cases for this kind of prediction.
To conclude how the prediction of the PV System outcome works in general, there are two different steps which are presented in Figure 1. The first step is the analysis of current or past weather situations by specific variables. Those variables are collected by different sources like satellite images, sky images or sensors. The prediction of the weather can then be conducted by two different ways, by time series techniques or with machine learning. Afterwards the PV power outcome can be predicted with the help of the formerly created weather forecast. With those two steps it is possible to create i.e., a day-ahead PV production forecast.

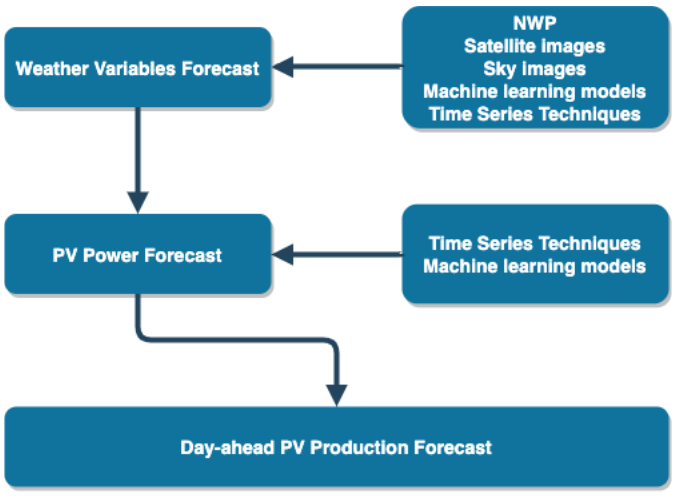


Figure 1 – Division of PV supply prediction

2.2 Time Series and Machine Learning Techniques

This can either be done by deterministic or probabilistic predictions or even by more advanced time series methods like *Autoregressive integrated moving average* (ARIMA) [@Eur17]. All of those mentioned methods to predict time series data do not use machine learning for the computation of the predictions. Because those models need detailed information about the PV systems and also about the weather to predict it precisely, they are not suitable for every use case. More in detail the input data of those models could be *numerical weather prediction* (NWP) [W95].

For the forecast of the weather the period in which the prediction takes place is also an important question. In general, there are two kinds of periods, the short-time and long-time predictions. The short-time prediction only includes a short period, like a few seconds to an hour. For the long-time prediction, the period could increase up to a month, which may influence the accuracy of the forecast.

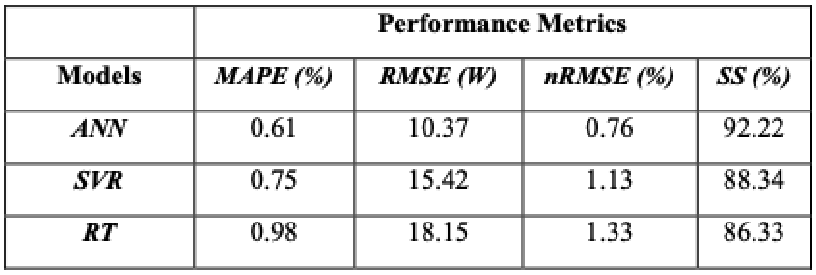
In [R09] short-time weather predictions like ARIMA were tested against machine learning models. According to those examinations the normal time series method outperformed the machine learning ones. The paper states, that the reason behind it could be that those time series methods are better in capturing the transitions in irradiance of the sun over a 24-hour period. Furthermore, it is unlikely that the weather will change dramatically over a short period of time. And because of that, ARIMA perform really good under those conditions.

But due to the fact, that the data received from the sensors and the PV stations may contain missing or corrupted data, the paper [TK18, MS18] states, that there is some other methodology needed to comprehend those flaws. Because of those results and the previous mentioned possible problems with the raw data, the usage of machine learning is appropriate. Furthermore, the paper [TK18] states, that machine learning models are gaining even more attention in the field of time series data. That is because of the ability of machine learning methods to predict relationships between the inputs and outputs without knowing the physical parameters. The paper [TK18] also made some experiments with well-known machine learning techniques which are predicting the output of PV systems:

- Artificial Neural Network (ANN), especially a Feed Forward Neural Network (FFNN)
- Support Vector Machines (SVM)
- Regression Trees (RT)

Those techniques just had the information about the output of the PV Systems, without any information about the NWP. As Table 1 states, the FFNN (in the Table 1 presented as ANN) performed the best, with the lowest MAPE and nRMSE score. The other two methodologies performed similarly.

Table 1 – Performance Overview [TK18]



	Performance Metrics			
Models	***MAPE (%)***	***RMSE (W)***	***nRMSE (%)***	***SS (%)***
ANN	0.61	10.37	0.76	92.22
SVR	0.75	15.42	1.13	88.34
RT	0.98	18.15	1.33	86.33

The Study [PC12] did the same comparison with five different forecasting techniques, namely the Persistent model, ARIMA, kNNs, ANNs, and ANNs optimized with Genetic Algorithms (GAs/ANN). For those tests the machine learning tests were more successful than the time series models, because they need more information about the physical parameters like NWP. But it should be mentioned that the testing of those models was only used for the prediction of the power output of PV Systems and not for the weather forecast. [PC12] concludes the same observation about the better performance of machine learning techniques on local areas rather than different locations as [KP11].
In other research like [AB17, DA16] techniques like the *Principal Component Analysis* (PCA) and feature engineering with RT were used. In addition to that the NWP data were smoothed with various approaches and the authors used a grid of NWP data around the PV System and took the spatial average of those collected data. One of the main conclusions of the experiments were, that aspects as the feature engineering and also the usage of PCA con-

cludes to better results for the weather forecast. In [DA16] PCA was combines with an ANN and *Analog Ensemble* (AnEn) to predict solar irradiance. In this experiment the PCA was also used for the feature selection. Furthermore, it was compared, if the model would give better results without the usage of PCA within the model. But the results prove, that they give better outcomes with it.

For long-term predictions [CD11] used an ANN as the prediction method and NWP as an input data. The model was sensitive to prediction errors within the NWP data and also showed deterioration while forecasting on rainy days. During cloudy or sunny days, the model produced results with MAPE at around 8%. [SL12] on the other site proposed a day-ahead weather classification, which is using SVM to forecast the PV system power output on a 15-minute interval. For that, it divides the weather information in different classes, like: clear sky, cloudy day, foggy day and rainy day. The reason behind the classification is the analysis of local weather forecast and the PV system energy outcome. For each categorization a separate SVM model was created. This experiment shows how to use SVM model for training models on specific climatic conditions.

2.3 Feature Preparation

Most machine learning algorithms need a specific way of data inputs for the training and also later for the prediction. Those are called *structured* or *tabled data*. Thus, it is not possible to use this raw data for machine learning. Another citation for using unqualified data for the training of machine learning is *garbage in*, *garbage out*. Which means, that if the machine learning model is fed with unstructured or not useful datasets it also produces a model which does not fit for the prediction at the end. [BRO20]

Furthermore, the raw data may also include outliers within the datasets. Outliers are data-points within the raw data, which are not representative for the structure of the other data-points within the data. This behaviour is represented in Figure 2. To detect those outliers and treat them correctly there are several techniques available. [CC10]

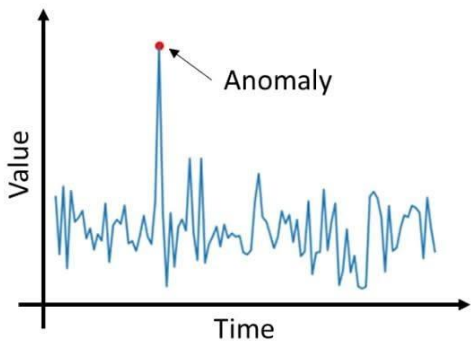


Figure 2 – Presentation of outliers within raw datasets [@Cou20]

In general, it is not possible to use a general framework to detect and treat those outliers for every scenario, because every observation contains different underlaying datasets which has to be treated differently. For example, in the use-case of using the observations of blood-pressure sometimes the outliers can tell the machine learning algorithm, that something is wrong with the blood sample [CC10]. In the paper [TSB07, BRO20] there is a comparison between raw-data which is prepared which outliers detection and just using the raw-data. It is shown that the adaption of the outliers to the corresponding neighbours have a positive impact on the machine learning algorithm which is used afterwards.
After the cleaning of outliers within the dataset the data may seem well-fitted enough for the task, but there may also be some noise within the data which can lead to not useful prediction results in the end.

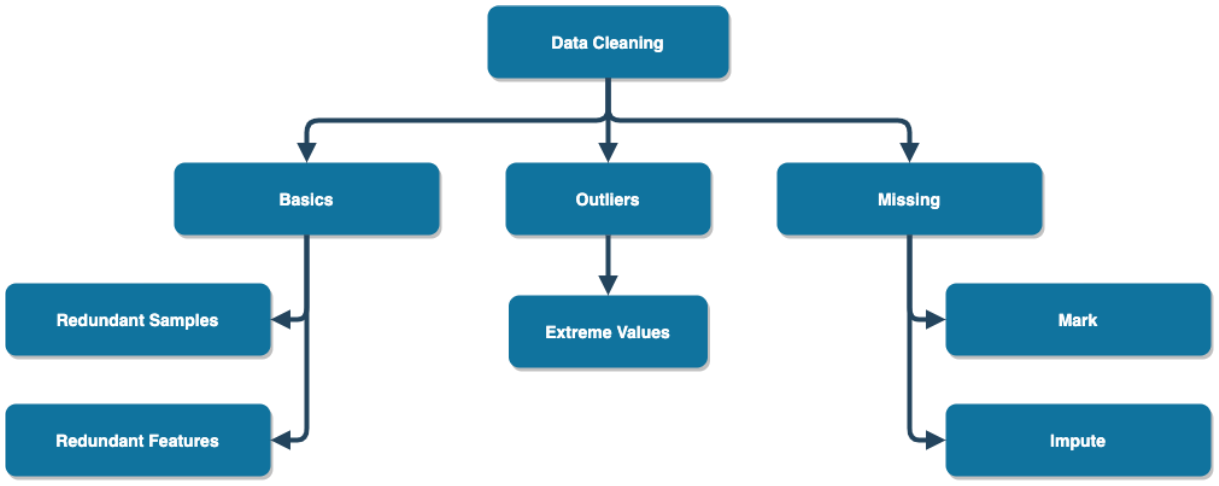


Figure 3 – Data Cleaning Methodologies

To prevent this behaviour of the machine learning model, it is necessary to use one of the, in Figure 3 presented, techniques. Beforehand the handling of outliers was already discussed. But maybe there are also missing values of some kind in the dataset. This can happen if i.e., the sensor, which accumulates the data, had a power loss or failed completely for some time. For this there are two options to treat those missing values. The first one is the deletion of the whole period within the dataset, which will decrease the number of observations within the dataset and could lead to a worse machine learning algorithm in the end. The other solution would be to simply take the average of the neighbour numbers of the observation and take those values as the missing number. This technique can only be applied, if the number of sequentially missing values are not that great. [BRO20]
Sometimes the observations are not in the desired data format. Figure 4 describes that there are two examples of data types which are desired for the usage within machine learning model. For the usage of i.e., regression type ML-Algorithms numerical figures are necessary.

Those can be represented as an integer or float value. In comparison there can also be categorical ML-Model which can be divided in nominal, ordinal or Boolean values. The nominal and ordinal values describe some kind of label for the end-state of the ML-Model. [BRO20]

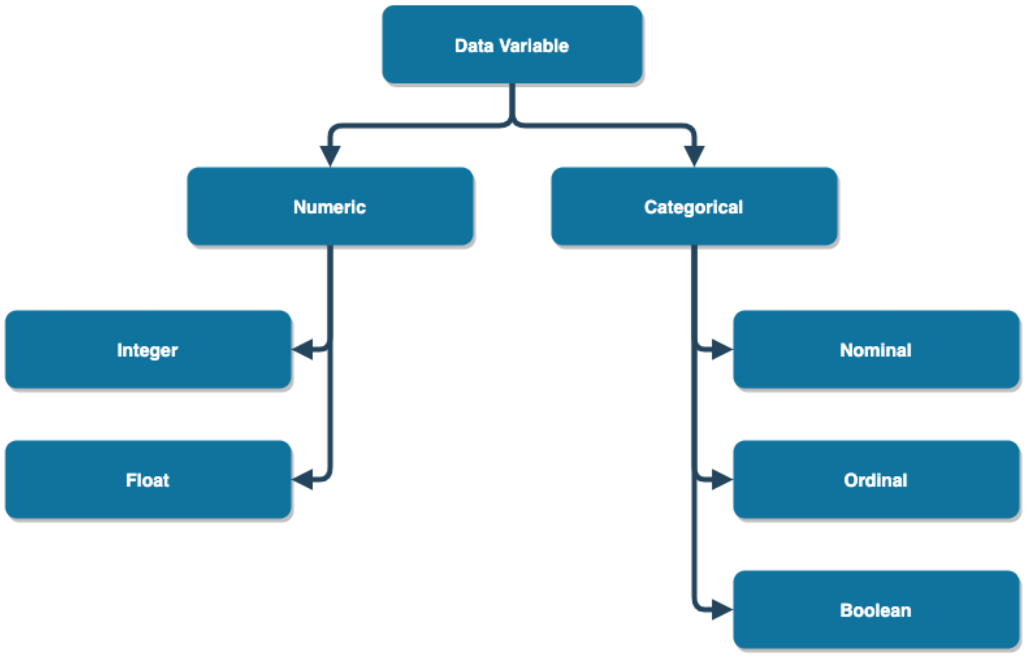


Figure 4 – Data preparation, data transformation

2.4 Feature Extraction

As states in the previous sub-chapter and also in [AB17] the usage of different NWP data as features helps the prediction of the PV system energy output significantly. For this reason, it is necessary find the relevant ones and sort out redundant information within them. Furthermore, it is necessary to normalize those data to stabilize them.

Table 2 – NWP from Meteomatics [IC18]

Variable name	NWP	Unit
total_cloud_cover : p	Total cloud cover	%
wind_speed_10m : ms	Wind speed at 10m	m/s
wind_dir_10m : d	Wind speed direction at 10m	m/s
t_2m : C	Temperature at 2 meters	C
clear_sky_rad : W	Clear sky radiation	W/m^2
clear_sky_energy_1h : J	Accumulated clear sky energy of 1h	J
sfc_pressure : hPa	Surface pressure	hPa
diffuse_rad : W	Instantaneous flux of diffuse radiation	W/m^2
global_rad : W	Instantaneous flux of global radiation	W/m^2
direct_rad : W	Instantaneous flux of direct radiation	W/m^2
effective_cloud_cover : p	Effective cloud cover	%
high_cloud_cover : p	High cloud cover	%
medium_cloud_cover : p	Medium cloud cover	%
low_cloud_cover : p	Low cloud cover	%
precip_1h : mm	Accumulated precipitation of 1h	mm
fresh_snow_1h : cm	Accumulated fresh snow of 1h	cm
global_rad_1h : J	Accumulated energy of global radiation of 1h	J
direct_rad_1h : J	Accumulated energy of direct radiation of 1h	J
diffuse_rad_1h : J	Accumulated energy of diffuse radiation of 1h	J
cape : Jkg	Convective available potential energy	J/kg of air
relative_humidity_2m : p	Relative humidity at 2m	%

Table 2 shows some example data from a weather station nearby some PV systems. As stated in [IC18, AB17] it is useful to eliminate the information which are not useful for this task. Here the *Gradient Boosting Regression Trees* (GBRT) are used to search for variables which do affect the cost function mostly and rank them on that behave. This methodology is proven to be effective by different sources like [PC09]. There are also some other techniques to choose the most effective features like *Relief feature selection* (ReliefF) and *Correlation feature selection* (CFS) [KC19].

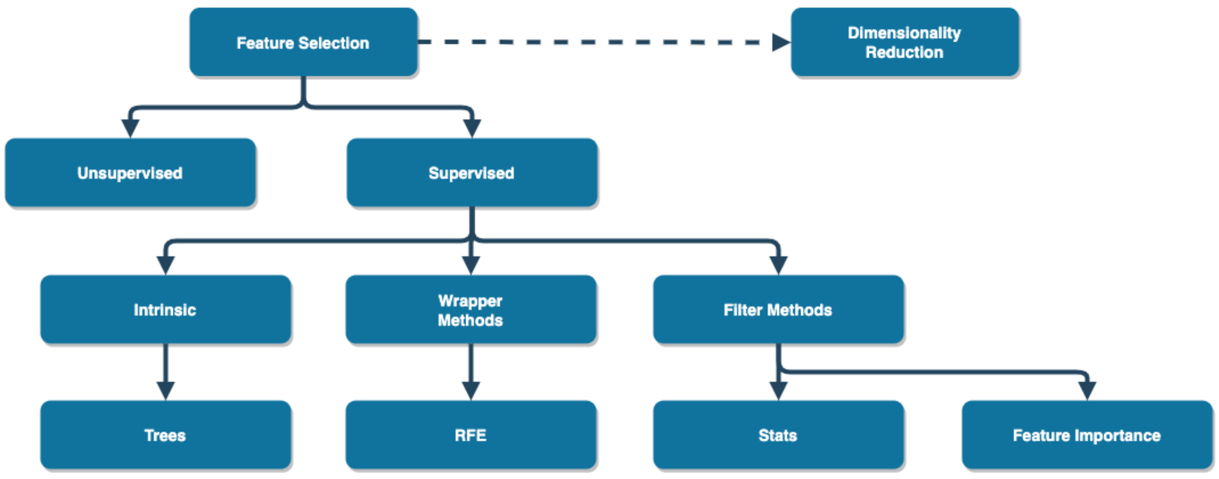


Figure 5 – Feature selection methods [Bro20]

Figure 5 also presents a distinction between the different methodologies of feature selection. The first one is the usage Trees, which is also used in the paper [IC18]. This method uses the intrinsic information of the features to present correlations between them. Furthermore, *wrapper methods* are introduced in the graph. One of the techniques is called *recursive feature elimination* (RFE). [BRO20]

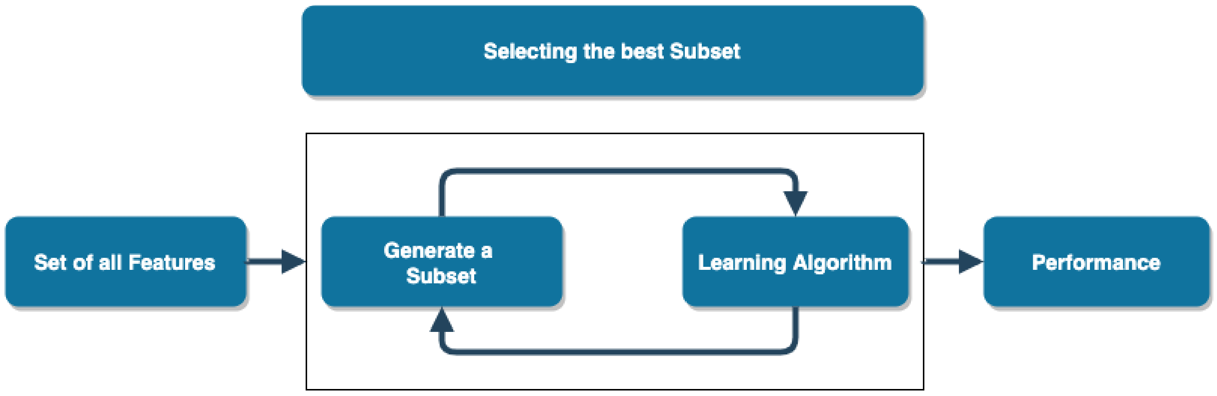


Figure 6 – Methodology of wrapper selection methods

Figure 6 presents how the wrapper selection for a set of features work. In the first step all available features are taken and will be split up into a train and test dataset. Afterwards those selected features will be used in an actual learning algorithm. After the measurement of the performance of the subset of features a new subset of the features will be randomly generated and evaluated. After all possible combinations of the features are used, the ones with the best performance at the end will be chosen. [@KAU16]
There are different techniques to use those wrapper selection methods:

1. Forward selection
2. Backward elimination
3. Recursive feature elimination

The forward selection starts with only one feature (the most important one) in the beginning and keeps adding more features iteratively, until the performance does not improve anymore. In comparison to that, the backward elimination does the complete opposite. It starts with all features in the subset und removes one after another until the performance does not increase anymore. In addition to that the RFE creates a new ML-Model after each iteration and measures the performance. Afterwards the worst feature gets eliminated. [@KAU16]

2.5 Conclusion

One of the important conclusions which could be taken from almost every study is the importance of using NWP data as inputs for the different model to increase the accuracy. In Addition to the importance of NWP data is the sensitivity to NWP data errors. One way to overcome those errors is to collect the data from several sources. Furthermore, the usage of data preparation is presented, which cleans the data and also handles the occurrence of possible outliers and noise within the dataset

Also, it is important to only choose the most important weather attributes for the prediction. For this reason, several techniques were presented to identify those attributes.

Furthermore, there are many different machine learning techniques presented in those studies, especially the optimizations of the different algorithms have a positive impact on those ANNs. But in the scope of this thesis there is going to be a general overview of different machine learning techniques, as stated in chapter one.

3 Methodology

The scope of the thesis requires to predict the forecast of the PV system energy outcome with past energy measurements, which are augmented by observational weather-related parameters. For this several data pre-processing and ML-Model methodologies were used. In this section of the thesis all the used techniques are explained in detail.

3.1 Structure of the Data

For the evaluation of the interpolated and precise weather information two different kind of datasets were created. The first one describes the Glava dataset and it consists of several weather parameters, as well as information about the energy production of the PV system. In this dataset the weather station is co-located with the PV station and contains fine granular samples. The second dataset consists of interpolated weather parameters. One part of those information will be gathered with MeteoStat (https://meteostat.net/en) and the global radiation with SMHI (https://www.smhi.se/). The separation of the radiation and the other weather information is due to the fact, that the radiation is not available at MeteoStat. The interpolated radiation information from SMHI has an error of accuracy of 30 percent for the global radiation and 60 percent for the direct radiation. For the interpolated weather information of MeteoStat, there were no official information about the accuracy loss.
For the structure of the data, it consists of six years of observations for the month July in Glava, Sweden. That information consists of hourly measurement points and have the following features for the weather:

Table 3 – Weather information

Variable name	Unit
Temperature	C
Wind Direction	Gradient
Wind Speed	m/s
Humidity	%
Precipitation	L/m²
Barometric Pressure	mBar
Global Radiation	W/m²
40 Degrees Radiation	W/m²
30 Degrees Radiation	W/m²
Indirect Radiation	W/m²

Table 3 represents the available weather information from the corresponding weather stations. In total there are ten different parameters, which can be used as an input for the prediction. The only difference between the two datasets is, that the three features *40 Degrees Radiation, 30 Degrees Radiation* and the *Indirect Radiation* from the Glava dataset are not available from the MeteoStat or SMHI dataset. For the data of the PV systems there are only three different measurements, which are relevant for the prediction:

Table 4 – PV System information

Variable name	Unit
Stot	kVA
Qtot	kVAr
Ptot	kW

Ptot describes the amount of power, which is usable for the end-user of the system. This energy can be directly converted to practical energy for i.e., energy outlets. Qtot describes the power, which is only available for the system itself and is used to keep the system itself usable. Stot describes the total power in the system itself. This includes the Qtot and Ptot and can be calculated as followed:

$$Stot = \sqrt{\left(Qtot^2 + Ptot^2\right)} \tag{1}$$

3.2 Data Preparation

After the explanation of the different parameters of the weather and PV energy observations, this sub-chapter is going to explain different techniques on how to pre-process the data for the machine learning algorithms to be developed.

3.2.1 Outliers

As explained in chapter 2.3, outliers are values within the raw data which are not in the normal deviation of the i.e., sensory data. For the handling of such data points there are several techniques like the *interquartile range* (IQR) or *quantile-based censoring* (winsorization) available. The interquartile range describes the *body* of a dataset. This means, that the dataset is split up into three different quartiles. First the median of the dataset has to be found. Afterwards the first quartile (Q1) will be calculated the same way as the median, but with the dataset cut in half and only the lower part is considered. That means, that the Q1 now holds the 25 percent below the median. In the same way the upper quartile can be calculated (Q3). But now only the upper half of the dataset is considered. Now the interquartile range can be computed by the following formula, which is stated at [BA09, p. 123]:

$$IQR = Q3 - Q1 \tag{2}$$

Figure 7 illustrates this process using an example. First the general median is calculated, which is 71. Afterwards the median of the lower half and the upper half is going to be calculated. In this case it is 64 for the lower boundary and 77 for the upper one. Thus, the interquartile range is 13. [@Sla16]

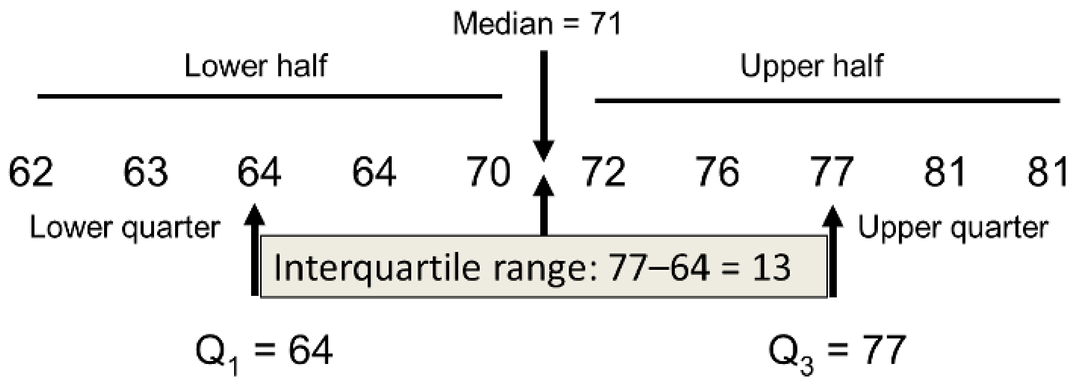


Figure 7 – Interquartile Range [@Sla16]

Most statistical tests assume, that the data is normal "*Gaussian*" distributed. This implies, that the majority of outlier data must lay far away from the majority of the other datapoints. To find actual outliers within the data points, an additional range to the Q1 and Q3 quartiles has to be added. This range is described by the value *k* in (3). As a rule of thumb, the value, represented by *k* is set to 1.5. Afterwards this value is added to the third quartile to find any outlier greater than this and subtract it from the first quartile to find any value lower than this.

$$\left[Q1 - k(Q3 - Q1), \left(Q3 + k(Q3 - Q1)\right)\right] \tag{3}$$

The next step is the handling with false datapoints. The easiest solution would be to just delete those from the dataset and continue with the left-over datapoints. But sometimes it is not the best solution, especially if there is only a limited amount observation available. For this reason, one approach is to set a median of the i.e., five datapoints before and after the outlier and set this value to the outlier itself. This technique is called *trimmed estimator*.

Another technique to further pre-process the data to get an improved dataset is the *rolling quantile* (RQ). This works similar to the IQR, but now not the whole of the dataset is considered, but only a pre-defined window of datapoints. The window size can be determined to any size the user wants. If for example the data does not consist of sensitive data, which can easily be changed in a matter of a few datapoints, the window should be made as small as possible, otherwise the window can also be a few hundred datapoints.

For the winsorization technique the outliers from the dataset will be set to a specific percentile. If for example a 90 percent winsorization was chosen, the data below the five percentile and the data above 95 percentiles will be chosen as outliers. Figure 8 visualizes how the winsorization works. The top graph represents the original dataset, which contains a few outliers on the positive as well as on the negative axes. After the winsorization was carried out, those values were taken care of and the dataset was cleaned. This graph also points out another characteristic of the winsorization, which is the symmetric property of this technique.

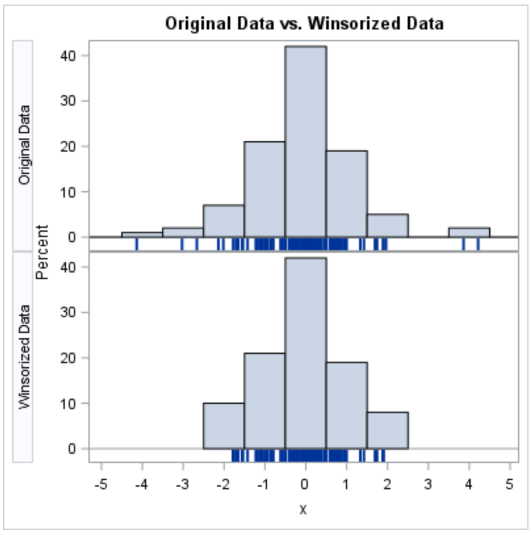


Figure 8 – Winsorization example [WIC17]

3.2.2 Formation of the Dataset

In some cases, some features of the dataset do not contain numeric values and consist of for example Strings. This can be mainly observed in the case of the feature wind direction. This feature is sometimes saved as *North, South, West and East*. The ML Algorithms used in this work are not able to handle those information as it only can comprehend numerical values. For this case that information has to be converted to normal values. With the example of the wind direction, each of the wind direction is saved as a numerical value with the *OrdinalEncoder()* from *Sklearn.* This function arranges each individual string to a unique number.

3.2.3 Normalization

Before the now well processed data can be used for the feature extraction or for the actual machine learning model, there is still a problem with the different ranges of the datapoints. This can be a problem, if for example the temperature is in the range of –10 to 30 Celsius and the barometric pressure has a value range between 0 mBar and 1500 mBar. The range of the barometric pressure includes much higher values than the temperature, for this reason the barometric pressure has a higher valence than the temperature.
To prevent this to happen a normalization or standardization of the whole dataset must be accomplished. The difference between on when to use normalization or standardizing is mainly on the distribution of the dataset. If the dataset is distributed as a Gaussian distribution the standardization is the better choice. [BRO20]
The formula of calculating the standardization [SS17] is described as follows:

$$z = \frac{X - \mu}{\sigma} \tag{4}$$

Equation (4) describes the formula on how the current value which is going to be standardized. μ is the mean value of the whole dataset for this feature and σ is the standard deviation. This procedure is going to be accomplished with every datapoint within the raw data.
For the normalization there are several techniques. For the scope of the master thesis, the *Min-Max Scalar* is going to be used. This normalization technique converts the datapoints to the fixed range between 0 and 1 and is described as follows:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{5}$$

The variables X_{min} and X_{max} in formula (5) describe the upper and lower range, in which the value normalize into.
The previous step of sorting out the outliers is important for the usage of normalization, because this process is sensitive to numbers, which are not in the normal distribution of the dataset. If the previous step for detecting the outliers has not been made, there is also a normalization process called *Robust Scalar* which jointly performs normalization and IQR processing and is more stable to outliers compared to the *Min Max Scalar*.

3.3 Feature Extraction

After the cleaning and pre-processing of the raw data to actual usable data for ML the next step is the feature extraction. Because there are ten weather features in total, it is maybe advisable to clarify which features are the most important ones and which are maybe not that well fitted for this task. For this reason, there are many techniques to figure out which features are the strongest ones and will help the performance of the ML-Algorithm in general.

3.3.1 Pearson Correlation

With the Pearson correlation it is possible to find a monotonic correlation between two variables in a dataset. This relationship between two variables can have one of the following correlations:

- If value A increases, the other observed value B also increases
- If value A increases, the other observed value B decreases

Mathematically the Pearson correlation can be described as following:

$$r = \frac{\sum(x - m_x)(y - m_y)}{\sqrt{\sum(x - m_x)^2 \sum(y - m_y)^2}} \tag{7}$$

$$\begin{aligned} r &= Correlation\ coefficient \\ x_i &= Values\ of\ the\ x - variable\ in\ a\ sample \\ m_x &= Mean\ of\ the\ values\ of\ the\ x - variable \\ y_i &= Values\ of\ the\ y - variable\ in\ a\ sample \end{aligned}$$

$$m_y \ = Mean\ of\ the\ values\ of\ the\ y - variable$$

The result of the Pearson correlation (7) ranges in the area between: $-1 \leq r \leq 1$. If the correlation is in the negative spectrum of the range, the two observed features influence themselves in the opposite direction. The same interpretation can be made for the positive spectrum, but reverse. If the correlation is close to one, the two features have a strong correlation between each other and if r is close to zero, there is no correlation in general.

3.3.2 Spearman Correlation

The Spearman correlation, in comparison to the Pearson correlation assesses monotonic relationships between two ordinal variables. That means, that the Pearson correlation only considers a perfect correlation (+1) between two variables, if the observing variables increase or decrease for the same amount each timestep. On the other side, the Spearman correlation also considers a correlation between two variables perfect, even when they do not increase or decrease in the same amount each timestep. The Spearman correlation coefficient can be described as follows:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{8}$$

$$\begin{aligned} \rho \quad &= Spearman\ correlation\ coefficient \\ d_i \quad &= Difference\ between\ two\ variables \\ n \quad &= Number\ of\ observations \end{aligned}$$

3.3.3 Principal Component Analysis

Another well-researched technique to analyse the importance of features in a dataset of several features is PCA. With this method it is possible to reduce the number of features in a huge dataset without decreasing the accuracy at best. [Tmp02]
The structure of the PCA methodology is divided into six different steps, which are going to be explained in detail now:

1. Standardization

This step is as important as the standardization for the usage of the IQR, because this technique is also sensitive to the variance of the initial variables. The calculation of the standardization is the same as the one previously explained.

2. Covariance Matrix Computation

This matrix consists of the covariance information of each feature to another. That means, that the dimensions of this matrix are (p x p) with p as the number of features in the dataset. This is the same computation as for the Pearson Correlation.

3. Compute the Eigenvectors and Eigenvalues of the covariance matrix

With the help of the calculation of the eigenvectors and eigenvalues of the covariance matrix it is possible to determine the *principal components* (PC) of the data. The principal components are a linear combination of the initial variables. These combinations are so structured, that they are uncorrelated and most of the information of the initial variables are squeezes into the first component. In the case of the weather features there also exist ten PC. The PCA algorithm tries to put as much information in the first component and then the maximum of information in the second one and so on, until it reaches the last one. These steps of putting the left-over information into the next PC is presented in the Figure 9.

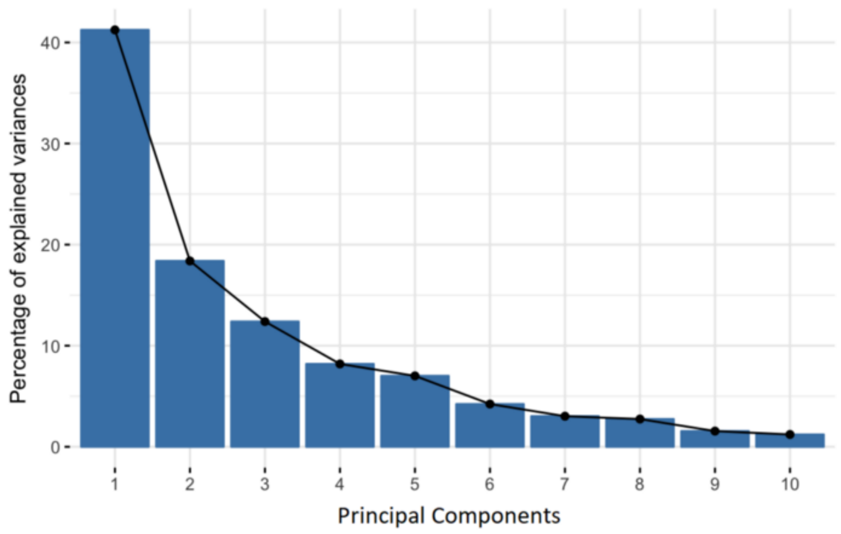


Figure 9 – Graph to present the different variances of the PCAs [Tmp02]

4. Calculation of the PC

For each PC there are as many eigenvectors and eigenvalues as features available. Those eigenvectors describe the directions of the axes where there is the most variance, this axis is called the PC. The eigenvalues, which are corresponding to the eigenvectors present the amount of variance carried by each PC or eigenvector. Afterwards those PCs are going to be ranked by the importance from highest to lowest.

5. Calculating the feature vector

The next step is to decide whether to keep all the calculated PC or to discard some of the irrelevant ones. This is going to be handled with the feature vectors. The feature vector contains columns of the eigenvectors of each PC. The first step is the reduction of the dimensionality. This means, that I.e., out of *n* dimensions of the PC only *p* dimensions are left over.

6. Recast the data along the PC axes

Apart from the standardization there were no changes made by the original dataset. But the input datasets remain in the original axes. Now, to find out, which corresponding feature are present with each PC the original data must be formed to the axes described by the PCs. This step can be made by multiplying the transpose of the original data by the transpose of the feature vector:

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T \tag{9}$$

3.3.4 Random Forest

One of the most popular ML-Algorithms is the random forest. They are well-established for the usage of ML-Tasks but are also used for the task of feature selection. Random Forest is used in many ways, because they are highly accurate, generalize better and are even interpretable for humans. [TMP04] For the random forest a user-selected amount of decision trees is created. For those trees there are several options of trees:

3.3.4.1 ID 3

The ID3 methodology was created in 1986 by Ross Quinlan. This algorithm creates a multiway tree which finds for each node the feature with the highest informational gain for the target. Those trees are grown to their maximum size and then get pruned to improve the ability of the tree to generalise the unseen data.

There are **three** steps for the creation of an ID3-Decision Tree:

1. **Find the feature with the highest amount of information gain**

$$IG(S,A) = H(S) - \sum_{i=1}^{n} \frac{\#(S_i)}{\#(S)} * H(S_i) \tag{9}$$

S	$= Dataset$
A	$= Attributes$
$H(S)$	$= Entropy\ of\ the\ whole\ Dataset$
$\sum_{i=1}^{n} \frac{\#(S_i)}{\#(S)} * H(S_i)$	$= Sum\ of\ the\ weighted\ entropy\ of\ the\ attributes$

For each attribute A the information gain over the whole dataset S is calculated (9). The informational gain is the difference of the entropy $H(S)$ of the whole dataset, which is represented by the formula (10) and the individual entropy weighted entropy $H(S_i)$ of the observed attribute, which is shown in formula (11).

$$H(S) = \sum_{x \in X} -p(x) log_2 p(x) \tag{10}$$

S	$= The\ whole\ Dataset$
X	$= The\ set\ of\ features\ in\ S$
$p(x)$	$= The\ number\ of\ elements\ in$ $feature\ x\ to\ the\ number\ of\ elements\ in\ S$

$$IG(S,A) = H(S) - \sum_{t \in T} p(t)H(t) \tag{11}$$

$H(S)$	$= Entropy\ of\ the\ whole\ Dataset$
T	$= Subsets\ created\ by\ splitting\ S\ by\ attibute\ A$
$p(t)$	$= Propotion\ of\ t\ to\ the\ number\ of\ elements\ in\ S$
$H(T)$	$= Entropy\ of\ subset\ t$

2. **Creation of the root node**
   The attribute with the highest informational gain is the attribute chosen for the root node.

3. **Recursive calculation for new root nodes for the sub-branches**

3.3.4.2 ID 4.5

This tree is the successor of the ID3 algorithm. The improvements of the ID4.5 decision tree algorithm are:

- It is possible to handle continuous and discrete attributes
- The ID4.5 algorithm can also handle training data with missing attribute values
- Pruning trees after the creation

3.3.4.3 Classification and Regression Trees

Decision Trees are one of the most commonly used, practical approaches for the task of supervised learning. It can be used to solve both, regression and classification tasks. Within the trees the class labels are represented by the leaves and the branches denote the conjunctions of features leading to those class labels. The regression tree is used, when the prediction outcome is a real number, and the classification trees are used to predict the class to which the data belongs to. These two categories are collectively called *CART*.
The *Gini Index* or *Gini Impurity* is calculated by subtracting the sum of the squared probabilities of each class from one. It favours mostly the larger partitions and are very simple to implement. The *Gini Index* ranges between zero and one, where zero represents purity of the classification and one denotes random distribution of the elements among various classes. The middle shows that there is an equal distribution of elements across some classes.

As a metric or cost function for the regression tree the *least square* is used. Because the *CART* decision tree is used as a regression tree in this thesis, only the least square cost function is going to be explained:

$$y = a + bx \tag{12}$$

With this function it is possible to calculate the corresponding y-value with the knowledge of the x-value. But for this function, it is necessary to previously calculate the a and b value. The following calculations are used to describe those figures.

$$a = y - bx \tag{13}$$

$$b = \frac{n \sum xy - (\sum x)(\sum y)}{n \sum x^2 - (\sum x)^2} \tag{14}$$

This described cost function is the equivalent to the informational gain of the previous shown ID3 and ID4.5 decision trees. For the usage of random forest, this methodology is the most used one.

3.3.5 Wrapper Feature Selection

All those feature selection methods, which were mentioned and explained earlier were filter methods to classify the features with the most informational input to boost the performance and accuracy of the ML-Algorithm afterwards. Another approach is the usage of wrappers. [TMP05]
There are four different types of wrapper algorithms, which are also explain in the up-coming sections.

3.3.5.1 Sequential Forward Selection

The SFS starts with a subset of zero features and increases the number of features for each iteration. The feature which maximises the criterion function is going to be selected each iteration until desired number of features is reached.

Input values: $Y = \{y_1, y_2, y_{3,\dots\dots,}y_d\}$
Output values: $X_k = \{x_j \mid j = 1,2, \dots, k \colon x_j \epsilon\ Y\}, where\ k = (0,1,2,3, \dots, d)$
The algorithm takes the whole d-dimensional feature set as an input and returns a subset of features k, where $k < d$.

Initialization: $X_0 = \emptyset, k = 0$

The algorithm is initialized with an empty feature set.

Recursive steps (inclusion):

$x^+ \quad = \arg\max J(X_k + x)\,, \;\; where\; x \in Y - X_k$

$X_{k+1} = X_k + x^+$

$k \quad = k + 1$

In this recursive step each iteration the additional feature x^+ is added to the feature subset X_k. x^+ is the feature which maximizes the criteria function. This recursive step is iterated, until the amount of desirable feature is reached ($k = p$).

3.3.5.2 Sequential Backwards Selection

The SBS starts with a subset of all available features and decreases the amount each iteration. The feature which maximises the criterion function is going to be selected each iteration until the amount corresponds to the desired amount.

Input values: $Y = \{y_1, y_2, y_{3,\ldots\ldots,}y_d\}$

Output values: $X_k = \{x_j \mid j = 1,2,\ldots.,k\colon x_j \epsilon\, Y\}\,, where\; k = (0,1,2,3,\ldots.,d)$

The algorithm takes the whole d-dimensional feature set as an input and returns a subset of features k, where $k < d$.

Initialization: $X_0 = Y, k = d$

The algorithm is going to be initialized with an empty feature set.

Recursive steps (exclusion):

$x^- \quad = \arg\max J(X_k - x)\,, \;\; where\; x \in Y - X_k$

$X_{k-1} = X_k - x^-$

$k \quad = k + 1$

In this recursive step each iteration the feature x^- is removed to the feature subset X_k. x^- is the feature which maximizes the criteria function. In this case the criteria function describes the feature with the least informational input. This recursive step is iterated, until the amount of previously selected features is reached ($k = p$).

3.3.5.3 Exhaustive Feature Selection

The EFS starts with an amount of possible feature subsets and finds the optimal feature subset at the end of all possibilities. The feature subset, which maximises the criterion function is selected at each iteration until every possibility was tested.

Input values: Set of features X , size of feature set n , size of target feature subset d , set of possible feature subsets F of X where each subset is the size of d.
Output values: Optimum feature subset Y_{opt} of size d.
Initialization: $Y_{opt} = \emptyset$
$G_{opt} = -\infty$

The algorithm is initialized with an empty optimal feature subset and a minus infinite place-holder optimal feature subset.

Recursive steps:

```
For all $Y_i \epsilon F = \{Y_0, Y_1, \dots \dots, Y_k\} \mid k = \binom{n}{d}$ do
    $G_i = J(Y_i)$
    If $G_i > G_{opt}$ then
        $Y_{opt} = Y_i$
        $G_{opt} = G_i$
    End if
End for
```

In this recursive step each iteration evaluates with the criteria function whether the current combination of features performs better than the former G_{opt}, the new subset is the optimal one. This is repeated, until all combinations are used and the optimal one is chosen.

3.3.5.4 Bi-Directional Elimination

The Bi-Directional Elimination (Stepwise Selection) is similar to the SFS, but in each iteration while adding a new feature, it also checks the significance of already added features in the subset. If one of the already chosen features are not significant enough, it removes this feature from the subset via SBS. So, this algorithm is a combination of both previous explained wrapper algorithms. [@Gct18]

Input values: $Y = \{y_1, y_2, y_{3,\dots\dots,y_d}\}$

Output values: $X_k = \{x_j \mid j = 1,2, \ldots., k: x_j \epsilon\, Y\}, where\ k = (0,1,2,3, \ldots., d)$

The algorithm takes the whole d-dimensional feature set as an input and returns a subset of features k, where $k < d$.

Initialization: $X_0 = \emptyset, k = 0$

The algorithm is initialized with an empty feature set.

Recursive steps (Bidirectional):

Perform the next step of SFS to select the best feature of the dataset

$$x^+ \quad = \arg\max J(X_k + x), \ \ where\ x \epsilon\ Y - X_k$$
$$X_{k+1} = X_k + x^+$$

Perform SBS on the new selected features and remove the ones with the poorest performance from the dataset.

$$x^- \quad = \arg\max J(X_k - x), \ \ where\ x \epsilon\ Y - X_k$$
$$X_{k-1} = X_k - x^-$$

There are some limitations to the algorithm to guarantee, that SFS and SBS are converging to the same solution:

- Features, which are already selected by SFS are not removed by SBS.
- Features, which are already removed by SBS are not added by SFS anymore.

3.4 Machine Learning Model

As stated in chapter 2.2 there are many different techniques to predict and handle time series data in general. Mostly they are divided into two main categories: statistical or machine learning algorithms. For the scope of this master thesis only the machine learning algorithms were used. In detail the following algorithms in particular:

- LSTM (Long Short-Term Memory)
- Facebook Prophet
- Random Forest

In this chapter the LSTM and also the Facebook Prophet algorithms are going to be explained. The Random Forest was already explained in chapter 3.3.3 and because of that, its omitted in this chapter.

3.4.1 LSTM

In comparison to normal *MLP* (Multilayer Perceptron), which consists of many layers with neurons in it and the input data is propagated through the network itself, the LSTM has recurrent connections. This means, that the state of the previous activations is also used as a context for the output. But in comparison to normal RNN the design of the LSTM Network allows to overcome the problem of the *vanishing* or *exploding gradients*. This means, that the weight update procedure changes the weights so fast in one direction or the other, that it is graduate to zero or infinity. Those phenomena make the neural network useless for longer sequences. [BRO17a, GLF09]
In general, RNNs are good for the processing of sequential data and for the prediction of those. But those networks suffer from short-term memory. To overcome this obstacle, LSTM networks were created, which uses gates to migrate short-term memory to those algorithms. Primarily gates are neural networks, which regulate the flow of information through a sequence chain.

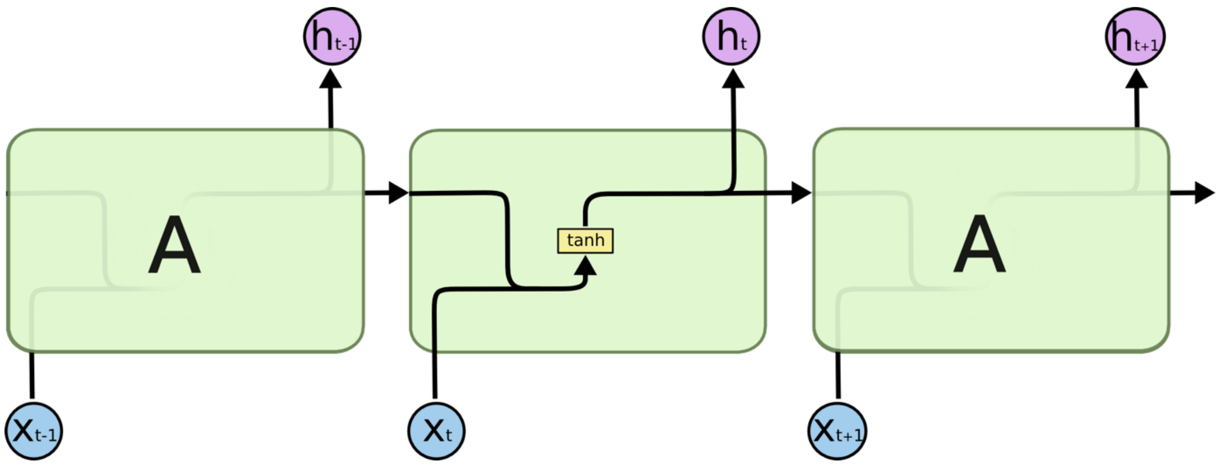


Figure 10 – Repeating Module in normal RNN [@Col15]

The previously mentioned problem of the normal RNN is called the *long-term dependency problem*. To overcome this problem, Hochreiter and Schmidhuber introduced the architecture 1997 [HS97] and were refined during the years by many people afterwards. Figure 10 illustrates the architecture of a single-layer RNN. The single layer consists of a chain of repeating modules and contains a *tanh* activation function within them. With this layer it is possible to squish the incoming values in the range of -1 and 1.

In comparison to that, a LSTM Network also contains a chain structure, but instead of a single *tanh*, the repeating module consists of three different gates in general. This structure can be seen in Figure 11.

In this diagram the yellow boxes represent the different NN-Layer. The red dots are pointwise operations, like vector addition or multiplication. The arrows represent the copying of the whole vector from one state to the next one.

The main features of the LSTM Networks are the so-called *gates* and *cell states*. With the gates the flow of information can be regulated. With those operations the network can decide to keep information or forget them. The cell states act as a transport highway. Those carry the information from one layer to the next one. In combination of those two features the networks learns in each layer, which information are relevant or not.

The gates consist of *sigmoid activations*. Those acts similar to the tanh activation of the normal RNNs. Except, that in the case of the sigmoid activation, the values are going to be squished in a range of 0 to 1. With this it is easier to decide which information can be forgotten. Because any value, which is multiplied by 0 results also in a 0. In the following, the three different gates of a LSTM Network are going to be explained.

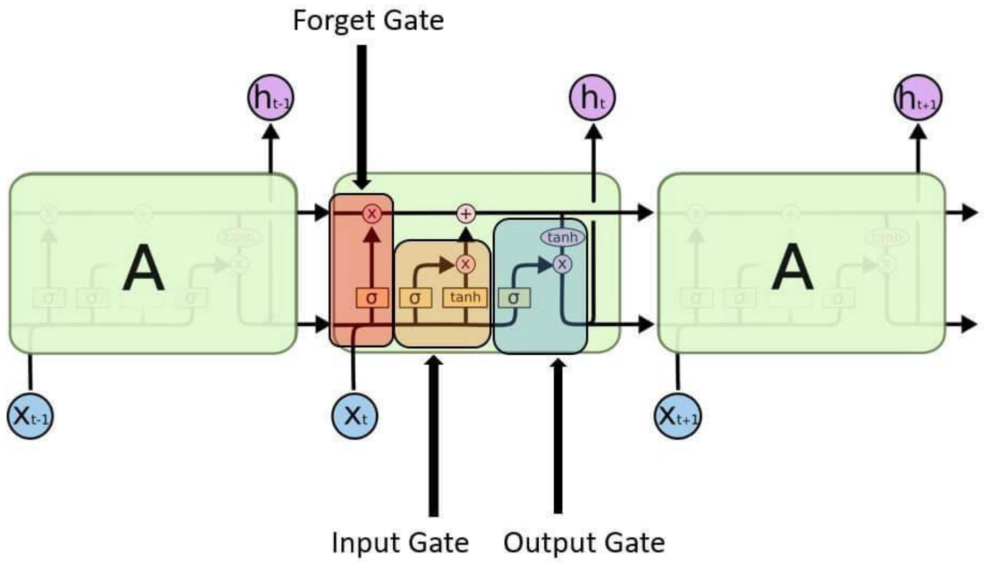


Figure 11 – LSTM Architecture [@Col15]

Forget Gate

With the help of this gate (represented as a red box in Figure 11), the network decides, what information is important or can be forgotten. For this, the information of the previous hidden state (bottom arrow from the previous hidden cell h_{t-1}) and the current input (X_t) are merged and the result is passed through the sigmoid function. The outcome of this results in a value between 0 and 1 and is called f_t. Afterwards this number is then multiplied with the previous cell state (C_{t-1}).

Input Gate

The input gate is in charge of updating the cell state. This gate is illustrated as the orange box in the upper figure. Firstly, the previous hidden state (h_{t-1}) and the current input (X_t) are going through the sigmoid function (i_t). Simultaneously those numbers go through the *tanh* activation layer (ς_t). This value is called the candidate. Afterwards the results of those layer are multiplied.

Cell State

Now, after the importance of the current input in combination with the previous hidden layer were calculated, the new cell state is computed. This happens with the following formula:

$$C_t = f_t * C_{t-1} + i_t * \varsigma_t \tag{15}$$

Output Gate

The last gate in the LSTM Network is called the output gate. This gate decides, what the next hidden state should be. This gate is visualized as the blue box in Figure 13. First the previous hidden state (h_{t-1}) and the current input (X_t) are going through another sigmoid activation layer. Simultaneously the new cell state will go through a *tanh* activation layer. The results of both computations are multiplied afterwards. This figure represents the new hidden state (h_t), which is passed to the next step.

3.4.2 Facebook Prophet

"Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical

data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.” [TL17]
In other words, with this open-source framework which Facebook created, it is possible to make a forecast of time series data. It is especially useful for the prediction of data with strong seasonal effects, which also occurs with the weather and energy data from the project.

For the algorithm Facebook Prophet uses a decomposable time series model. This means, that the overall machine learning algorithm contains several smaller ones inside. This methodology was first introduced in the paper [HP90]. In this paper the model contains a breakdown into a trend, seasonal and irregular model. In total the formula for the algorithm is composed of the following components:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \tag{16}$$

In this formula, $g(t)$ describes the trend function. This includes non-periodic changes within the time series data. $s(t)$ implies periodic changes, for example weekly, yearly or seasonally changes within the data. And the third component $h(t)$ contains the effects of holiday seasons throughout the year. The last component ε_t represents the idiosyncratic changes which cannot be retrieved by the other components of the model $y(t)$. For this model it is assumed, that the error ε_t is normally distributed.
In the following, the three main components of the Facebook Prophet model are going to be briefly explained:

Trend Model

In general, two different kinds of model were created to (mainly) fit the most applications Facebook needed. On the one hand a *saturating growth* model and on the other a *piecewise linear* model. The saturating growth model is described as following:

$$g(t) = \frac{C}{1 + \exp\left(-k(t-m)\right)} \tag{17}$$

With C as the carrying capacity, k as the growth rate, and m as an offset parameter. The carrying capacity is i.e., the maximum amount of people who have access to internet to use Facebook. In the case of the PV outcome or weather prediction, this value can be described as the maximum amount of energy which the system can produce in the end. The second model which can be used as a trend model is the piecewise linear model:

$$g(t) = (k + a(t)^T \delta)t + (m + a(t)^T \gamma) \tag{18}$$

This model is beneficial, when the dataset does not show a saturating growth. A piecewise constant rate of growth also creates a useful model for those cases. In this model k also describes the growth rate while δ has the rate adjustments. m is the offset parameter.
Another question for the trend are the so-called *changepoints*. Those points in time describe any underlaying changes in the time series data. For example, a new launch of a phone or any other product can change the growth rate tremendously. For this reason, at those points it is allowed for the growth rate to increase or decrease. [@Ch18, TL17]

Seasonality Model

This part of the main model is the most important for time series data, which is infected by seasonality during a specific range of time. This can be i.e., by vacation times during the year, which are mostly during the summer or holidays. In the use-case of weather information the weather during the four seasons are completely different to each other. In the case of Facebook Prophet, a Fourier series provides a flexible model of periodic effects:

$$s(t) = \sum_{n=1}^{N} (a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right)) \tag{19}$$

The variable P describes the period of time. This can be for example a year $(P = 365.25)$ or a week $(P = 7)$. [@Ch18, TL17]

Holiday Model

This model is relatively complex to describe, because each country has its own holidays. For example, in the United States, there is thanksgiving, which always falls on the fourth Thursday in November. Many other countries have holidays which follow the lunar calendar. For this, it is possible to set a list of holidays which, then can be feed in the machine learning algorithm. With this the future holidays are going to be set on the same day as the past ones. Also, the algorithm knows, that those days have to be treated specially. Otherwise, it is also possible to set a flag for a specific country with pre-defined holiday information given by Facebook.

3.5 Performance Metrics

Performance Metrics calculates an error or accuracy for two or more observing variables. For this reason, those metrics are used to measure the performance of machine learning algorithms, where the actual value is compared with the predicted one. For the scope of this master thesis various performance metrics were used. In the following equations Y represents the actual value and $\hat{Y}$ the predicted one.

Mean Absolute Percentage Error

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{Y_t - \hat{Y}_t}{Y_t}\right| \tag{20}$$

Mean Square Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{21}$$

Root Mean Squared Error

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2}{N}} \tag{22}$$

Mean Absolute Error

$$MAE = \frac{\sum_{i=1}^{N}\left|\hat{Y}_i - Y_i\right|}{N} \tag{23}$$

R^2

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{24}$$

4 Implementation

In this section of the master thesis, the implementation of the different methodologies, which were explained in chapter 3, are going to be described. Furthermore, the development environment is going to be explained. The implementation was carried out with Python 3 version 3.8.5 to be precise. Python was used in this project, because it is a wide-spread programming language in the field of machine learning and inherits many libraries like TensorFlow and Sklearn.

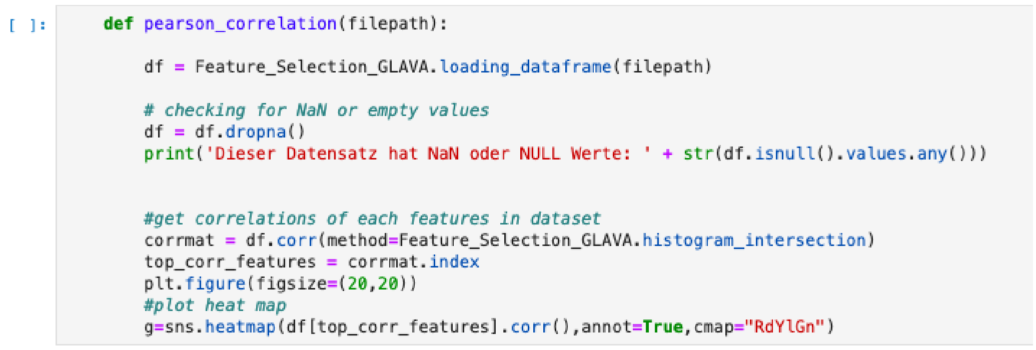
The implementation is divided in five different sections. The first one describes the retrieval of the different datasets, such as the Glava, SMHI and Meteostat. The second section outline the different techniques of the data preparation. After the different datasets are well-prepared for the next step, the methodologies of the feature selection are explained in the third sub-chapter. Now, that the datasets are cleaned and the most important features are selected, those features are used as an input for the different ML-Model, which are explained in the fourth section. The last sub-chapter explains the methods for the evaluation of the different ML-Model and the different use-cases.

4.1 Development Environment

For the development of the project Jupyter notebook was used. This development environment was founded in February of 2015 by Fernando Pérez and Brian Granger. With this environment it is possible to use different interactive data science and scientific computing across all programming languages. Furthermore, Jupyter provides the Jupyter hub, which allows multiple people to work on one server simultaneously and share the resources of the server, like GPUs and CPUs. [@Jup20]

Within the datacentre of the University of Karlstad a new server for the Jupyter hub was created. For each user it contributes enough computing power to accelerate the training of i.e., machine learning model training with a Nvidia RTX 2080.

One of the main advantages of the usage of Jupyter notebook is, that it is accessible through mostly any web browser. Furthermore, each notebook is structured in different small cells which contains the code, which is presented in Figure 12.



```
[ ]:  def pearson_correlation(filepath):

          df = Feature_Selection_GLAVA.loading_dataframe(filepath)

          # checking for NaN or empty values
          df = df.dropna()
          print('Dieser Datensatz hat NaN oder NULL Werte: ' + str(df.isnull().values.any()))


          #get correlations of each features in dataset
          corrmat = df.corr(method=Feature_Selection_GLAVA.histogram_intersection)
          top_corr_features = corrmat.index
          plt.figure(figsize=(20,20))
          #plot heat map
          g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

Figure 12 – Jupyter Notebook Cell

4.2 Data Downloader

In this section the different downloader for the various datasets are explained. In total there are three data sources, which are used: Glava Energy Center, SMHI and Meteostat. For the data of the Glava Energy Center there are two different ways to access the data. The first one is via the virtual desktop and the second one is the web API. In the beginning of the project the web API was not accessible, so the data was retrieved via the virtual desktop and the Metrum Software (see Figure 13). The problem with the received data was, that there were several errors within the downloaded data. The Metrum Software is able to retrieve the correct data and present it, but the exporter simply adds every value from the previous state, which makes the outcome useless. Furthermore, it is only possible to retrieve a maximum of six days with the Metrum Software, because otherwise the output file will be corrupted.

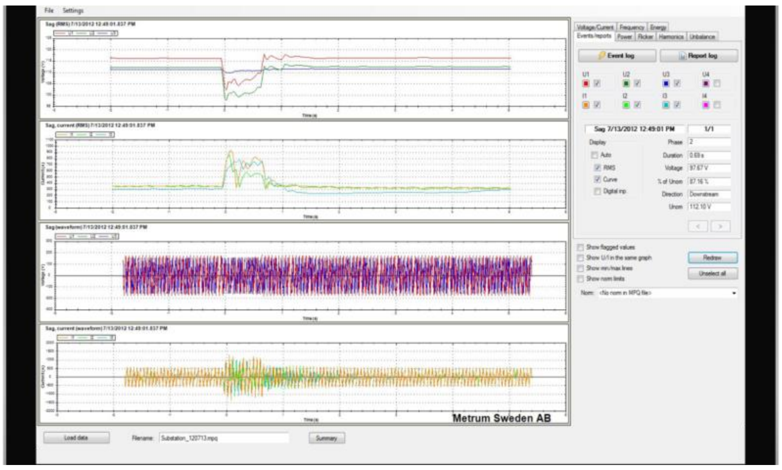


Figure 13 – Metrum Software

With the Web API it is possible to retrieve the necessary data from GLAVA faster and more stable. For example, the download of a whole month was possible within a few minutes. The received data from the Web API will be stored as a JSON file afterwards. The structure of the file is presented in Listing 1.

The JSON file consists of three main attributes. The *mPoint* represents the name of the sensory endpoint. In this case it is the *ABB Inverter*. The next main attribute is the *datatype*. This element contains the information about the type of data within the dataset. Here it represents the information about the *General Inputs*. The last relevant information within this JSON file is the *channels* attribute. In this object the information about the different sensory pairs are saved. It contains the date and the corresponding value for each sensor pair.

```
"mPoint":
   {
    "name": "ABB Inverter",
    "id": 1
   }
}

{
"dataType":
    {
    "name": "General Inputs",
    "id":   3
    }
}
{
"channels":
    {
    "0":
        {
        "name": "GI1"
        "values": [
                  {
                  "value": 197.216,
                  "time": "2015-07-01"
                  },
                  "...."
                  ]
        }
    }
```

Listing 1 – JSON Structure GLAVA

As seen in the Listing 1, there are the two attributes *mPoint* and *datatype*. The Glava Web API has 14 different kinds of attributes, which are represented as the *datatype* and four different input sources, which are labelled as *mPoint*. All the different data inputs and data types are presented in the Figure 14. The four input sources are: ABB Inverter, ELTEK VALERE Inverter, Laddstolpe and the HSB ACES. Each of those sources have most of the 14 attributes, which are also presented in the Figure 15. Most of those attributes are about electronic information from the system and only one is specific for the weather information. This one is called *General Inputs*.

For the scope of this thesis and also for the prediction of the weather and the corresponding PV energy outcome, only a few of those variables are necessary. The two most important attributes are the *Energy* attribute and the *General Inputs* attribute. Those two attributes are also divided in different features. The energy attribute for example consists of three different input values: Ptot, Stot and Qtot. As described in chapter 3.1. the total amount of energy, which is produced by the PV station is saved in the value Stot.

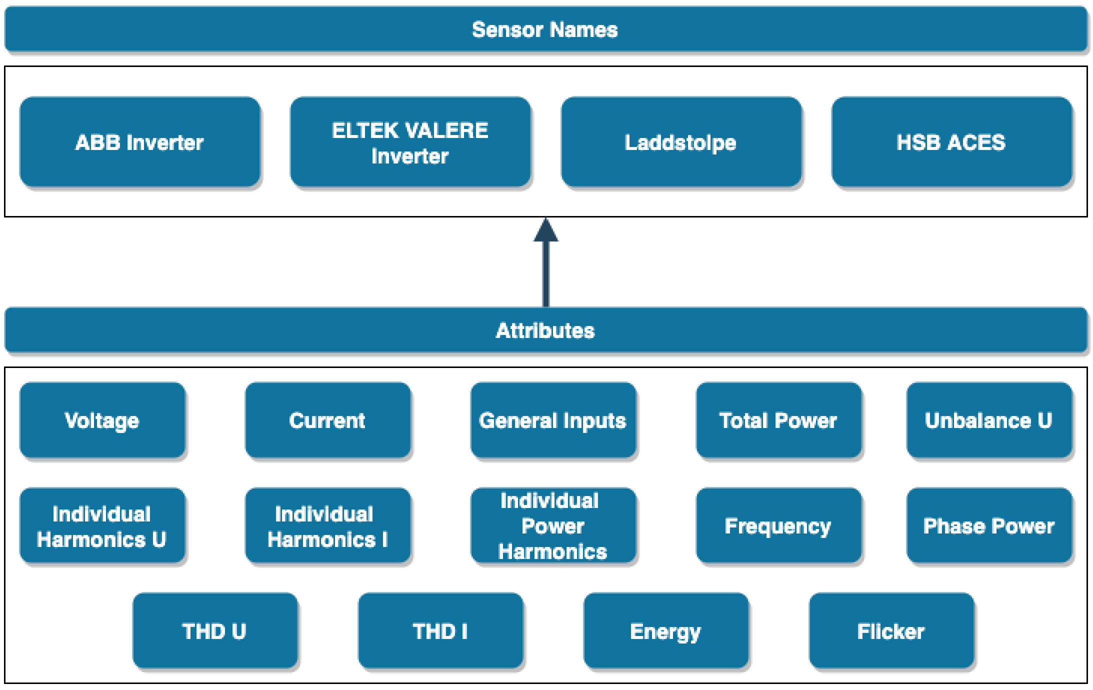


Figure 14 – Glava Sensor Information

The other attribute *General Inputs* inherit the information about the weather. Each of the four sensors has four different weather information in total. For this reason, the weather information has to be gathered separately from those four sensors. In the following Figure 16 the weather information of each sensor is presented. In general, there are ten different weather features, which are present in the database of Glava. The *ABB Inverter* and the *HSB ACES Inverter* have two or more endpoints, which are currently not available or not in use. Because of this, the *HSB ACES* input source is not used in general for retrieving any weather information.

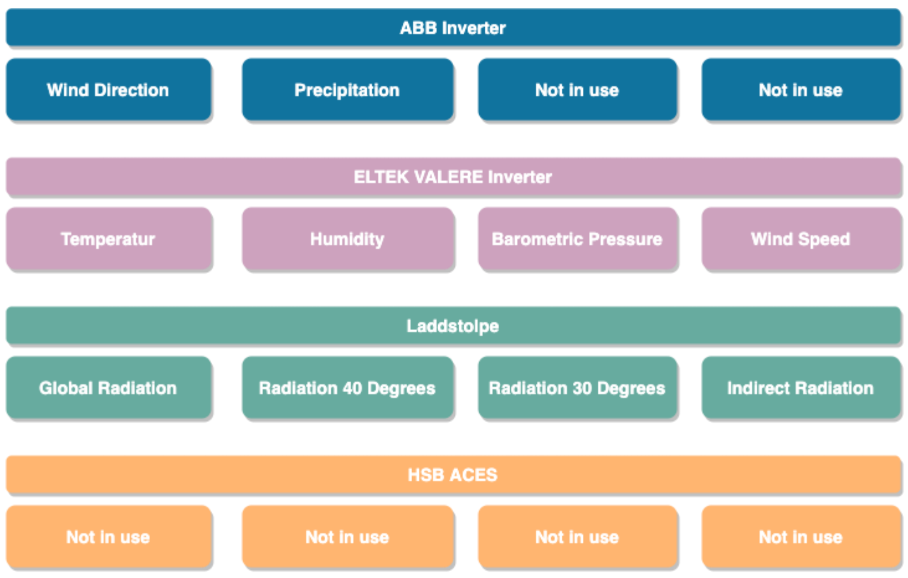


Figure 15 – Glava Weather Features

For the download of the Glava weather and energy data the two libraries *urllib* and *json* were used in general. As seen in Listing 2, the first step is to disable the SSL certificate verification. Afterwards the needed URL for the download is going to be set together. This is done by creating a fixed URL and set the needed parameter in the String. The needed parameters are the *sensorID* and the *typeID*. The user of the system can set that information via variables while calling this function.
Then the URL is going to be opened with the help of the library *urllib* and the JSON is requested. The call back JSON file is then saved within a folder the user selected.

```
# disable ssl certificate verification
ssl._create_default_https_context = ssl._create_unverified_context

# Creating the url
url_fixed ='https://systemet.glavaenergycenter.se:444/MetrumWebapi
           /api/longtimedata?dataTypeId='

final_url = url_fixed + typeID + '&mpointId=' + sensorID + '&from=' +
            begin + '&to=' + end

with urllib.request.urlopen(final_url) as url:
    data = json.loads(url.read().decode())
    print("Successfully Downloaded the Timeseries data from GLAVA")

    with open(filepath + 'GLAVA_Sensor_' + sensorID + '_Type_' + typeID +
              '_From_' + begin + 'To_' + end + '.json','w') as json_file:

       json.dumps(data)
```

Listing 2 – Glava Downloader

For the input source Meteostat there is an official Python library to use. To receive the weather information from the closest weather station, the first step is to find the closest one, which is available in the database of Meteostat.

```
02404      Arvika / Högvalta     SE    S 02404     ESKV      59.6667     12.5833
02418      Karlstad Flygplats    SE    S 02418     ESSQ      59.3667     13.4667
02416      Nolgård               SE    S 02416     ESOK      59.3600     13.4700
```

Listing 3 – Meteostat closest stations

Listing 3 is presenting the three closest stations to the city Glava, where the actual PV stations are located. The second step is to receive historical data from the closest station to Glava, which is Arvika in this case. Listing 4 is showing an example of downloading hourly data from the Meteostat database. The three parameters for the *Hourly*-Function are the *id* of the weather station, the start and end date of the observation. With the function *fetch()* the previously described data will be downloaded and set into a dataframe. Because only weather information for the last five years (2015 - 2020) in July is needed, the downloader will automatically only download the datasets for those years.

```python
from meteostat import Stations, Daily, Hourly

# Get hourly data
data = Hourly(id_m, start, end)
data = data.fetch()
```

Listing 4 – Meteostat, download hourly data

Because the Meteostat database only serves the data for some of the needed weather information, another source for the radiation has to be included. For this reason, the interpolated data of SMHI also has to be acquired. For this dataset there is no python API available and that information had to be received manually from the official SMHI website (http://strang.smhi.se/extraction/index.php). There it is possible to retrieve the global radiation for a specific longitude and latitude during a pre-defined date.

4.3 Data Preparation

In the previous chapter it is explained, how the different data sets are obtained. Naturally there are some issues with those, like outliers, missing data or even the structure of the data is not optimal for the upcoming usage of those. Because of that there are several steps of data preparation and cleaning needed to make the handling easier later on.

4.3.1 Combining Datasets

As stated in the previous chapter 4.2 the Glava datasets for the weather information had to be downloaded from each sensor separately. This is the same for the data of the energy produced (Stot). To use this information comfortably in the next steps of feature extraction and creating the model it is necessary to create one big data set with all the different information together.
This is described in the Listing 5. First each of the loaded Dataframes receive the unique name for each column and afterwards the different Dataframes gets combined on the *axis=1*. Afterwards the different columns of the Dataframe are getting scaled. This is needed, if the user of the system wants to use a different kind of timescale for the machine learning algorithm. The default time is six seconds for each sample of the data and can be either used like this or it can be changed in every possible scale for seconds bigger than six seconds, minutes, hours or respectively days. All of the different features within the dataset are then meaned for the scale which is set.

```
# Rename the different Columns
ABB.columns    = ['Date','Wind
Direction','Precipitation','Unused','Unused']
ELTEK.columns  = ['Date','Temperatur','Humidity','Barometric
Pressure','Wind Speed']
LADD.columns   = ['Date','Global Radiation','Radiation 40
Degrees','Radiation 30 Degrees','Indirect Radiation']
Energy.columns = ['Date','Ptot','Qtot','Stot']


# Combine the different DF to one
frames = [ABB, ELTEK, LADD,Energy]
combined = pd.concat(frames, ignore_index=True, axis=1)

# Set the Scale of the DF
combined = combined.apply(pd.to_numeric, errors = 'coerce')
combined.reset_index()
combined.index = pd.to_datetime(combined.index)

combined = combined.resample(scale).mean()
```

Listing 5 – Combining the Glava Datasets

For the interpolated datasets from Meteostat and SMHI those modifications also have to take place. Almost all weather information can be taken from the Meteostat dataset, except for the radiation.

```
del meteo['Dew Point']
del meteo['Snow']
del meteo['Sunshine']
del meteo['W-Code']
del meteo['Peak Wind Gust']

meteo = meteo[pd.to_datetime(meteo['Date']).dt.month == 7]
        meteo = meteo[(pd.to_datetime(meteo['Date']).dt.year == 2016) |
                      (pd.to_datetime(meteo['Date']).dt.year == 2017) |
                      (pd.to_datetime(meteo['Date']).dt.year == 2018) |
                      (pd.to_datetime(meteo['Date']).dt.year == 2019) |
                      (pd.to_datetime(meteo['Date']).dt.year == 2020)]


meteo['Irradiance'] = smhi['Irradiance'].values
meteo['Stot']       = energy['Stot'].values
```

Listing 6 – Combining Interpolated Datasets

Furthermore, it is necessary to include the energy information from Glava in this dataset as well. As stated in the Listing 6, first the unnecessary columns of the Meteostat dataset are

deleted, because those features are not present in the dataset of Glava and for the comparison it is needed to have the same attributes. Afterwards only the month of July is selected from the dataset, because the Glava dataset only contains these months and the years 2016 until 2020. The year 2015 is not considered, because at that point only a few features of the weather information are available.
The last step of the combination is the inclusion of the *radiation* from the SMHI dataset and the *Stot* of the energy dataset from Glava.

4.3.2 Data Cleaning

After the combination of the different datasets into two main datasets, one for Glava and one for the interpolated data, there are still some obstacles with the data in general. Like in chapter 3.2.2. described, there are still outliers and possibly missing values in the dataset. Furthermore, there could be possible datatypes in the dataset, which are also not numeric.

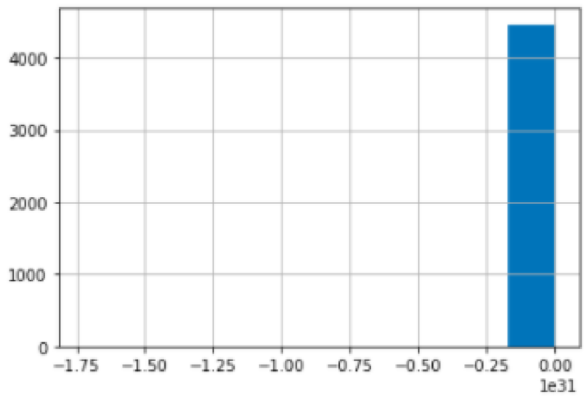
```
lower_p = combined['Temperatur'].quantile(0.01)
higher_p = combined['Temperatur'].quantile(0.99)
combined['Temperatur'] = np.where(combined['Temperatur']
<lower_p,lower_p,combined['Temperatur'])
combined['Temperatur'] = np.where(combined['Temperatur'] >higher_p,
higher_p,combined['Temperatur'])


# accelerating the negativ radiation figures
combined["Global Radiation"]     = np.where(combined["Global Radiation"]
<0,0,combined['Global Radiation'])
combined["Indirect Radiation"]   = np.where(combined["Indirect Radiation"]
<0,0,combined['Indirect Radiation'])
combined["Radiation 30 Degrees"] = np.where(combined["Radiation 30
Degrees"] <0,0,combined['Radiation 30 Degrees'])
combined["Radiation 40 Degrees"] = np.where(combined["Radiation 40
Degrees"] <0,0,combined['Radiation 40 Degrees'])
```

Listing 7 – Correction of Outliers

Listing 7 describes exemplary for the temperature, how the in chapter 3.2.1. described quantile-based censoring was implemented using the *quantile()* function from *NumPy*. First, the quantile range for the data is calculated. Normally the range for the upper and lower quantile is 25 percent each. The outliers within the used dataset are only a few datapoints within a huge scale, for that reason a winsorization of two percent was chosen. After some tests with

adjustments of the quantiles the presented two percent did not cut of datapoints and only selected the outliers. [NG06]
This methodology was used in most of the existing features of both datasets, to make sure that there are no existing outliers anymore. Furthermore, there is an alignment for the radiation features of the Glava dataset in Listing 7. This measurement has to be done, because there are several readings of the sensors, which presented the radiation on the negative level, which is not possible.



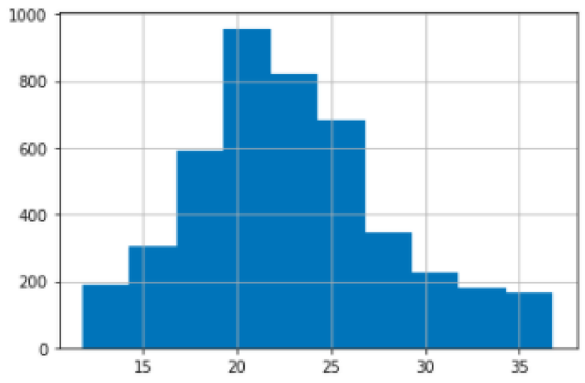


Figure 16 – Winsorization on Temperature (left: cleared, right: without)

As seen on Figure 16, there is a clear advantage of using the winsorization on datasets, which contains a lot of outliers. This example shows that the temperature feature of the Glava dataset contained many outliers. The left side of the Figure 16 shows the temperature feature before the clearing of outliers. This figure clearly shows that now there is a normal distribution of the datapoints within the data. The datasets, which contained the outliers were deleted from the dataset. There is also the possibility of finding the median of the direct neighbours of the outlier, but this technique can lead to problems, if the neighbours are also outliers and thus contribute into the median calculation. [@Sin19]
Another aspect of data cleaning is the identification of values, which are *Not a Number* or NaN or NULL values. For this reason, the function in Listing 8 checks the whole dataset for those missing values and return *TRUE*, if there are missing values and *FALSE* if the dataset is cleared from those. If there are any missing values the simplest way to deal with those is the deletion of those.

```
print('Dieser Datensatz hat NaN oder NULL Werte: ' +
str(combined.isnull().values.any()))
```

Listing 8 – Checking Dataset for NaN or NULL Values

4.3.3 Feature Extraction

Feature Extraction describes the progress of making a dataset smaller and more convenient for machine learning. With this process it is possible to reduce the number of features within a dataset. This methodology is primarily used to only use features, which are most likely to improve the accuracy of a machine learning model and reduce the training time simultaneously.

4.3.3.1 Pearson Correlation

As described in the chapter 3.3.1. the Pearson correlation calculates the correlation between two variables. In the dataset of Glava there are eleven features in general. Listing 9 presents how the Pearson correlation is calculated. With the function *df.corr()* it is possible to calculate those correlations. The function is included in the *pandas* framework. Furthermore

```
#get correlations of each features in dataset
corrmat = df.corr(method=Feature_Selection_GLAVA.histogram_intersection)
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))

#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

Listing 9 – Pearson Correlation

Figure 17 represents the correlation from each feature of the Glava dataset to each other as a heatmap. The figures within the heatmap diverge between -1 and 1. A high association to minus one represents a negative correlation. This means, that if for example the total amount of energy in the system (Stot) increases, the corresponding figure grows in the negative way. Features which are in the positive scale of the Pearson correlation grows in the same direction as the observed value. If the relationship between two features is close to zero, there is no immediate correlation between those figures. In the heatmap of Figure 19 the connection between two values is red, if the correlation is negative, green for a positive connection and yellow, if the combination of those features diverges to zero.
Out of those eleven features only seven have a high correlation to the *Stot*, which represents the total amount of energy produced with the PV station. In this case the features *Wind Direction*, *Precipitation* and *Barometric Pressure* have a low impact on the output feature *Stot*. All of those variables are close to zero with the Pearson correlation. But those attributes do not only have a low impact on the energy output of the PV systems, but also on the other weather

features. The Pearson correlation does imply, that the seven features, which are going to influence the machine learning model the most are: *Temperature*, *Humidity*, *Wind Speed*, *Global Radiation*, *Radiation 30 Degrees*, *Radiation 40 Degrees* and *Indirect Radiation*. All of them, except of the *Humidity* have a positive Pearson Correlation, which means, that they are grow in the same way as the attribute *Stot*. *Humidity* is the only feature, which has a high negative correlation. This means, that it grows in exact the opposite direction as *Stot*. This makes sense because, when it is raining, there are also a lot of clouds, which affects the energy production of solar panels. These results were produced with a limited dataset of one month in the summer. The results could vary, with data from other seasons with more cloud occurrence. For the interpolated dataset the results for the different features are mostly similar. Except, that the amplitude is not so strong for each attribute. For example, the *Humidity* only received a score of -0.53, instead of the -0.62 in the case of the Glava dataset.

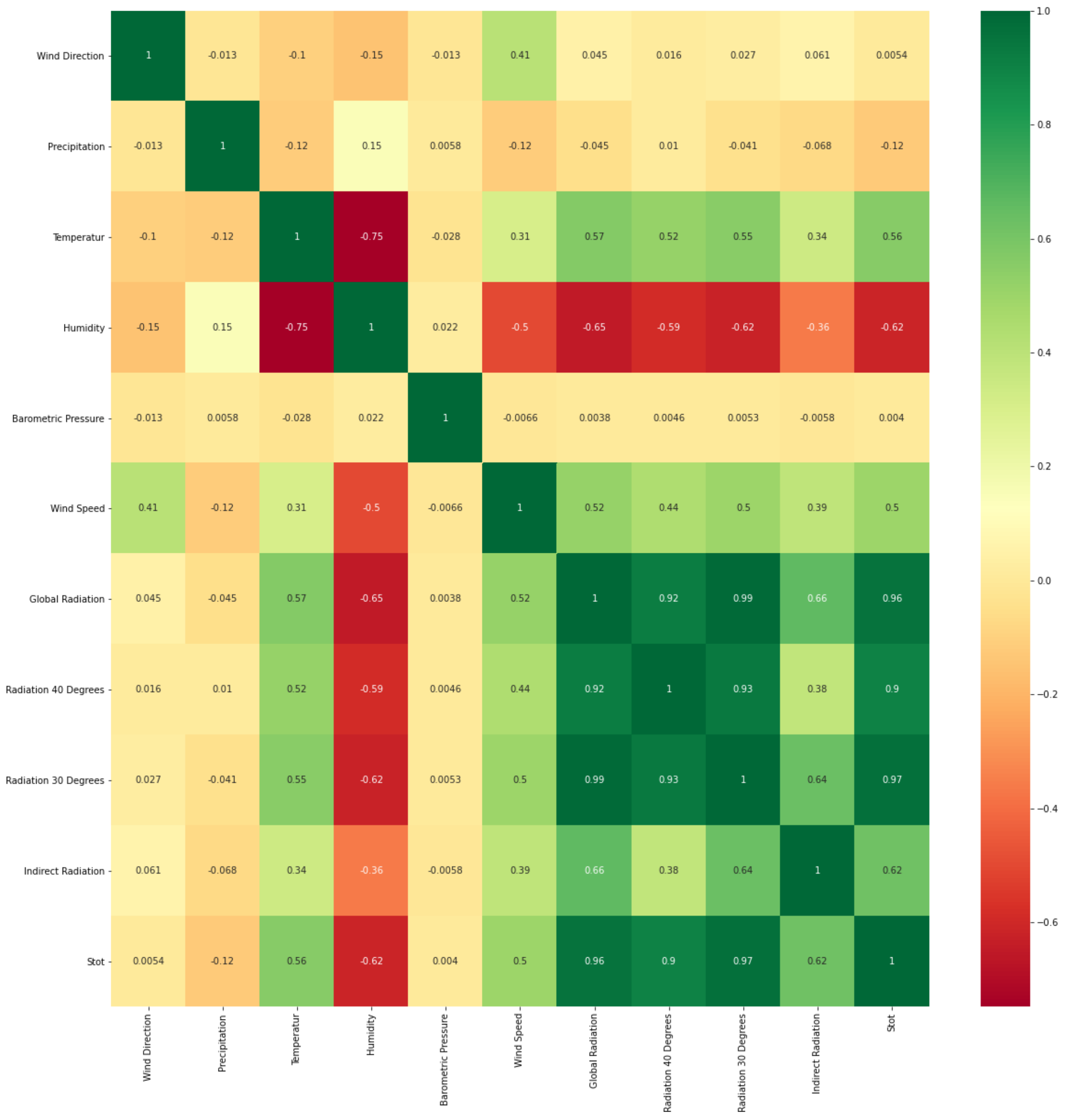


Figure 17 – Pearson Correlation Heatmap

4.3.3.2 Spearman Correlation

The Spearman correlation does, similar to the Pearson correlation, calculate the relation between two variables. As defined in chapter 3.3.2. the difference between the Spearman and the Pearson correlation is, that it also considers a difference in the increase or decrease of the two observed variables.

```
for column in df:
        result  = spearmanr(df[column],df['Stot'])
        print("Result " + str(df[column]) + ": " + str(result))
```

Listing 10 – Calculation of the Spearman Correlation

Listing 10 shows, how the Spearman correlation is calculated within the system. For this the implementation of the Python library *SciPy* was used. This library contains many different modules for i.e., optimization, linear algebra and interpolation. The function only needs all values of the two different features to calculate their correlation to each other.

Table 5 – Spearman Correlation Results

Variable name	Spearman Glava	Spearman Interpolated
Temperature	0.633	0.571
Wind Direction	0.042	0.166
Wind Speed	0.584	0.304
Humidity	-0.636	-0.591
Precipitation	-0.042	-0.060
Barometric Pressure	0.087	0.051
Global Radiation	0.977	0.932
40 Degrees Radiation	0.959	-
30 Degrees Radiation	0.984	-
Indirect Radiation	0.883	-

Table 5 represents the results for each feature in both datasets. The results are mostly similar to the ones from the Pearson Correlation. Except, that each result has a stronger amplitude in the positive or negative direction. Only the wind speed shows a stronger difference between the two tested datasets. But in general, the Spearman correlation suggests the same features.

4.3.3.3 Random Forest

As presented in the chapter 3.3.4. the random forest can be used for machine learning tasks, but also for the feature extraction. In this implementation the *ExtraTreeClassifier()* function from the python library *Sklearn* was used. This implementation uses the classification and regression trees (CART) methodology to build the trees.

```
# Select X and Y
Y = df['Stot']
X = df.drop("Stot", 1)

lab_enc = preprocessing.LabelEncoder()
Y = lab_enc.fit_transform(Y)

extra_tree_forest = ExtraTreesClassifier(n_estimators = 15, criterion
='gini', max_features = 7)
extra_tree_forest.fit(X, Y)
```

Listing 11 – Random Forest Implementation

Listing 11 presents the implementation of the random forest. The first step is the division of the dataset into *X* and *Y*. This means, that X represents the feature dataset which is going to be evaluated if they have a correlation to the output value, which is embodied by Y. Afterwards the random forest can be build up. There are three values which can be tweaked:

- *n_estimators*: described the number of trees which are going to be build
- *criterion*: measures the quality of the split, which includes the information gain
- *max_features*: number of features which are considered when looking for a split

In this case a number of 15 trees were chosen, because a higher number did not impact the variance of the results in general. There are two functions available for the criterion function: *Gini* and *Entropy*. Both work similar, except, that the Gini methodology does not require logarithmic functions for the computation, which makes it not as computational heavy as the entropy. Because of this the Gini criterion function was chosen. For the number of features seven were chosen, because of the previous testing the Pearson and Spearman Correlation.
Figure 18 presents the results of the random forest feature extraction. On the left side are the results for the Glava dataset, which states, that the most important features are: *Wind Direction, Temperature, Humidity, Global Radiation, 30 Degrees Radiation, 40 Degrees Radiation* and *Indirect Radiation*. This results almost reflects the outcome of the previous tests, except for the *Wind Direction* and *Wind Speed*. For the interpolated dataset the results are almost the same, but the informational gain for *Wind Speed* is rated higher than in the Glava dataset.

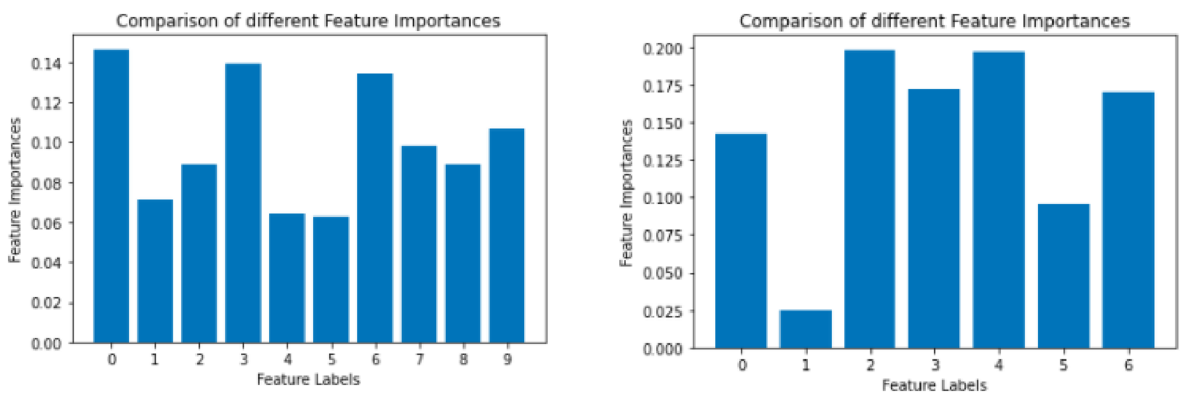


Figure 18 – Random Forest Results (left: Glava Dataset, right: Interpolated Dataset)

4.3.3.4 Principal Component Analysis

The PCA methodology generates a subset of different features and analyses how well those combinations work in correlation to the output feature. Because PCA converts high dimensional datasets into low dimensional ones, there is no resemble of the output principles to the real features anymore. If the results were used for the training and generating of machine learning model, only those principles can be used and not the real features. Because of this reason, the methodology was implemented but not used afterwards. Those principled could be used in the ongoing project to test if those principles are generating better results.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

pca = PCA()
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
```

Listing 12 – Implementation of PCA

Listing 12 describes the implementation of the PCA. Fist different training and test datasets have to be created. For this, the *train_test_split()* function of *Sklearn* were used. This function has two variables: *test_size* and *random_state*. The first variable controls the size of the test size and the second one sets, if there should be a random state within the datasets. Because time series data has to be in order, the *random_state* was set to zero. The next step scales the data with the help of the *StandartScaler()* from *Sklearn*. This function scales each value in the dataset individually by subtracting the mean and divides this new value by the standard deviation. Afterwards the *PCA* function of *Sklearn* is used to generate the different principles of the former dataset.

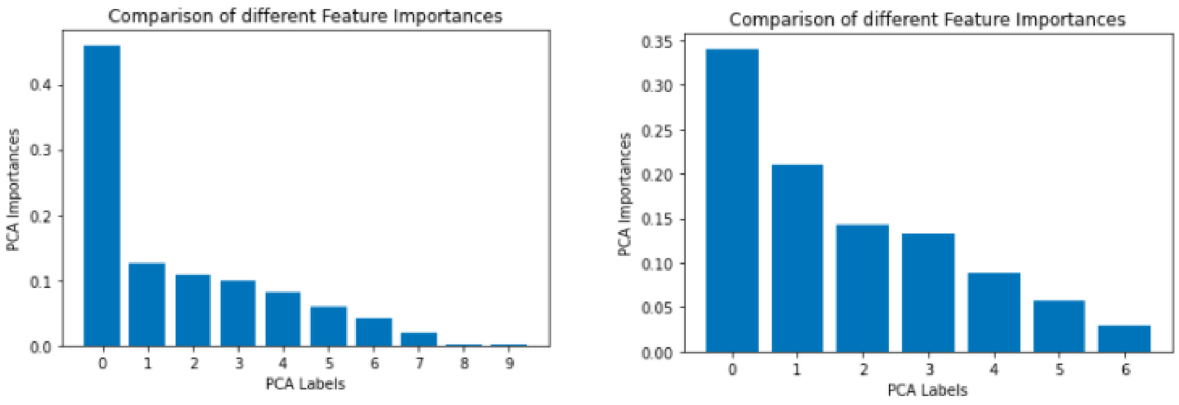


Figure 19 – PCA Results (left: Glava Dataset, right: Interpolated Dataset)

The results of the PCA shows, that in the Glava dataset there is one principal, which inherits over 40 percent of the informational gain. The other nine principles only contain a gain of close to ten percent and lower. For the interpolated data the informational gain of each principle is more distributed between them. This means, that if those principles were used for machine learning, there are more principles which can be ignored in the Glava dataset. This is not the case for the interpolated dataset. The reason for this could be, that there are less features in this dataset than in the Glava one.

4.3.3.5 Wrapper Feature Selection

The last feature extraction method is the wrapper selection. As described in chapter 3.3.5. there are five different wrapper selection algorithms. For all of those different methodologies the Python library *mlxtend* was used. This library is specialized in data science tasks. The user of the system can choose between those five different methodologies or run all of them at the same time. Listing 13 presents the implementation of those five wrapper methods. The first three methodologies are using the *SequentialFeatureSelector()* method from *mlxtend*. This method is a greedy search algorithm, which is used to reduce an initial d-dimensional feature space and reduces it to a k-dimensional one. Where $k < d$. For this task different estimator classes can be used. For this task the *LinearRegression()* and the *K-Nearest Neighbour()* functions from *Sklearn* were used as the estimators. The user of the system can also choose the number of features, which are going to be chosen with this methodology. The next two parameter *forward* and *floating* can be toggled between *True* and *False*. The *forward* variable represents if the forward selection or the backward elimination is going to be chosen. With the *floating* variable it is possible to activate the bi-directional wrapper selection.

```
sfs =
SFS(LinearRegression(),k_features=num_features,forward=True,floating=False,
scoring = 'r2',cv = 0)

sbs =
SFS(LinearRegression(),k_features=num_features,forward=False,floating=False
,cv=0)

sffs =
SFS(LinearRegression(),k_features=num_features,forward=True,floating=True,c
v=0)

efs = EFS(LinearRegression(),
min_features=1,max_features=num_features,scoring='r2',cv=10)
```

Listing 13 – Wrapper Feature Selection Implementation

The exhaustive feature selection is using the *EFS* function from *mlxtend*. This method is creating all possible subsets of the selected feature dataset evaluates over all of them. Because of that, the runtime of this function is substantial longer than the normal wrapper feature selection with an average 43 seconds in comparison to two seconds for the other wrapper methods.

Table 6 – Wrapper Selection Results, Glava Dataset

SFS	SBS	Bi-Directional	Exhaustive
Wind Direction	Wind Direction	Wind Direction	Wind Direction
Temperature	Temperature	Temperature	Temperature
Wind Speed	Wind Speed	Wind Speed	Wind Speed
Global Radiation	Global Radiation	Global Radiation	Global Radiation
Radiation 40 Degrees	Radiation 40 Degrees	Radiation 40 Degrees	Radiation 40 Degrees
Radiation 30 Degrees	Radiation 30 Degrees	Radiation 30 Degrees	Radiation 30 Degrees
Indirect Radiation	Indirect Radiation	Indirect Radiation	Indirect Radiation

Table 6 represents the results of the feature selection for each method. Each row represents a feature which was selected from all ten possible features. They are not in an increasing order of importance. As the table shows, are the results for each test exactly the same. But those results show, that with the wrapper selection also the feature *Wind Direction* seems to have an important role for the variable *Stot*. Those tests were made with the number of features set to seven.

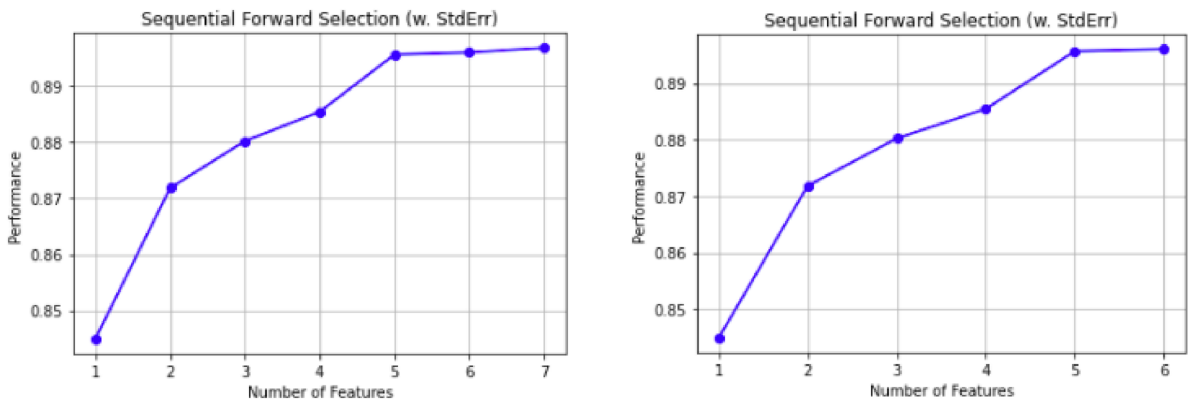


Figure 20 – SFS Wrapper Selection on Glava Dataset (left: seven features, right: six features)

Figure 20 represents the performance of the SFS wrapper method. On the left side seven features in total were selected and on the left side only six. It is visible, that the performance increase of using seven features instead of six is marginally. The feature, which is not in the subset of the six chosen features anymore is the *Wind Direction*. This means, that this feature is more important than the other four, but only effects the performance slightly in general.
For the interpolated dataset only seven features in general were accessible. For this reason, a subset of five features were chosen for the different wrapper methods. Surprisingly two features, which were not visible during the testing of the Glava dataset, were chosen during the tests with the interpolated data. Those features are the *Barometric Pressure* and the *Precipitation*. Another difference is the result of the exhaustive feature selection. Here only a subset of two features in total were chosen.

Table 7 – Wrapper Selection Results, Interpolated Dataset

SFS	**SBS**	**Bi-Directional**	**Exhaustive**
Barometric Pressure	Barometric Pressure	Barometric Pressure	Global Radiation
Temperature	Temperature	Temperature	Temperature
Wind Speed	Wind Speed	Wind Speed	-
Global Radiation	Global Radiation	Global Radiation	-
Precipitation	Precipitation	Precipitation	-

As Figure 21 presents, does the feature performance increase only slightly with the inclusion of five features instead of four. Because of this reason the tests were repeated with only four features and the *Precipitation* was the feature which was excluded this time.

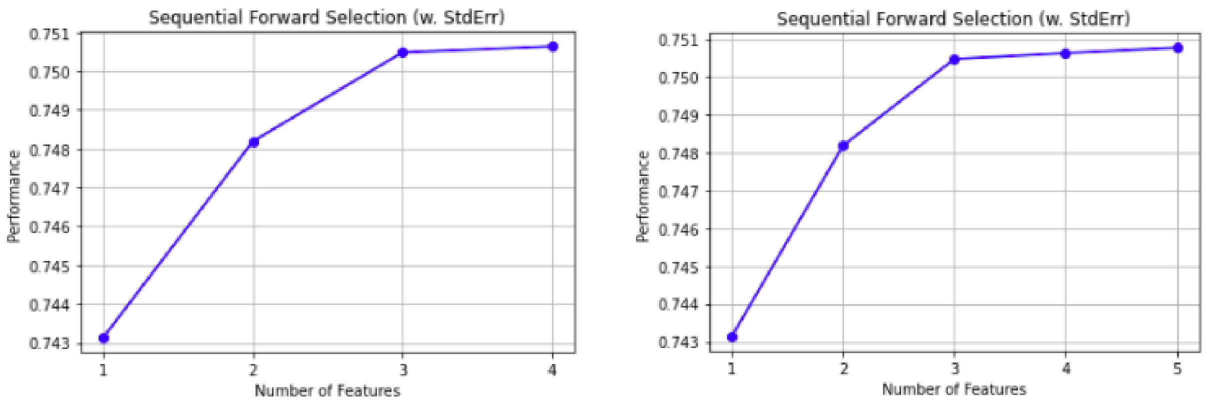


Figure 21 – SFS Wrapper Selection on Interpolated Dataset (left: four features, right: five features)

4.3.3.6 Conclusion

The five different methods for feature selection conclude dissimilar results for the ten or seven features in the two datasets. But not for all features. For example, the following features are important in every of the conducted tests: Temperature, Global Radiation, 30 Degrees Radiation, 40 Degrees Radiation, Indirect Radiation and Wind Speed. Because those features are important in every feature selection test, those will be taken for the implementation of the machine learning model. Furthermore, other paper like [ASM20] also did feature selection for weather information in the case of PV outcome prediction. The weather information were measured in the Applied Science Private University (ASU) in Amman, Jordan. At the same location, where the PV stations are located.
Table 8 concludes their results for the correlation between weather information and PV station energy outcome. The results conduct the same conclusion, that the *solar radiance* or *irradiance* are the most important weather features for the prediction of PV station energy production. Furthermore, the *temperature* and *humidity* also have a strong correlation with the output value. In addition to that, there are two features in this paper, which are not available in the Glava dataset: *cloud type* and *dew point*. Those features also have a strong correlation to the PV energy outcome but could not be tested in the scope of this thesis, due to the limitations of the datasets. There are two features, which are not comparable to the results shown above and those are: *precipitation* and *wind speed*. In the feature selection tests the correlation or informational gain between precipitation and the PV system energy outcome was not relevant enough in comparison to the other features. But in the testing of [ASM20] it shows a correla-

tion of 0.3409, which does not coincide with the results of the preceding tests. The feature wind speed does only have a correlation of 0.1970 in these testing, but reached i.e., a Pearson correlation of 0.5 in the previous testing.

Table 8 – Feature Correlation [ASM20]

Variable name	Correlation
Temperature	0.7615
Solar Irradiance	0.9840
Cloud Type	-0.4847
Dew Point	0.6386
Humidity	-0.4918
Precipitation	0.3409
Wind Direction	0.1263
Wind Speed	0.1970
Air Pressure	0.0815

Those inconsistences between the correlation of features can be observed, when the actual weather data is perceived closely. The dataset which is used in the paper [ASM20] is collected throughout two years and also consist of all four seasons and their different seasonal effects on the weather.

Table 9 – Spearman Correlation, Yearly and Monthly Datasets

Variable name	Spearman July Dataset	Spearman Yearly Dataset
Temperature	0.633	0.734
Wind Direction	0.042	0.156
Wind Speed	0.584	0.256
Humidity	-0.636	-0.467
Precipitation	-0.042	-0.040
Barometric Pressure	0.087	0.241
Global Radiation	0.977	0.952
40 Degrees Radiation	0.959	0.961
30 Degrees Radiation	0.984	0.988
Indirect Radiation	0.883	0.808

The data, which were used for the feature extraction in this paper consists only of weather information of July for five years. It is possible, that the seasonality also changes those results. These changes can be observed in Table 9. It is clearly visible, that the feature *wind speed* loses almost half of its importance. Also, the *barometric pressure* wins importance. Only the *precipitation* is still irrelevant to the PV energy outcome. Precipitation can only occur, when there are also clouds. Because of that, the paper [IK18] researched the variability in the data.

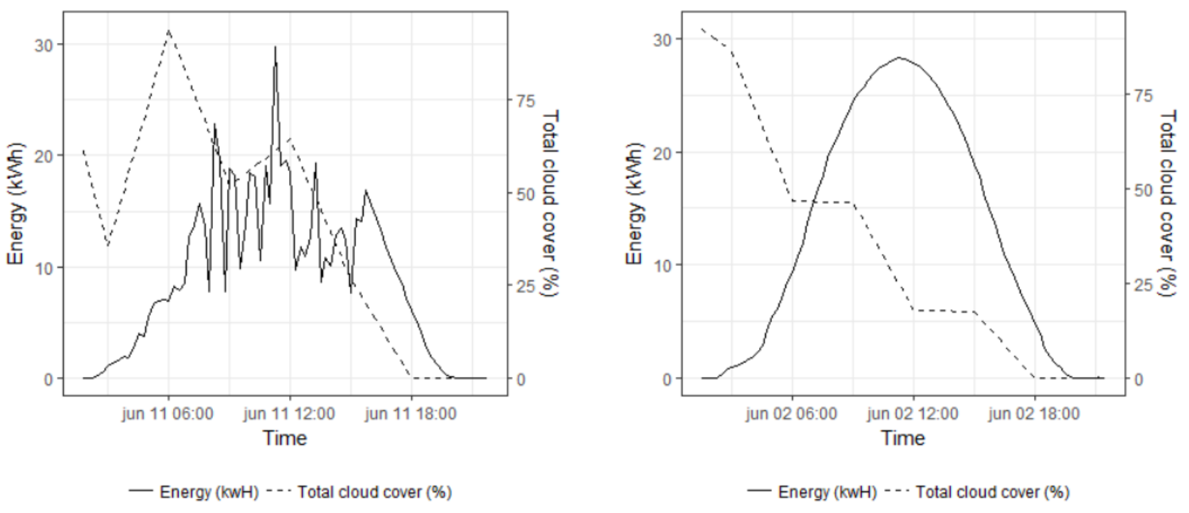


Figure 22 – Left: Cloudy Day, Right: Sunny Day [IK18]

Figure 22 displays this variability in the data of cloud coverage. The left image represents a cloudy day and the right one a sunny day. Both cloud coverage forecasts result in completely different PV energy outcome profiles. In general, those variabilities in the *precipitation* concludes to a low correlation between the PV system energy outcome and this feature.
In summary, there are seven features, which are going to be used for the creation of the machine learning model for the Glava dataset: *Wind Speed, Temperature, Humidity, Global Radiation, 30 Degrees Radiation, 40 Degrees Radiation* and *Indirect Radiation*. And for the interpolated dataset the following features are going to be considered: *Temperature, Wind Speed, Global Radiation* and *Humidity*.

4.4 Machine Learning Model

There are different methodologies for the prediction of time series data. In the scope of this master thesis three different kind of machine learning model are going to be used: Random Forest, LSTM and Facebook Prophet. For each of those methods three kind of model were created:

Table 10 – Machine Learning Model

Input	Output
Weather features	PV power outcome
Weather and PV Feature	PV power outcome
Weather Features	Weather Features

The following sub-chapter of the master thesis contains the implementation of the in chapter 3 mathematically explained model.

4.4.1 Random Forest

As described in chapter 4.3.3.3. random forest can be either used for classification i.e., feature extraction, but also for regression. For the task of predicting time series data this characteristic is needed. Listing 14 describes the implementation of the random forest prediction model. In the first step of the training section the whole dataset is split up in *X* and *Y*. Furthermore, those two separate datasets are split up into *X_train, X_test, Y_train* and *Y_test*. Whereas the training splits represent the dataset for the training and the testing ones the dataset for the testing afterwards. To create those subsets of the dataset the function *train_test_split()* of *Sklearn* was used.
The *Sklearn* function *RandomForestRegressor()* was used as the regression function. It inherits the same variables as the *ExtraTreeClassifier(),* used in the previous chapter to filter features. For the evaluation of the machine learning model the *metrics* functions from *Sklearn* were used. In detail the mean absolute error, mean squared error and the root mean squared error, mean absolute percentage error and the R^2 score. Afterwards the actual values are going to presented overlapping the predicted ones for presentation purposes. The user of the system can choose between creating different modes, for the three different machine learning model and also for the two datasets.

```
# Training
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=test_size, shuffle=False)

regressor = RandomForestRegressor(n_estimators=2000, random_state=0)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

# Evaluation
print('Mean Absolute Error:',
metrics.mean_absolute_error(test_list,pred_list))
print('Mean Squared Error:', metrics.mean_squared_error(test_list,
pred_list))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(test_list, pred_list)))
print('Mean Absolute Percentage Error:' ,
(metrics.mean_absolute_error(test_list,pred_list)*100))
print('R2 Score:' , metrics.r2_score(test_list,pred_list))


plt.figure(figsize=(18, 6))
plt.plot(test_list, label='Actual Datapoint')
plt.plot(pred_list, label='RandomForestRegressor')
plt.tick_params(axis='x', which='both', bottom=False,
top=False,labelbottom=False)

plt.ylabel('Predicted ' + temp_list[i])
plt.xlabel('Time Steps')
plt.legend(loc="best")
plt.title('Regressor predictions and their average')
plt.show()
```

Listing 14 – Random Forest Regression Implementation

To find the best hyperparameter for a random forest, the *Sklearn* function *RandomizedSearchCV()* was used. This function uses a grid of pre-defined hyperparameter ranges and randomly samples each possible combination to find the best solution with K-Fold cross validation.

Listing 15 presents the creation of the grid, which consists of four adjustable variables: n_estimators, max_features, max_depth and bootstrap. For each of those variables a predefined range was selected. Afterwards a random forest is going to be filled with this grid and the best solution for those variables are iterative selected.

```
# Hyperparameters
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 2000, num =
2000)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(1, 110, num = 110)]
max_depth.append(None)
# Minimum number of samples required to split a node
bootstrap = [True, False]
# Create the random grid
random_grid = {
'n_estimators': n_estimators,
'max_features': max_features,
'max_depth': max_depth,
'bootstrap': bootstrap}

rf_random = RandomizedSearchCV(estimator = rf, param_distributions =
random_grid, n_iter = 1000, cv = 3, verbose=2, random_state=0, n_jobs = -1)

rf_random.fit(X_train, y_train)
best_random = rf_random.best_estimator_
```

Listing 15 – Hyperparameter Tuning Random Forest

For this task the *RandomizedSearchCV()* function selected the following variables as the best solution:

Table 11 – Hyperparameter Tuning Results Random Forest

Variable	Selected Value	Previous Value
n_estimators	681	10
max_features	auto	auto
max_depth	108	50
bootstrap	True	True

Afterwards those results were compared with the previous used values for those variables. The mean squared error of the previous model was 59.31 and with the new values set for the variables, the error decreased to 43.20, which is an improvement of 27.16 percent.

4.4.2 Facebook Prophet

With the Facebook Prophet library, it is possible to also make predictions on time series data. The methodology was already explained in chapter 3.4.2. and this sub-chapter treats the implementation of the three different machine learning model, which are going to be evaluated in this master thesis.
The input structure of Facebook Prophet consists of two columns within the dataset: *ds* and *y*. Because the Facebook Prophet model is implemented to use continuous datasets as an input and performs poor with bigger gaps in between time steps, only one month of input data was chosen.
The column *ds* describe the timestamp of each value. The date has to be in the correct Pandas format. The second column *y* consists of the actual feature, which is going to be predicted. The Listing 16 presents the restructuring of the Glava dataset. The user of the system chooses a feature, which is going to be predicted. Out of the whole Glava dataset, with all features this one is going to be selected and chosen as the *y* parameter. Furthermore, the date column of the Glava column had to be adjusted. The Glava date timestamp includes the time zone, which is not compatible with the structure of Facebook Prophet.

```
features_considered = [feature]
df = df[features_considered]
df = df.reset_index()

# recreate the column names
df.columns = ['ds','y']

# Removing the timezone from the date
df['ds'] = pd.to_datetime(df['ds'], format='%d-%b-%Y:%H:%M:%S' , utc=True)
df['ds'] = df['ds'].dt.tz_convert(None)
```

Listing 16 – Facebook Prophet Data Structure

The next step for the prediction with Facebook Prophet is the creation of the training and test datasets, which is shown in Listing 17. Because Facebook Prophet always assumes that the dataset is continuously, only one year of the six available were chosen. Otherwise, the results of the prediction would be inconclusive. Afterwards a new *Prophet-Object* is going to be created. For this the *yearly_seasonality* was set to *False*, because there is no seasonality in a dataset, consisting of only one month of values. In the end the historical dataset is going to be *fit* in the Prophet model.

```
# Creating the test and training data
print("Creating the test and training sets")
train = df[(df['ds'] >= '2019-07-01') & (df['ds'] <= '2019-07-31
23:00:00')]
test  = df[(df['ds'] > '2019-07-31')]

m = Prophet(yearly_seasonality=False)

print("Fitting the Regressors into the model!")
m.fit(train)
```

Listing 17 – Facebook Prophet Fitting the Model

After the model is fitted into the Prophet model, it is possible to create a prediction out of it. For this, the function *make_future_dataframe()* was used to create a new dataframe which also extends into the future. This function requires a period and a frequency. In the Listing 18 a future dataframe consisting a forecast of 24 hours in the future is going to be created. Afterwards the prediction is going to be executed with the *predict()* function. For each row in the future dataset a predicted value is going to be assigned. The prediction returns three different variables: *yhat*, *yhat_lower* and *yhat_upper*. Where yhat represents the actual predicted value and the other two contain the upper and lower border for the uncertainty interval.

```
future = m.make_future_dataframe(periods=24, freq='H')
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
fig1 = m.plot(forecast)
fig2 = m.plot_components(forecast)

print("Cross Validation: ")
cv_results = cross_validation(model = m, horizon ="24 hours")
df_p = performance_metrics(cv_results)
print(str(df_p))

fig3 = plot_cross_validation_metric(cv_results, metric='mape')
```

Listing 18 – Facebook Prophet Creating the Model

Furthermore, the Facebook Prophet library contains the functionality of cross-validate the predictions. For this the function *cross_validation()* was used. This function needs the trained model, an initial span of data from the dataset, a horizon and a period. Figure 23 presents, how those variables are distributed in the dataset. The initial variable is the initial training period, the horizon is the forecast horizon, and the period is represented by cutoff in the Figure 23. For the validation only the horizon has to be set, the others are going to be determined automatically. The initial training period is set three times the horizon and the period is created for every half a horizon.

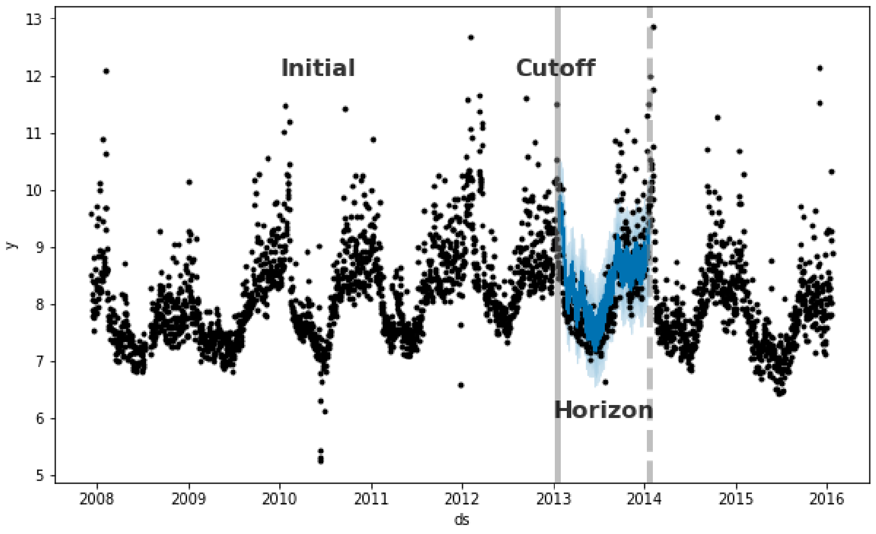


Figure 23 – Facebook Prophet Cross-Validation [@Pro20]

Because those functionalities are only working for univariate datasets, a workaround had to be implemented. For each new feature, which is going to be included in the prediction a new regressor had to be included in the Facebook Prophet model. Furthermore, those regressors already need to be future values. Because there are no future values for those features in the dataset, they had to be predicted beforehand with the same methodology as for the univariate presented previously. For this the iterative function *create_future_feature()* was created.

```
future_dict = create_future_feature()
data.columns = ['ds','y','Temperatur', 'Humidity' , 'Wind Speed','Global
Radiation', 'Indirect Radiation','30 Degrees Radiation', '40 Degrees
Radiation']

m.add_regressor('Temperatur')
m.add_regressor('Humidity')
m.add_regressor('Wind Speed')
m.add_regressor('Global Radiation')
m.add_regressor('Indirect Radiation')
m.add_regressor('30 Degrees Radiation')
m.add_regressor('40 Degrees Radiation')
```

Listing 19 – Facebook Prophet Multivariate Prediction

Afterwards the different columns of the Glava dataset are going to be added and named after the naming conventions of Facebook Prophet, where *y* represents the output feature. Each of

the pre-predicted features are then added to the Prophet model with the *add_regressor()* function. The fitting, creation of the model and the prediction afterwards is working in the same way as in the case of the univariate prediction.
For the hyperparameter tuning of the Facebook Prophet machine learning model a similar approach as the one from random forest is used. First a parameter grid has to be created with the possible variables. For the Prophet framework, the number of parameters, which can be observed is limited. Only the following parameter can be checked:

- Changepoint_prior_scale: Describes the flexibility of the trend and how much the trend changes the observed value.
- Seasonality_prior_scale: This parameter controls the flexibility of the seasonality.
- Holidays_prior_scale: This controls the flexibility to fit holiday effects.

Afterwards all combinations of the parameters are going to be created and for each combination a new model is trained. The results are stored in an array and the best result is presented.

```
param_grid = {
'changepoint_prior_scale': [0.001, 0.01, 0.1, 0.5],
'seasonality_prior_scale': [0.01, 0.1, 1.0,5.0, 10.0],
'holidays_prior_scale' :   [0.01,0.1,1.0,5.0,10.0]
}

# Generate all combinations of parameters
all_params = [dict(zip(param_grid.keys(), v)) for v in
itertools.product(*param_grid.values())]
rmses = []

# Use cross validation to evaluate all parameters
for params in all_params:
    m = Prophet(**params).fit(df)  # Fit model with given params
    df_cv = cross_validation(m, horizon='24 hours', parallel="processes")
    df_p = performance_metrics(df_cv, rolling_window=1)
    rmses.append(df_p['rmse'].values[0])

best_params = all_params[np.argmin(rmses)]
```

Listing 20 – Hyperparameter Tuning Facebook Prophet

As Table 12 presents, there are three parameters, which can improve the performance of the model in general. Especially the *holidays_prior_scale* changed from 10 to 0.01. This could be, because Prophet assumes a dataset in a yearly manner and the Glava dataset only consists of data from one month.

Table 12 – Hyperparameter Tuning Facebook Prophet Results

Variable	Selected Value	Previous Value
Changepoint_prior_scale	0.01	0.05
Seasonality_prior_scale	10	10
Holidays_prior_scale	0.01	10

Both variants were used to compare the quality of the prediction. With the new parameters a mean squared error of 33.90 was achieved. The previous model with the default values was only minor worse than the new model with a MSE of 33.95. Those results come to the conclusion, that the three adjustable parameters, do not have such a high impact on the prediction. Mainly, because those variables primarily target effects of seasonality or holidays on the dataset. Those effects are not visible in the observed dataset.

4.4.3 LSTM

The methodology of LSTM machine learning model was already introduced in chapter 3.4.1. and this sub-chapter implies the implementation. For a variation of testing scenarios, a total of 18 different kinds of LSTM model were made. As presented in Table 10, there are three combinations of input and output features. In addition to that, three implementations of the LSTM model itself were produced. For the simplest one only a single LSTM-Layer in combination with a dense-Layer were used. The more complex one inherits multiple stacked LSTM-Layer. The last one introduces the usage of bi-directional LSTM-Layer.
For the implementation of a LSTM network is divided in three steps: preparing the dataset for the usage of a LSTM machine learning model, defining and fitting the model and the last step is the hyperparameter tuning and evaluation of the specific model.

```
# Load the Dataset
df = Training_Model.loading_dataframe(filepath)

# Normalize the Dataset
scaler = MinMaxScaler(feature_range=(0,1))
df = scaler.fit_transform(df)

# Create the training and testing Dataset
x_train_multi, y_train_multi = Training_Model.multivariate_data(df[:,0:8],
df[:,7:8], 0,TRAIN_SPLIT, past,future, STEP)
x_val_multi, y_val_multi = Training_Model.multivariate_data(df[:,0:8],
df[:,7:8],TRAIN_SPLIT, None, past,future, STEP)
```

Listing 21 – LSTM Data Preparation

Listing 21 presents a sample preparation of the Glava dataset for the training of a LSTM model. After the model was loaded, all the values within that dataset has to be normalized. For the normalization, the *Sklearn* function *MinMaxScaler()* was used. The theory of this scaler was described in chapter 3.2.3 with the equalization (5). For the training a total of four subsets of the whole dataset have to be created. For the training and testing each two separate datasets for the input and output have to be constructed. For the construction of multistep and multivariate datasets the function *multivariate_data()* was created. This function just needs the input and output shape, what sizes the different datasets need and the step. The step characterises in this case, in what timesteps it samples the data.

```
multi_step_model = tf.keras.models.Sequential()

multi_step_model.add(Bidirectional(tf.keras.layers.LSTM(32,return_sequences
=False,input_shape=x_train_multi.shape[-2:])))
multi_step_model.add(tf.keras.layers.RepeatVector(output))
multi_step_model.add(Dropout(0.2))
multi_step_model.add(Bidirectional(tf.keras.layers.LSTM(32,return_sequences
=True)))
multi_step_model.add(Dropout(0.2))
multi_step_model.add(Bidirectional(tf.keras.layers.LSTM(32,return_sequences
=True)))
multi_step_model.add(Dropout(0.2))
multi_step_model.add(Bidirectional(tf.keras.layers.LSTM(32,return_sequences
=True)))
multi_step_model.add(Dropout(0.2))
multi_step_model.add(tf.keras.layers.Dense(x_train_multi.shape[2]))

optimizer = tf.keras.optimizers.RMSprop(lr=0.003, clipvalue=1.0)
multi_step_model.compile(loss = "mse", optimizer = optimizer, metrics =
['mae', 'mse', 'mape', 'rmse'])
```

Listing 22 – LSTM Model Definition

After the different training and validation datasets were created, the next step is the definition of the machine learning model. Listing 22 describes the definition of a stacked LSTM machine learning model with four bi-directional LSTM-Layer. As defined in chapter 3.4.1 the LSTM-Layer contain internal activation functions (tanh and as a recurrent activation the sigmoid function) and thus the LSTM-Layer do not need additional activation functions. The 32 parameter describes the dimensionality of the output space.
As a model type a sequential model was chosen. After testing different combinations with the parameter *return_sequences* from the LSTM-Layer and the *RepeatVector()* layer of Keras, the best results resulted with the usage of a *RepeatVector()* after the first LSTM-Layer and afterwards use the *return_sequences* statement. To reduce overfitting the model the additional layer *Dropout()* was used.
The last layer, which is used for the bi-directional LSTM model, is the *Dense-Layer*. With the help of this last layer, it is possible to reduce the number of input data to a specific number of output values. In the case of Listing 21 it is possible to reduce the number of features, which are outputted to the same amount of input features.
Afterwards the optimizer is defined. There are several optimizers available with the Keras framework [@Ker20], but for the usage of LSTM machine learning model the *Adam* optimizer outperforms the other ones [@Seb20]. The same behaviour was visible during the creation

of the different LSTM model. To reduce or overcome exploding gradients within the LSTM model also the *clipvalue* parameter was set.
Exploding gradients is the problem, when the weights of the machine learning model converge to a value greater than one. Because in that case, the subsequent multiplications between and within the layer will increase the gradient exponentially. This is the opposite of the vanishing gradient, where the values become too small. [MAB20]
The last step is the compiling of the machine learning model. For this, the *compile* function of Keras was used. It needs a loss function, an optimizer and an error metric. As a loss function the mean squared error was used and the previous explained optimizer. For the evaluation of the performance four different metrics were used. The mathematical explanation of those metrics took place in chapter 3.5. The R^2 evaluation metric is not supported within the Keras framework and thus was used in the evaluation later on.
Afterwards the compiled LSTM model can now be fitted, which is presented in Listing 23. The *fit()* function needs several input variables to kick off the training: the training and validation datasets, the amount of epochs and steps. Furthermore, it is possible to create callbacks, which are able to trigger actions during the various stage of the training. Here one callback for the *earlystopping* was created. This callback checks for each epoch in the training, a specific *monitor*. In this case the *validation_loss* was observed. Furthermore, the parameter *patience* can be adjusted, which adds a delay to the cancellation of the training. If the argument *restore_best_weights* was set to *True*, the weights from previous epochs with the best value is chosen and maybe not the ones from the actual epoch.

```
early_stopping = EarlyStopping(monitor='val_loss', patience = 3,
restore_best_weights=True)

multi_step_history = multi_step_model.fit(train_data_multi,
                         epochs=EPOCHS,
                         steps_per_epoch=EVALUATION_INTERVAL,
                         validation_data=val_data_multi,
                         validation_steps=EVALUATION_INTERVAL,
                         verbose=1,
                          callbacks=[early_stopping])
```

Listing 23 – LSTM Model Fitting

For the hyperparameter tuning of the LSTM model, the library *Kerastuner* was used. With this library it is possible to evaluate the number of units within each layer and the learning rate. Listing 24 presents the implementation of the *Hypermodel*. This special model represents the future structure of the real model. There are only minor differences between this model and the actual one. The first difference is that the first layer does not need the input shape of the dataset. Secondly the observed parameter *units* and *learning_rate* now contain a range of possible values instead of a fixed one.

```
def build_model_small(hp):
    multi_step_model = tf.keras.models.Sequential()
    multi_step_model.add(tf.keras.layers.LSTM(units =
hp.Int('units',min_value=32,max_value=512,step=32),return_sequences=False))

    multi_step_model.add(tf.keras.layers.RepeatVector(output))
    multi_step_model.add(tf.keras.layers.LSTM(units =
hp.Int('units',min_value=32,max_value=512,step=32),
activation='relu',return_sequences=True))

    multi_step_model.add(tf.keras.layers.Dense(1))
    multi_step_model.compile(loss = "mse", optimizer =
keras.optimizers.Adam(hp.Choice('learning_rate',values=[1e-2, 1e-3, 1e-
4])), metrics = ['mae', 'mse'])
```

Listing 24 – LSTM Model Hyperparameter Tuning Hypermodel

After the creation of the Hypermodel, the tuner for the hyperparameter tuning is going to be defined. For this the framework pre-defines four different tuners: *RandomSearch*, *Hyperband*, *BayesianOptimization* and *Sklearn*.

```
bayesian_opt_tuner = BayesianOptimization(
                    Training_Model.build_model_big,
                    objective='mse',
                    max_trials=3,
                    executions_per_trial=1,
                    overwrite=True,
                    directory=os.path.normpath('BIG'))


bayesian_opt_tuner.search(x_train_multi,
y_train_multi,epochs=EPOCHS,validation_split=0.2,verbose=1)
bayesian_opt_tuner.get_best_hyperparameters()[0].values
bayesian_opt_tuner.get_best_models()[0].summary()
```

Listing 25 – LSTM Model Hyperparameter Tuning Tuner

For the evaluation of the best parameters the Bayesian optimization was used, as presented in Listing 25. The *objective* describes the metric to maximize or minimize, which is the mean square error in this case. The *max_trials* parameter defines the number of model configurations are tested at most. If the search space is exhausted, the tuner will stop the search before reaching the pre-defined value. *Execution_per_trial* is the number of models that should be built and fit for each trial. The Boolean *overwrite* reloads an existing hyperparameter project, if there was one created beforehand. And the last parameter *directory* sets the path where the results of the hyperparameter testing are stored.

Table 13 – Hyperparameter Tuning LSTM Results

Variable	Selected Value	Previous Value
Units	63	32
Learning Rate	0.001	0.003

Table 13 presents the results of the hyperparameter tuning. The results suggest a number of units for each LSTM-Layer of 63, instead of the previous used 32. For the learning rate, it proposes a decreasing from 0.003 to 0.001. Both options were afterwards tested with the training dataset. The model with the new selected parameters achieved a mean squared error of 22.41 and the previous one had an MSE of 28.20 for the prediction. That is an improvement of 20.53 percent.

5 Evaluation

In this section, we evaluate our approaches focussing on prediction quality of the different models. In addition, we evaluate the impact of using interpolated data on prediction quality.
For the evaluation of the different models, we use metrics presented in chapter 3.5. Furthermore, a window size analyzation and a walk-forward cross validation of the training and test datasets was performed.
In addition to that, the dataset with the actual weather information from Glava was benchmarked against the interpolated weather information from the other dataset. For the comparison of each model the datasets used in this work are normalized with the *MinMax-Scaler*. That means, that also the calculated MSE is presented as normalized in the end.

5.1 Window Size Analysis

For each of the in chapter 4.4 presented machine learning model a window size analysis was carried out. The general assumption is, that the greater the window size of the training data, the better the performance of the actual machine learning model. [KU18]
The window size of the training dataset was set between 5 and 95 percent of the whole dataset. For each step, the amount of data in the training dataset was increased by five percent.

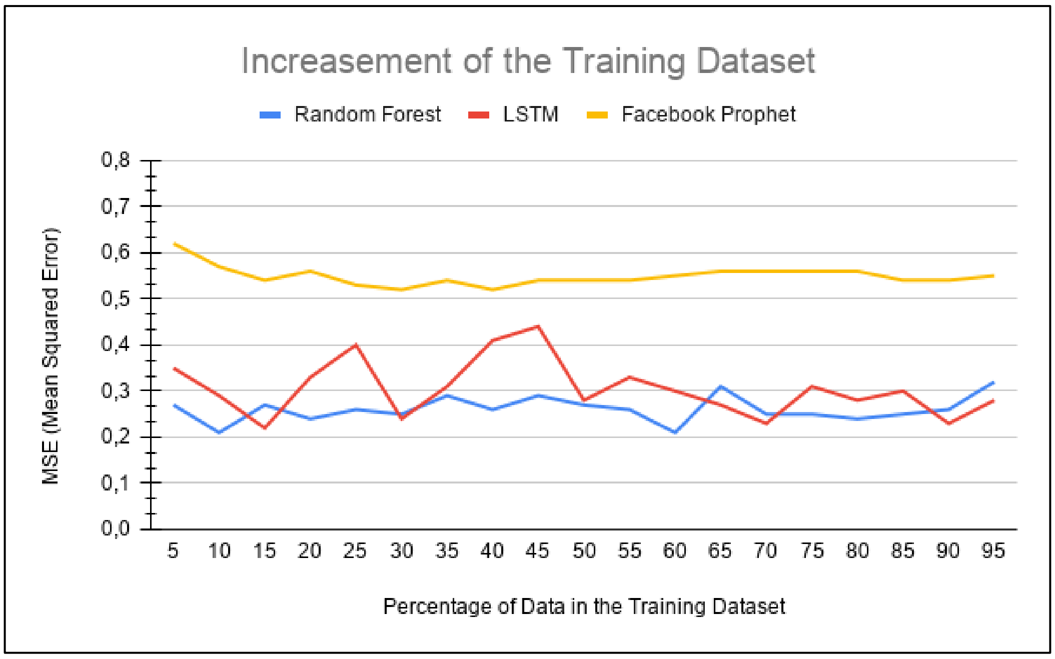


Figure 24 – Training Dataset Analysis

Figure 24 presents the effect of the increasing amount of data within the training dataset. The MSE was calculated between the actual and interpolated values, which are normalized according to (5). The impact of a training dataset lower than 50 percent is clearly visible on the LSTM model. Afterwards, with a bigger dataset the prediction error stabilized itself. Whereas the Facebook Prophet model only show an unstable state at the beginning, with a small dataset for the training. After the data within the training set increased to over 20 percent, the prediction error is stabilizing. For the random forest, the fluctuation of the mean squared error is unstable for most of the time and behaves in a similar way as the LSTM model. The model stabilizes itself with a training dataset over 65 percent.

5.2 Walk-Forward Cross Validation

The walk-forward cross validation is an important evaluation method, which is especially useful for time series data, where the order of the data is mandatory. Figure 25 illustrates the functionality of this methodology. Each row represents the whole possible time steps. In the first row, only six time steps (blue dots) were used as a training dataset to predict the next timestep (red dot). After each step the amount of data in the training set is increased by one time step. It is also possible to create bigger steps than singular ones.

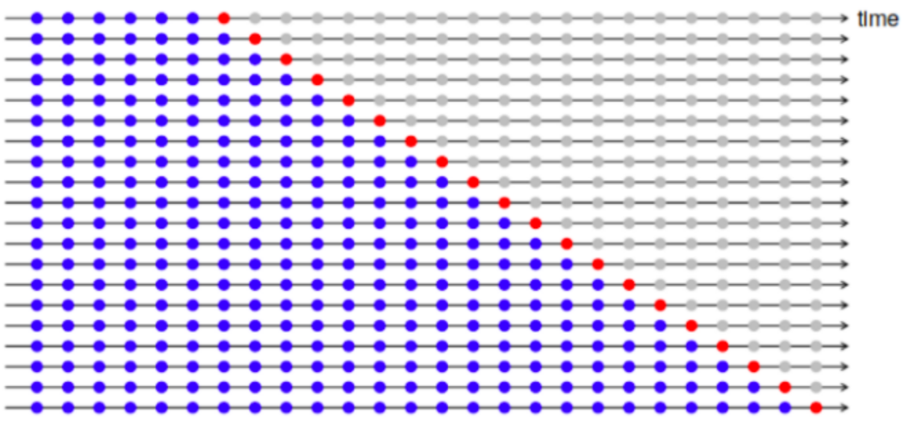


Figure 25 – Walk-Forward Cross Validation [HA18]

The performance of each machine learning model was measured with MSE between the predicted and the actual values for each step in the cross validation. As mentioned previously, a bigger step count was chosen for the cross validation. In this test scenario for each iteration the step was increased by two hours until the prediction horizon reached 24 hours, with a dataset which contains hourly data. Figure 26 describes the different errors of each machine learning model during the different periods. For the cross validation of the LSTM and the random forest the whole dataset was used. Because the dataset contains six months of data with 31 days, a total of 4464 time steps were available for the cross validation. For the Facebook Prophet model only one month of data was used, because of the limitations of the framework with gaps within the continuous dataset as described in chapter 4.2.2. The Facebook Prophet model is completely stable during the cross validation. But in general, the MSE shows, that it does not perform as good as the other two methodologies. Normally, machine learning model do perform worse, the higher the prediction horizon is set. This behaviour is visible in the representation of the LSTM model with an increasing horizon. But at a horizon of 12 hours, the error decreases again. This behaviour of LSTM model with an increasing horizon is also visible during the experiments in [BFO18]. The random forest model has the lowest error from the three machine learning techniques used. Furthermore, the error increases when the forecast horizon is extended, but not as massive, as the LSTM model in the beginning.

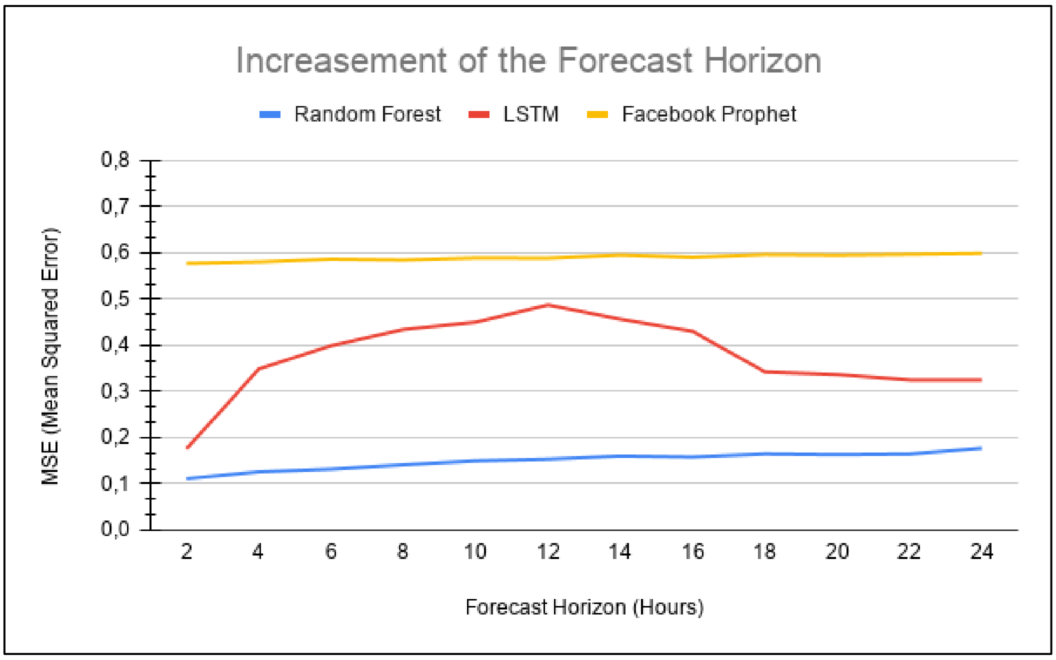


Figure 26 – Walk-Forward Cross Validation

5.3 Ex-Ante Forecasting Performance

This sub-chapter introduces the evaluation of the performance of the different machine learning model. The machine learning model use previous observations of 120 hours, which are five days, to predict the next 24 hours. This process of only use existing information of the data is called *Ex-Post*. The opposite methodology is *Ex-Ante*. With this methodology observations, which extends beyond the time steps of the executed forecast are used for the prediction. An example for these terminologies is that, in the case of *Ex-Ante*, the machine learning model is predicting over test data, which it has not seen before. While with the *Ex-Post* method, the model predicts over a dataset, which is known by the model itself.

5.3.1 Evaluation of the Residuals

Residuals describe the difference between the actual observations and the forecasted values. This distribution of the residual describes whether a model has adequately learned all information within a given dataset. A well-trained forecasting should have a residuals outcome with the following properties [KU18]:

- The residuals should be uncorrelated
- The residuals should be normally distributed
- The residuals should have a mean of zero

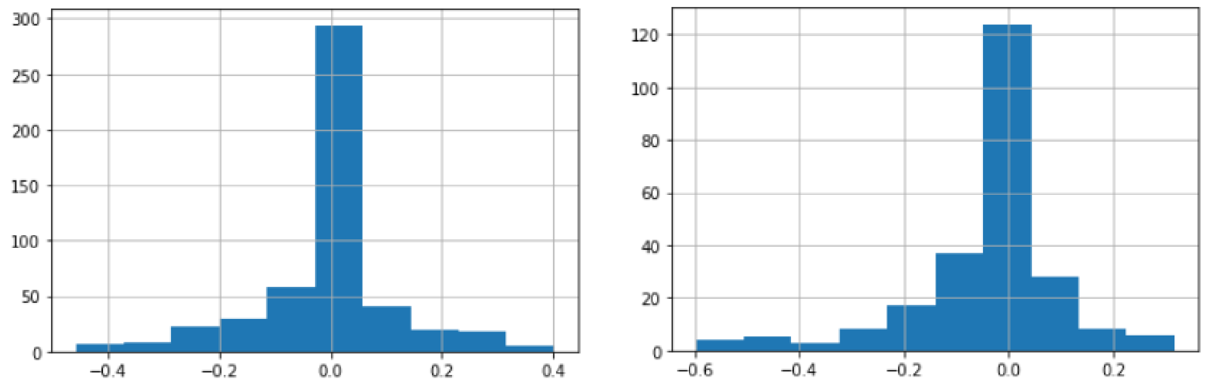


Figure 27 – Residual Distribution (left: Facebook Prophet, right: LSTM)

Figure 27 presents the residual distributions of the Facebook Prophet and LSTM model. The Y-axes describes the frequency of the residual and the X-axes the residual itself. Both arrangements have a normal distribution and a mean around zero which propose that the models do not have any correlation between the residuals. But both residual graphs have a moderate left tail.

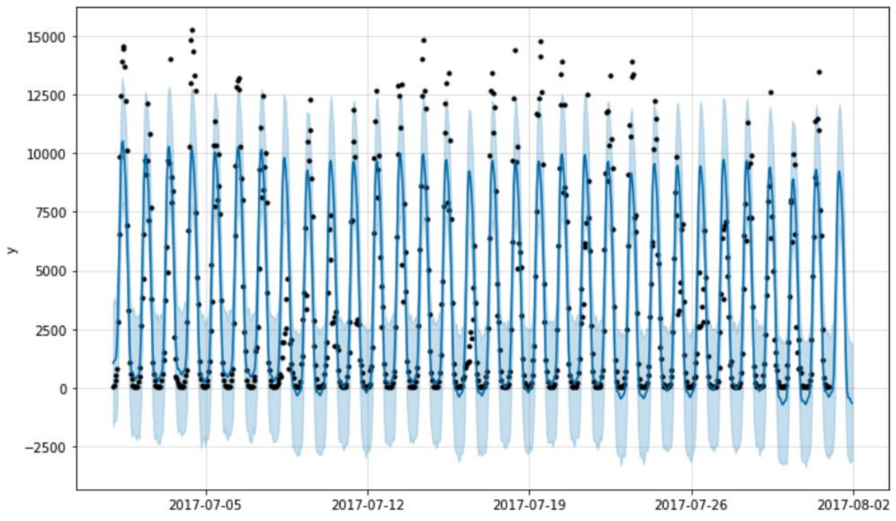


Figure 28 – Facebook Prophet Ex-Ante Forecasting Performance

Figure 28 shows the prediction of the Facebook Prophet model. The Y-axes represents the PV station energy production. In general, the predictions are close to the actual values of the dataset. In addition to that the framework creates an upper and lower interval for each time step, which is represented by the light-blue areas in the graph. The blue wave line is the predicted forecast, and the black dots show the actual datapoints. The forecast horizon is 24 hours, while the graph also shows the results of the training.

The prediction of the bi-directional LSTM model in Figure 29 also shows the 24-hour prediction. The red dots represent the predicted values and the blue ones the true values. It shows that the algorithm learned the pattern well, but with sudden changes within the dataset it struggles.

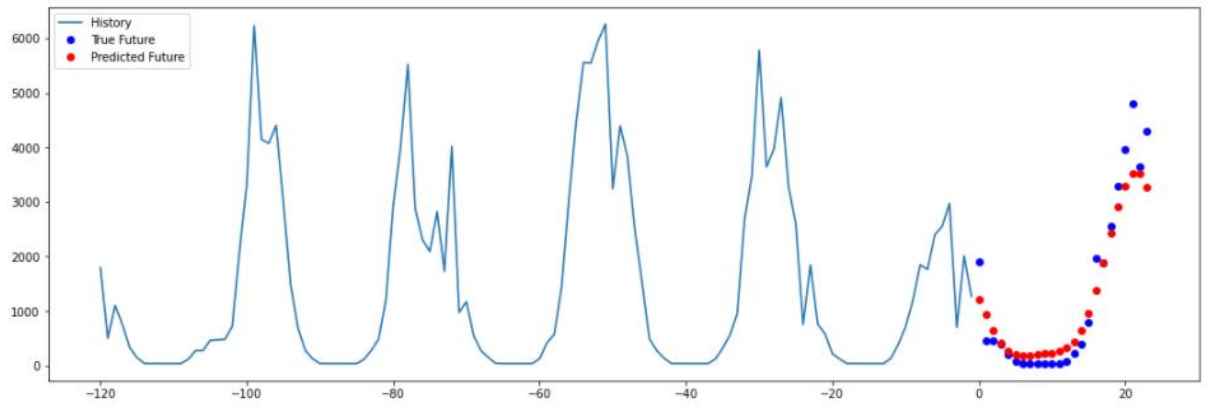


Figure 29 – Bi-Directional LSTM Forecasting Performance

5.3.2 Evaluation of Forecasting Performance

For the uncertainity of the prediction of renewable energy production, two different datasets were compared. In general, two sets of machine learning model were created. One with the precise weather information and one with the interpolated ones. For the prediction each of the created model received both of the datasets individually, which is presented in Table 14. That means, that i.e., the model trained with the actual weather information made a prediction with the same dataset it was trained on and with the predicted dataset. These test-cases were created, because not every household in a smart grid has a corresponding weather station and the usage of interpolated data is needed, which may lead to an uncertainity in the prediction.

Table 14 – Combinations of Input and Output Features

Model	Training Input	Training Output	Prediction Input	Prediction Output
Model A	WI Actual	PV Actual	WI Actual	PV Actual
Model B	WI Actual	PV Actual	WI Interpolated	PV Actual
Model C	WI Interpolated	PV Actual	WI Interpolated	PV Actual
Model D	WI Interpolated	PV Actual	WI Actual	PV Actual

Furthermore, for each model presented in Table 14, three different model were created. The three test scenarios are:

- *PVALL*: Contains all weather information and the past PV energy output and predicts the future PV energy outcome
- *PVWeather*: Contains all weather information and predicts the future PV energy outcome
- *Weather*: Contains all weather information and predicts each of them. The weather information are: Temperature, Wind Speed, Humidity, Global Radiation, Indirect Radiation, 30 and 40 Degrees Radiation

In total 36 different machine learning model were created and evaluated, 12 for each machine learning model type. Figure 30 presents the results of each of the previously explained model with random forest, as the prediction model.

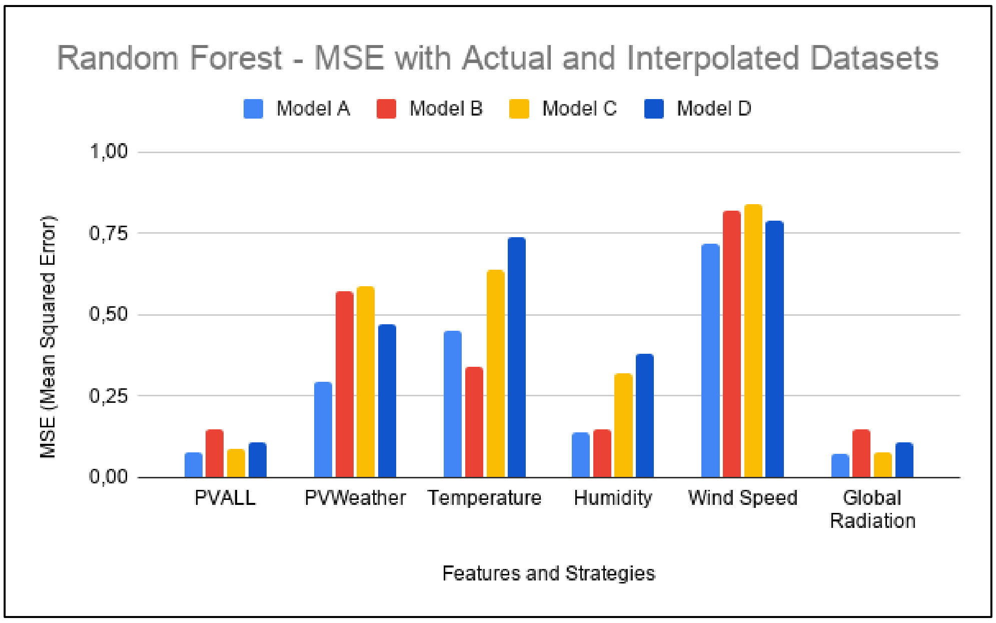


Figure 30 – Random Forest – Performance Results

The prediction of *PVALL* with random forest achieves a better MSE, than the *PVWeather* model, which only uses the weather information without the past PV energy outcome. For the prediction of the weather features the results vary from each feature. The forecast of the global radiation, humidity and the temperature were predicted mostly accurate, with the exception, that the temperature prediction with the interpolated model performed bad in this case. For both prediction types, the forecast of the wind speed turned out to perform even worse. In general, it is visible, that the prediction with the actual weather information performs better than the interpolated information, with an increasement of 16 percent of the normalized MSE.
Furthermore, the error increases in small amounts, when the i.e., interpolated dataset was used for a model which was trained on the precise weather information.
The prediction of all twelve variations for the forecast of the PV energy outcome and the weather information with Facebook Prophet are visible in Figure 31.

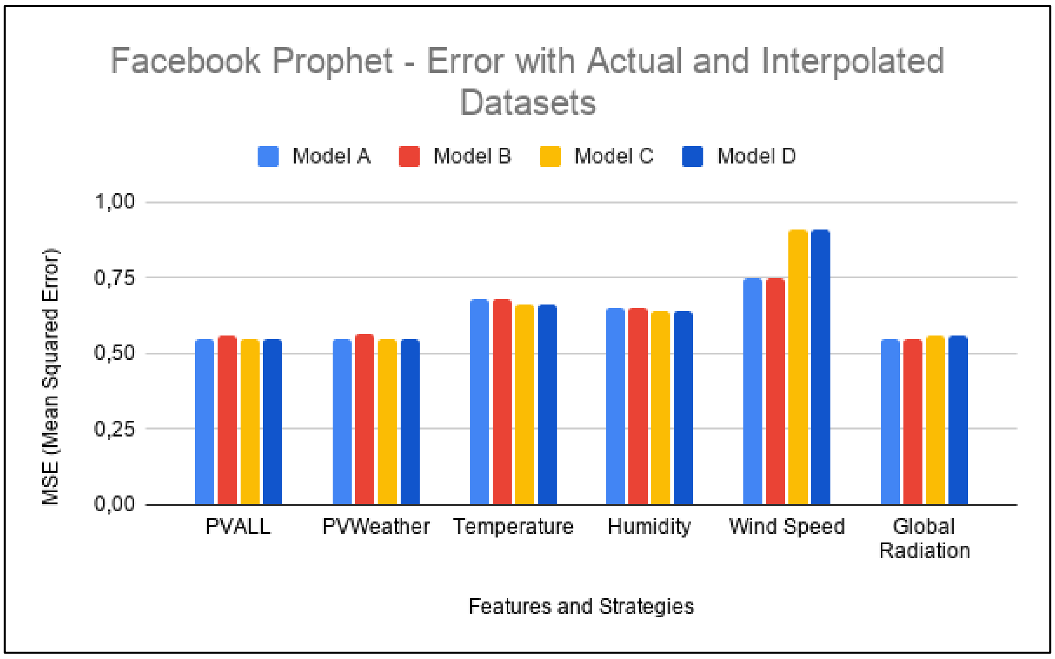


Figure 31 – Facebook Prophet Performance Results

The prediction of the Facebook Prophet machine learning model performs more stable with the interpolated data, than the random forest model. This is evident from the Figure 31, which shows the difference of the prediction on the two datasets with the Facebook Prophet model presented in chapter 4.4.2. Almost every iteration of the sample prediction leads to the same MSE.

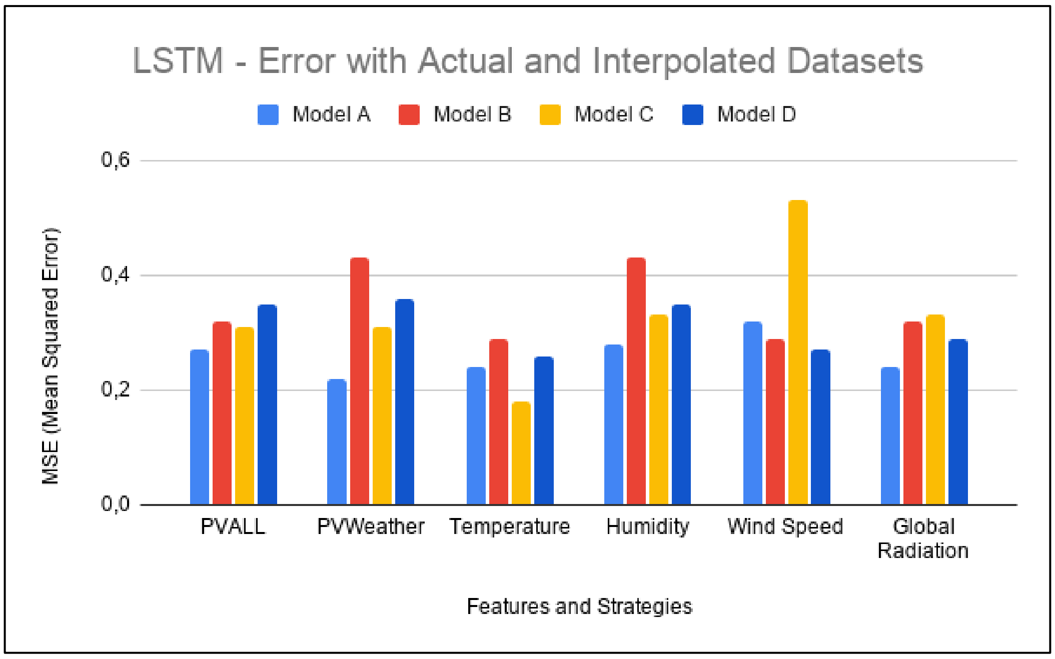


Figure 32 – LSTM Performance Results

The last model, which was also evaluated with the same test-scenarios, is the LSTM machine learning model. For these tests the, in chapter 4.4.3 presented, bi-directional LSTM model was used because this model predicted the forecasts with the lowest error between the different LSTM model. In general, the pattern of the different errors for each model look similar to the calculated errors of the random forest prediction. Except, that the errors of the LSTM models are in average higher. But the difference of the performance between the interpolated weather information and the precise ones only concluded to an increase of the normalized MSE of 14 percent, which is better than the result from the Random Forest model.

5.3.3 Effects of Seasonality

In addition to the uncertainity of forecasting renewable energy between accurate and interpolated weather information another aspect is the seasonality of the weather in general. Each season of the year has a strong impact on the weather and the corresponding energy production with renewable energy sources like PV stations. Because the datasets used for the forecasting in the previous chapter contained only monthly observations during the summer season, the LSTM model was used in this test scenario for the prediction of a dataset, which inherits data from the winter season.
More accurate, the dataset consists of the two months October and November in 2020 in an hourly basis. Figure 33 presents the results of the comparison between using seasonal data, which correspond to the training data and observations from a different season. The normalized MSE for the LSTM model increases 0.24 percent to 0.41 percent for the prediction of *PVALL* and has an even stronger deviation in the case of the prediction of the feature *humidity,* where the MSE increases from 0.24 percent to 0.58 percent.

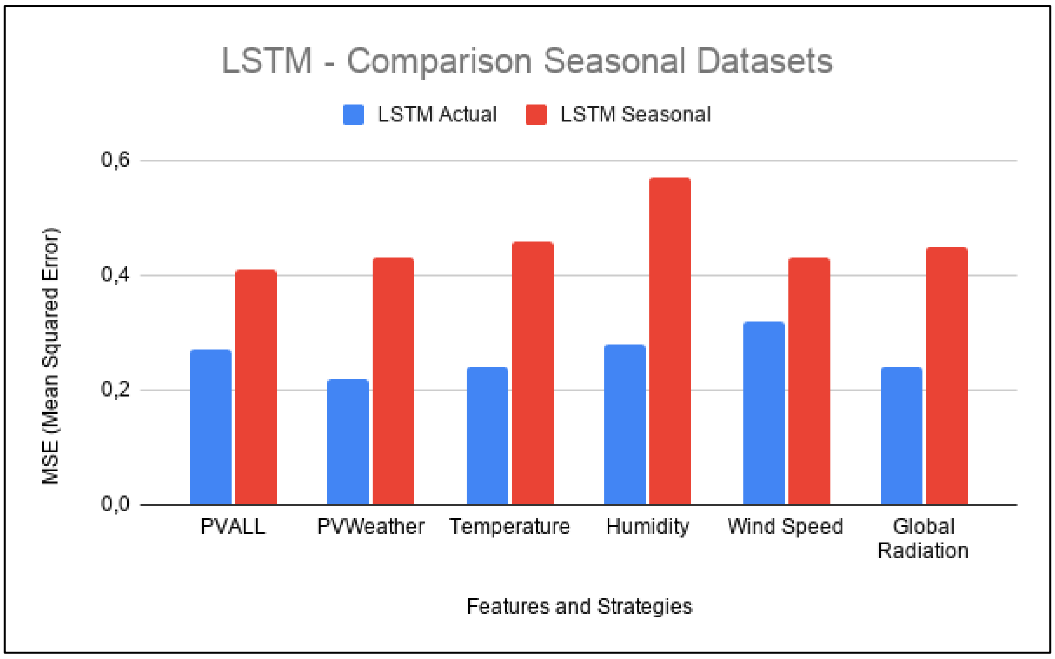


Figure 33 – Seasonality Comparison

6 Discussion

This section of the master thesis aims to discuss the studies from the previous chapter. Which introduced results from the forecasting applied on two different datasets. Furthermore, the objective fulfilment of the topic of the master thesis is another part of the discussion.

6.1 Forecasting Performance

The objective of this master thesis was the evaluation of the uncertainity of predicting renewable time series data with machine learning. To measure this uncertainity and also the performance of the machine learning model different approaches like performance metrics to measure the given error, walk-forward cross validation and the usage of residuals were used. To measure the error difference between interpolated and actual weather information 36 test scenarios were created. Those are specified in chapter 5.3.2. Furthermore, a comparison between the prediction of weather information from a specific time to another season was performed.
For the walk-forward cross validation an average MSE score of 0.55 was measured for the Facebook Prophet. For the random forest, the cross validation resulted in a score of 0.15 and for the bi-directional LSTM 0.37. The bad score of the Prophet model concludes out of the less training dataset and the non-continuous dataset used for the testing. To increase the performance of each machine learning model in general, hyperparameter tuning was performed. For the LSTM model an improvement by 20.53 percent was achieved. The tuning for the random forest model was also increased by 27.16 percent. Only the tuning for the Prophet model did not work as intended. The adjustable hyperparameter from Prophet are only for seasonality and holidays, which are not existing in the used dataset.
For the difference on the performance of interpolated and accurate weather information, the Random Forest performed 16 percent worse with the interpolated dataset. In comparison, the difference for the LSTM model was two percent less, with an increase of the normalized MSE of 14 percent.

Table 15 – Performance Comparison between Actual and Interpolated Data

Model	PVALL	PVWeather	Weather
Interpolated Random Forest + Actual Data	-51%	-52%	-9.5%
Actual Random Forest + Interpolated Data	-22%	21%	16%
Interpolated Prophet + Actual Data	-2%	0%	0%
Actual Prophet + Interpolated Data	0%	0%	0%
Interpolated LSTM + Actual Data	-16%	-49%	-22%
Actual LSTM + Interpolated Data	-12%	-16%	-28%

In addition to that, the models were cross tested with the two datasets. That means, that the trained model was i.e., trained with the actual weather information of the Glava energy centre and afterwards fed with the interpolated dataset and vice versa. Moreover, the general MSE of the different models trained with the actual and interpolated data were compared. Table 15 presents the performance comparison between the usage of the opposing dataset for the prediction. As mentioned in chapter 5.3.2 the performance decreases in general. A negative percentage in the table means, that the performance dropped by that amount and the other way around. Especially the usage of the interpolated data on a model trained on actual weather information resulted in a decrease of the performance by around 50 percent at some instances. The only model, where the usage of the other dataset did not infect the performance was the Facebook Prophet model.

The experiment of using a dataset from another season of the year also concluded in an increase of the MSE. The dataset was from the winter season, with the months October and November, which show different weather conditions, than the summer season from the training dataset. The increasement of the normalized MSE for the different seasons ranged between 20 and 50 percent. This behaviour of the seasonality could be overcome by using a machine learning model which was trained with data from the whole year or seasonality models. Furthermore, seasonality can be used or removed to increase the performance of the machine learning model. [Bro17b]

6.2 Objectives and Scope fulfilment

At the beginning of the master thesis, the objectives and scope of the work was set in chapter 1.4. For the first step a pipeline had to be implemented, which starts at the acquirement of the data and ends in a finished machine learning model with the option to evaluate the created model. The first step was the automatic call of the Glava database to receive the necessary weather and PV energy information. This data was collected via the web API of the Glava Energy Center. In the beginning the data had to be collected manually through the Metrum software. As stated in chapter 4.2, the output file was corrupted after every export. For that reason, the usage of the web API was mandatory, which also allowed an automatic download from the Glava database to the university server.

The next step was the data preparation, which is introduced in the chapter 4.3. The first step was the combining of the different data sources into one file. Afterwards the data had to be cleared from outliers and not existing values. As presented in chapter 4.3.2, the dataset had some values, which do not correlate with the rest. With the usage of winsorization, the datapoints for each feature seemed reasonable in comparison to the neighbour datapoints.

In Addition to the data preparation the different features of the dataset also had to be evaluated. The different methodologies in chapter 4.3.3. proved, that the features *precipitation, barometric pressure* and *wind direction* contain the least informational gain and for that reason, they were excluded from the dataset afterwards.
The next step of the pipeline was the creation of the three different machine learning model. As described in the previous chapter, the performance of those models was increased with the usage of hyperparameter tuning. For the implementation of the random forest and the LSTM model the frameworks *Keras*, *TensorFlow* and *Sklearn* were used, which are well established machine learning frameworks. The last machine learning model was created with the Facebook Prophet framework, which only needs the preparation of the dataset from the user. The rest of the creation of the model is handled by the framework itself.
In addition to that, also a comparative analysis of the forecasting methods was performed. The different models were compared in their performance among each other with methods described in chapter 5. Furthermore, their performance with different training and prediction input data was examined.

Research Objectives
How can the PV system energy outcome be proactively determined using machine learning model and weather information?

The first step was the framing of the problem as a supervised learning problem. That means, that the model is learning a function that maps a predefined input to an output. The model infers from labelled training data. The input data consists of a sequence of previous time series observations and the output represents the next x timesteps. With this structure it is possible to preserve the structure and order of the data, which is needed for time series data.
For the prediction of the PV energy outcome, not only the past energy outcome is a good indicator for future outcome, but also the current weather information. For that reason, also a good forecast of the weather information and the usage of those information for the prediction of the PV system energy outcome is needed.

Do interpolated weather information affect the future prediction of PV energy outcome?

One objective of this master thesis was the comparison of the usage of weather information, which were accumulated directly at the PV station and interpolated ones. For that reason, each of the three used machine learning model were once created and tested with the accurate dataset and a second pair with the interpolated dataset. Furthermore, the difference in the performance was also measured with three different types of input and output. The different types are presented with Table 14. The results of those tests came to the conclusion, that in general, the usage of an interpolated dataset effects the performance. In addition to that the tests yielded, that the usage of interpolated data for the prediction on a model, which was trained with the accurate dataset, also the performance drops.

7 Conclusion

In resemblance of the objectives from chapter one, a study on the forecasting of PV systems energy outcome was undertaken. The studies on this machine learning task are part of the forecast of time series data, which have some special requirements to the machine learning model. Furthermore, different data preparation techniques like the winsorization or normalization were used to prepare the dataset for the next tasks. These methodologies were especially needed for the Glava dataset, because these data structures inherit some outliers, which could result into a worse performance in general.

The dataset of the weather information consists of many different features, which had to be evaluated in the regard of the importance to the outcome of the prediction. Each of the features was compared to their specific informational gain for the prediction of the PV system energy outcome. In general, only seven out of the eleven available features from the Glava dataset were chosen, because the others only had a low effect on the prediction. For the determination of these features different techniques were used, which are presented in chapter 3.3.

For the prediction of the future PV system energy outcome three different techniques were developed, as presented in chapter 4.4. Each of them was trained on the same dataset of weather and PV information, except the Facebook Prophet model, because it only performance well with continuous datasets. The LSTM and also the random forest models experienced a performance boost with the usage of hyperparameter tuning with a decreasement of the MSE of 20.53 percent for the LSTM model and 27.16 percent for the random forest.

The evaluation results presented in chapter 5 presents, that in total the random forest model outperforms the other two machine learning model with an average MSE of 0.15. In comparison the bi-direction LSTM model achieved a MSE score of 0.37 and the Facebook Prophet model 0.58. But in general, the Facebook Prophet model could achieve a better prediction with more continuous data for the training. Furthermore, the results of the different input data show, that the prediction results are strongly affected by the accuracy of the weather information, which makes the prediction with interpolated data more uncertain. In addition to that uncertainity, the different seasons of the weather also play a big part in the uncertainity. As presented in the Figure 33, a model trained with yearly data for the summer had problems with the prediction using weather information from the winter season.

7.1 Future Work

For the extension of the work presented in this master thesis, the first step could be the usage of different inputs for the weather information. Because some of the weather information from the Glava dataset had many errors within it, which had to be cleaned. Furthermore, different ranges for the interpolated data could be worth to be invested. For the scope of this thesis, the interpolated data was taken from a weather station at the Karlstad centre, which is a distance of around 70 kilometres.

In addition to that, different machine learning model could be trained with a longer continuous time range, to fetch the seasonal effects of the weather to the PV energy outcome. The other option could be the training of several seasonal models.

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my research professor Dr. Andreas Kassler and Dr. Andreas Theocharis, for giving me the opportunity to do the research for this master thesis and providing me with invaluable guidance throughout this research. It was a great privilege and honour to work and study under your guidance. I would also like to thank you for your friendship, empathy and great sense of humour.
Furthermore, I would like to thank my project partner Viviana Raffa for all the great hours together in CARL.

VI Bibliography

[@Ch18] A. Choudhary: Generate Quick and Accurate Time Series Forecast using Facebook Prophet. https://www.analyticsvidhya.com/blog/2018/05/generate-accurate-forecasts-facebook-prophet-python-r/, last viewed: 25.11.2020.

[@Col15] Colah’s Blog: Understanding LSTM Networks. https://colah.github.io /posts/2015-08-Understanding-LSTMs/, last viewed: 20.12.2020.

[@Cou20] Countants: How Machine Learning Can Enable Anomaly Detection. https://www.countants.com/blogs/how-machine-learning-can-enable -anomaly-detection/. last viewed: 22.11.2020.

[@Gct18] Georgia College of Tech Computing: Lecture 16: Feature Selection: https://www.cc.gatech.edu/~bboots3/CS4641-Fall2018/Lecture16/16_Feature Selection.pdf, last viewed: 20.12.2020.

[@Jup20] Jupyter: Jupyter Notebook. https://jupyter.org, last viewed: 20.12.2020.

[@Kau16] S. Kaushik: Introduction to Feature Selection methods with an example. https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature -selection-methods-with-an-example-or-how-to-select-the-right-variables/. last viewed: 22.11.2020.

[@Ker20] Keras: Optimizers. https://keras.io/api/optimizers/, last viewed: 04.01.2020.

[@Pro20] Facebook Prophet: Diagnostics. https://facebook.github.io/prophet/ docs/ diagnostics.html, last viewed: 04.01.2020.

[@Seb20] Sebastian Ruder: An overview of gradient descent optimization algorithms. https://ruder.io/optimizing-gradient-descent/index.html, last viewed: 04.01.2020.

[@Sin19] Deeprika Singh: Cleaning of Outliers. https://www.pluralsight.com/guides /cleaning-up-data-from-outliers, last viewed: 20.12.2020.

[@Sla16] L. Sullivan, W. LaMorte: InterQuartile Range. https://sphweb.bumc.bu.edu/otlt/ mph-modules/bs/bs704_summarizingdata/bs704_summarizingdata7.html, last viewed: 22.11.2020.

[@Sta17] A. McQuistan: Using Machine Learning to predict weather. https://stackabuse .com/using-machine-learning-to-predict-the-weather-part-2/, last viewed: 02.01.2020.

[@Swe20] Sweden: Energy Use in Sweden. https://sweden.se/nature/energy-use-in Sweden, Last viewed: 14.01.2020.

[AA13] Allende-Cid H., Allende H.: Wind Speed Forecast under a Distributed Learning Approach. 2013 32nd International Conference of the Chilean Computer Science Society (SCCC), Temuco, 2013, pp. 44-48, doi: 10.1109/SCCC.2013.24.

[AB17] Andrade J., Bessa R.: Improving renewable energy forecasting with a grid of Numeral weather predictions. IEEE Transactions on Sustainable Energy, 8 (4):1571-1580, 2017.

[AO16] Antonanzas J., Osorio N., et. al.: Review of photovoltaic power forecasting. Solar Energy, 136:78–111, October 2016.

[ASM20] R. Ahmet, V. Sreeram, Y. Mishra et. Al.: A review and evaluation of the state Of The-art in PV solar power forecasting: Techniques and optimization. Re-Newable And Sustainable Energy Reviews, Volume 124, 2020.

[BA09] N. Bajpai: Business Statistics. Pearson Education India, India, 2009.

[BFO18] S. Bouktif, A. Fiaz, A. Ouni, et. Al.: Optimal Deep Learning LSTM Model for Electric Load Forecasting using Feature Selection and Genetic Algorithm. Energies 2018, 11, 2018.

[BRO17a] J. Brownlee: Long Short-Term Memory Networks with Python. Machine Learning Mastery, 2017.

[BRO17b] J. Brownlee: Introduction to Time Series Forecasting with Python. Machine Learning Mastery, 2017.

[BRO20] J. Brownlee: Data Preparation for Machine Learning: Data Cleaning, Feature Selection and Data Transforms in Python. Machine Learning Mastery, 2020.

[CC10] D. Cousineau, S. Chartier: Outliers detection and treatment: a review. International Journal of Psychological Research, pp. 2011-2084, Columbia, 2010.

[CD11] Chen C., Duan S., et. al.: Online 24-h solar power forecasting based on weather Type classification using artificial neural network. Solar Energy, 85(11):2856 - 2870, 2011.

[DA16] Davo F., Alessandrini S., et. al.: Post-processing techniques and principal Components analysis for regional wind power and solar irradiance fore-Casting. Solar Energy, 134:327 – 338, 2016.

[GLF09] A. Graves, M. Liwicki, S. Fernandez, et. Al.: A Novel Connectionist System for Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 31, no. 5, pp 855-868, 2009.

[GS99] Guralnik V., Srivastava J.: Event Detection from Time Series Data. ACM, University of Minnesota, 1999.

[HA18] R. Hyndman, G. Athanasopoulos: Forecasting: Principles and Practice. OTexts Australia, 2018.

[HS90] A. Harvey, S. Peters: Estimation Procedures for Structural Time Series Models. Journal of Forecasting, Vol. 09, pp 89 – 108, 1990.

[HS97] S. Hochreiter, J. Schmidhuber: LSTM can solve hard long time lag problems. Neural Computation, Volume 9, Issue 8, 1997.

[IC18] Isaksson E., Conde M.: Solar Power Forecasting with Machine Learning Techniques. School of Engineering Sciences, Stockholm, Sweden, 2018.

[IK18] E. Isaksson, M. Karpe: Solar Power Forecasting with Machine Learning Tech-Niques. Stockholm, School of Engineering, 2018.

[IP13] Inman R., Pedro H., et. al.: Solar forecasting methods for renewable energy integration. Progress in Energy and Combustion Science, 39(6):535 – 576, 2013.

[IR18] IRENA: Renewable power generation costs in 2017. Technical report, International Renewable Energy Agency, Abu Dhabi, 2018.

[K03] Kim K.: Financial time series forecasting using support vector machines. Neurocomputing 55 (2003) 307-319, South Korea, 2003.

[KC19] Khandakar A., Chowhury M., et. al.: Machine Learning Based PV Power Prediction Using Different Environmental Parameters of Qatar. Energies 2019, 12, 2782, Qatar, 2019.

[KM17] J. Konecny, H. McMahan, et. al.: Federated Learning: Strategies for Improv Ing, Communication Efficiency. 2017.

[KP11] Kostylev V., Pavlovski A., et al.: Solar power forecasting performance–towards industry standards. In 1st national workshop on the integration of solar power into power systems, Aarhus, Denmark, 2011.

[KU18] P. Kumar: Forecasting Cloud Resource Utilization Using Time Series Methods. KTH Royal Institute of Technology, Sweden, 2018.

[MAB20] M. Moocarme, M. Abdolahnejad, R. Bhagwat: The Deep Learning with Keras Workshop. Packt Publishing Ltd, 2020.

[MS18] Makridakis S., Spiliotis E. et. al.: Statistical and Machine Learning forecasting Methods: Concerns and ways forward, en. Ln: PLOS ONE 13.3, Greece, 2018.

[NG06] C. Nachmias, A. Guerrero: Social Statistics for a diverse society. Pine Forge Press, 2006.

[PC09] Pan F., Converse T., et. al.: Feature selection for ranking using boosted trees. In Proceedings of the 18th ACM conference on Information and knowledge management, pages 2025-2028. ACM, 2009.

[PC12] Pedro H., Coimbra C.: Assessment of forecasting techniques for solar power Production with no exogenous inputs, Elsevier, USA, 2012.

[PK13] Pelland S., Kleissl J., et. al.: Photovoltaic and Solar Forecasting: State of the

Art, IEA PVPS 14, Canada, 2013.

[R09] Reikard G.: Predicting solar radiation at high resolutions: A comparison of Time Series forecasts. Solar Energy, 83(3):342 – 349, 2009.

[SL12] Shi J., Lee W., et. al.: Forecasting power output of photovoltaic systems based On weather classification and support vector machines. IEEE Transactions on Industry Applications, 48(3):1064-1069, 2012.

[SS17] M. Spiegel, L. Stephens: Schaums Outline of Statistics, McGraw-Hill Education, 2017.

[TK18] Theocharides S., Kyprianou A. et. al.: Machine Learning Algorithms for Pho to-voltaic System Power Output Prediction, Conference Paper, IEEE, 1678 Cyprus, 2018.

[TL17] J. Taylor, B. Letham: Forecasting at Scale. PeerJ Preprints, Open Access, 2017.

[TSB07] A. Tabibi, N. Simforoosh, A. Basiri, et. al.: Bowel Transformation Versus no Preparation Before Ileal Urinary Diversion. Urology, Volume 70, pp. 654-658, 2007.

[W95] Wilks D.: Statistical Methods in the Atmospheric Sciences, Academic Press, Volume 59, 1995.

[WIC17] R. Wicklin: Winsorization: The good, the bad and the ugly. SAS, 2017.

Declaration

I (Phil Aupke) hereby declare in lieu of an oath that I have written this thesis independently and without the use of any aids other than those indicated; any ideas taken directly or indirectly from outside sources are marked as such. The thesis has not yet been submitted to any other examination authority in the same or a similar form, nor has it been published.

Place, Date

Signature