Advanced Project II

Live Forecasting Framework for Individual PV Plants

Author:
Mark C Koerner

Mentor:
Nicolas Sommer

Instructor:
Dr. Stefan Kettemann

January 5, 2020

JACOBS
UNIVERSITY

Executive Summary

Next Kraftwerke is a growing energy company with a primary focus on bundling their customer's energy as part of a Virtual Power Plant and marketing it on the European power exchanges. One requirement for marketing the customer's future energy generation is knowing the quantity to be produced, which is easy in case of traditional energy sources such as coal but difficult for renewable sources such as wind and solar power. Prior to this project, Next was using forecasts based only on weather forecast data for the surrounding area and not making use of the live infeed data it received from the plants as well.

The purpose of this project was to develop a framework for combining the live infeed data with the weather-based forecasts and potentially other factors in order to develop better live forecasting capabilities. Due to the large number of photovoltaic power plants and large differences between individual plants, the project had a number of requirements for this modelling framework. Namely, the algorithm had to be fast in terms of calculation time, stable in terms of performance for at least one hour into the future, safe in terms of outliers and independent from a specific set of variables, meaning for example that the number and kind of weather-based forecast could vary from plant to plant.

As a general outline of the process, I first developed a testing and evaluation framework as well as a general scope of potential modelling approaches, then visualized the available data, preprocessed it and finally tested out the different modelling approaches.

In terms of the testing and evaluation framework, both a train-test split and walk-forward validation were used in model selection. The error metrics used for predictions were both RMSE and MAE, which were grouped by the time step relative to the initial prediction for each instance. All modelling rounds had a forecast horizon of ten 15-minute periods into the future.

The framework was developed on data from several Dutch and German plants in separate files. Most files contained up to 40,000 rows of values of the infeed, weather-based forecasts and the Clear Sky Index variables, the latter of which represents the output of the plant under optimal conditions. After cleaning the data, the final modelling approach uses the Clear Sky Index to transform the infeed and other forecasts in order to mitigate seasonal patterns in the data.

The modelling process can be broken down into multiple stages. First, over ten algorithms were broadly tested in terms of general feasibility. Then, implementations of Support Vector Machines, Ridge Regression, Random Forests and XGBoost Regression were further optimized and tested using different forecasting approaches. Finally, XGBoost emerged as the algorithm with the best fit in terms of the initial modelling requirements. In order to find the optimal configurations, a grid search was performed for model parameters and the algorithm was further tested in terms of required training set size and refitting frequency. Although the overall approach could still be improved in some areas, the final framework and algorithm fulfilled all initial requirements and can easily be adapted for live implementation.

Contents

1 Introduction

The purpose of this report is to document the data analytics project I completed for Next Kraftwerke this past summer. Next Kraftwerke is a growing company in the energy sector, primarily focused on operating a Virtual Power Plant and trading on the different European power exchanges. Contrary to my first experience working at the company, when I was tasked with grid-wide wind power prediction for Germany, my latest task was concerned with predicting photovoltaic plant power output on an individual plant level. Next already produces a few individual forecasts based on local weather data, so the task was to combine those forecasts with live infeed data of the plant itself in order to create a prediction for the next couple of hours in quarter-hour frequency. Quarter-hour frequency is used as it is the time interval used for power trading at most Intraday exchanges. As Next operates in multiple countries and regions, however, the availability of these forecasts differs based on the availability of the data, meaning the framework cannot be entirely dependent on individual weather-based forecasts.

In total, Next Kraftwerke owns the rights to the energy output of around 2000 photovoltaic plants in several European countries and at various sizes, so the project was bound to several requirements. The forecasting framework had to be:

fast	low calculation time
stable	good performance starting at t + 1 hour
safe	no unrealistic movements or values in the predictions
independent	not bound to specific variables; reusable for varying parameter sets

As a first step, a general framework for testing and evaluation was created, which was then used to explore different modelling approaches. After choosing the best approach and finding the right model parameters, the best model was further evaluated in terms of performance and implementation details. As a final deliverable, I submitted a Python script and accompanying Jupyter notebook detailing the modelling procedure and containing a modelling class ready for implementation.

2 Data Description

Next Kraftwerke provided me with several different data files throughout the project. These files included data for a single PV plant and combined data for three plants in the Netherlands, and data for seven PV plants all throughout Germany. Towards the end of the project, I was provided with data for six more German plants. These files were all structured in a nearly identical way.

The first column featured timestamps in 15 minute intervals, and the following columns contained the plant's aggregated energy output for the specific interval (infeed), as well as the corresponding predictions using weather data for the plant's location and the Clear Sky Index for the plant. The Clear Sky Index is an estimate of the maximum possible output for the plant given the its characteristics, which was previously developed internally. Most output numbers are believed to be in kWh, with only the three combined Netherlands-based plants file containing MWh values. In terms of the time frame, most of the files included data starting from January 1st, 2018 up to May 2019, with some files having a slightly shorter period of available data. This means that the data for most plants consisted of 4-6 columns depending on the amount of available forecasts and a little over 40.000 rows.

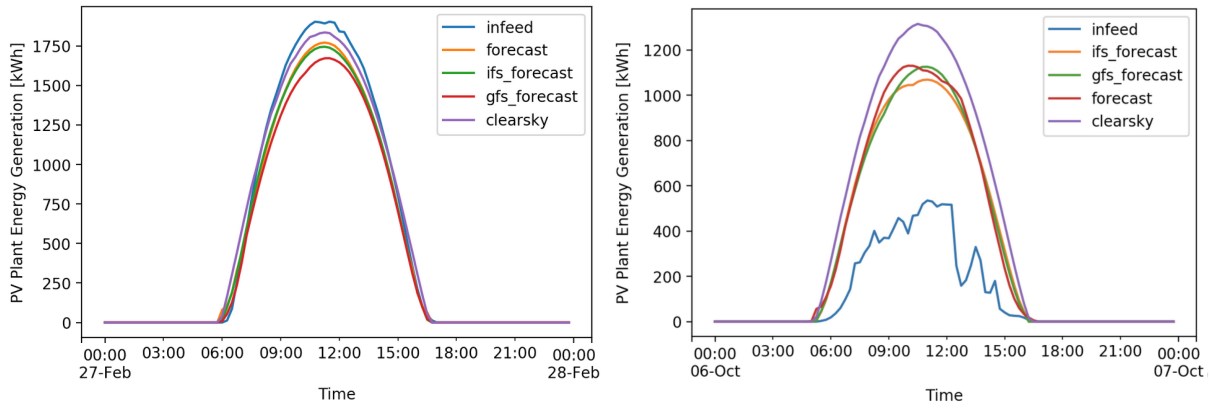
Although there exist weather-based predictions for each plant's output, there are a couple issues which motivated the search for an overall more accurate model. Since there are no weather sensors directly at the plant, the output predictions are based on weather data that is available for the surrounding area in different resolutions, allowing for multiple predictions of the plant output. Another factor that leads to further inaccuracy of the output predictions is the frequency of these forecasts. The weather forecasts are only available for a small number of key hours throughout the day, for example 9:00, 12:00 and 15:00, meaning that the output prediction values in 15 minute frequency have to be interpolated arbitrarily.

3 Data Exploration

After parsing the data, the first step was to gain a deeper understanding of it. Since PV plants follow a daily cycle, I decided to graph all variables for a number of different days and plants. Figure 1 shows two days from different plants which illustrate several key issues in the dataset.

When looking closely at 1a, it becomes obvious that the infeed values are actually larger than the Clear Sky Index for a large portion of the day, which should theoretically never be the case. In other words, the plant seems to either be outperforming the hypothesized maximum output given optimal conditions, which could be due to a mistake in the plant parameters the Clear Sky Index is based on or the method for calculating the Clear Sky Index being imprecise. Regardless of the cause, it seems that the Clear Sky Index is sometimes inaccurate, which should be kept in mind when processing and modelling the data.

Figure 1b illustrates another issue that is present in the dataset. In some cases, the weather-based forecasts are very inaccurate. In addition, they are also unpredictably inaccurate, meaning that the nature of their errors does not follow a set pattern. Thus, both clear sky and forecast values should be used with caution. Sadly, there was no way for me to validate the infeed data with the given dataset.

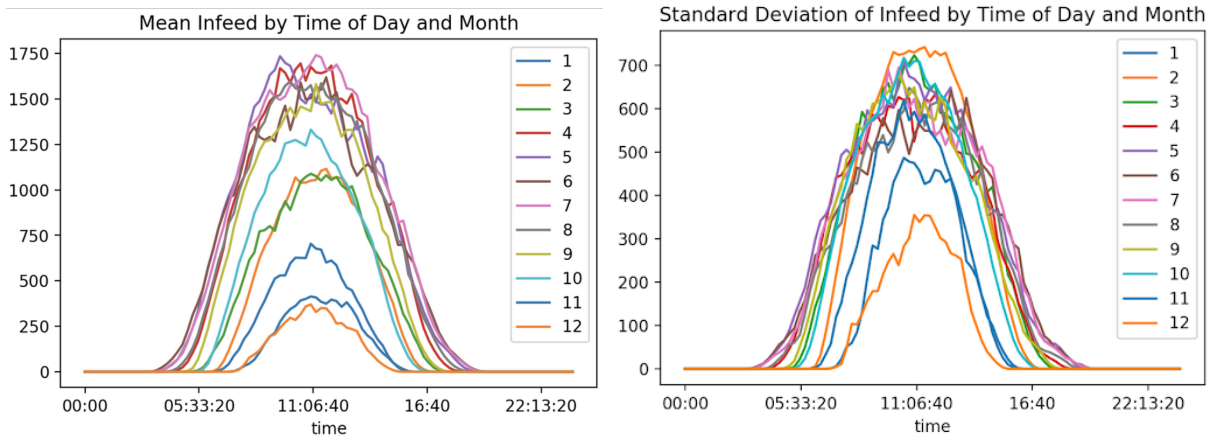


(a) Lineplot of raw data for February 27.

(b) Lineplot of raw data for October 6.

Figure 1: Infeed, weather-based forecasts and Clear Sky Index for two German PV plants for the days of Feb 27 and Oct 6, respectively.

Apart from these issues, the graphs also show the daily pattern of PV plant energy production. Generally, energy production is highest in the middle of the day and falls off strongly towards the beginning and end of sunlight hours. Thus, the production of PV plants should also follow a year-long seasonal pattern due to the changing sunlight hours and angle of the sun towards the PV modules [8]. Figure 2 gives an overview on how different the production and variance in production can be on average throughout the day for every month of the year. It is interesting to observe that while mean production is lower for all winter months, the standard deviation is only significantly lower for January and December, whereas February shows the highest standard deviation of any month during peak hours. Thus, it seems that while production is lowest in the winter months, the relative variance seems to be higher.



(a) Average Infeed grouped by Month and Time of Day.

(b) Standard Deviation of the Infeed grouped by Month and Time of Day.

Figure 2: Note: The line plots for January and December are the lower ones compared to November and February on each graph.

Figure 3 explores this concept a little further. Most of the summer months seem to have the same relative standard deviation as compared to the mean production hovering around a ratio of 0.5, whereas some of the colder month even have a higher standard deviation of production values than mean production values. Specifically, January and December consistently have the highest ratios throughout the day, showing that the average production for those months is not only the lowest but also the most variant relative to their production. When thinking forward towards the modelling stage, it seems that it would be difficult to use production data from June and July for example to train a model to forecast December production, due to the seeming difference in terms of overall value and volatility. It also seems that forecasting the winter months could potentially be the most challenging time to forecast again due to the high variance in output.

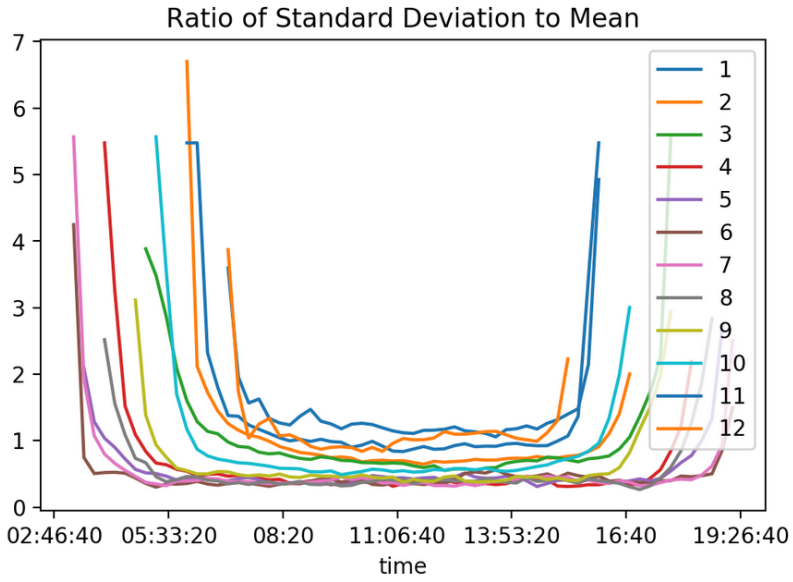


Figure 3: Ratio of Standard Deviation to Mean grouped by Month and Time of Day. Note: January and December are represented by the higher line compared to November and February.

In addition to the relative differences in average monthly production, Figure 3 also illustrates another important concept. Throughout the time of day, the relative standard deviation to the mean output spikes for the morning and evening hours. Part of the reason is the low average production during those times, yet it also shows that those hours of the day could be very hard to forecast with very limited reward. This constitutes another modelling challenge which will be addressed later on in the report.

Lastly, I decided to take a look at the infeed data from a traditional time series analysis point of view. One of the initially proposed modelling technique was the ARIMA method, which is usually preceded with a look at the autocorrelation function (ACF) of the time series in question [5]. The ACF calculates the correlation between the original time series and a lagged version of itself. For example, lag 7 means comparing every data point with the data point 7 periods prior. In the case of a 15 minute frequency, that is 1h45 prior. In addition to the ACF, there is also the partial ACF, or PACF, which factors out the autocorrelation of the previous lags [7].

The ACF and PACF plots for one German plant can be seen in Figure 4. Usually, one consults these plots to figure out the Auto-Regressive and Moving Average terms of the ARIMA model [6]. Figures 4a and 4b show the regular ACF and PACF for a sample German plant. The blue bands signify confidence intervals for significance. The ACF plot displays high autocorrelation overall and also a seasonal pattern with a local maximum around Lag 96. Lag 96 is significant because it represents the number of 15 minute intervals in a day and thus confirms the daily pattern again, but therefore also unsurprising.

Figures 4c and 4d show the ACF and PACF after regular and seasonal differencing. Differencing is a time series technique that takes the difference of the actual time series and a lagged version of itself. Thus, it shows the increment at every timestep, which is useful when modelling time series that show a pattern of significant autocorrelation values for a large number of lags, such as in the top left graph. Seasonality describes a phenomenon when the time series shows a recurrent pattern of the same length, just like the daily pattern seen in Figure 1. The seasonal difference is a simple difference with a lagged time series of the seasonal length, in this case 96. The graphs show a spike at around 96 for the ACF plot, and at multiples of 96 for the PACF plot, meaning these terms could be relevant parameters for a seasonal ARIMA model. Aside from ARIMA models, I also used these insights as a starting point for parameter creation for traditional Machine Learning algorithms, which will be discussed further in Section 5.

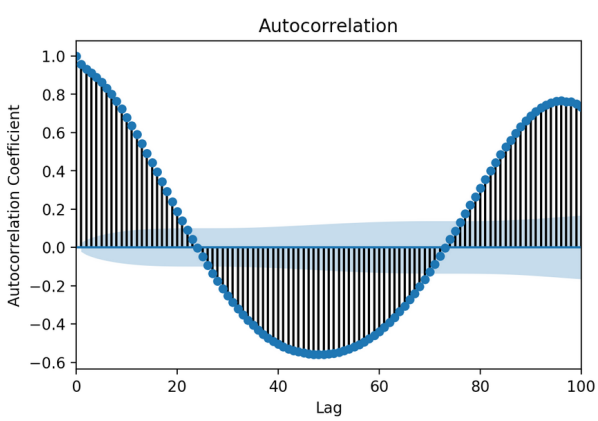
4 Modelling Framework

Before the modelling step, I was tasked with developing a framework on how to approach, test and evaluate different models. The testing and evaluation framework were to be developed first, with the modelling approach to be determined thereafter.

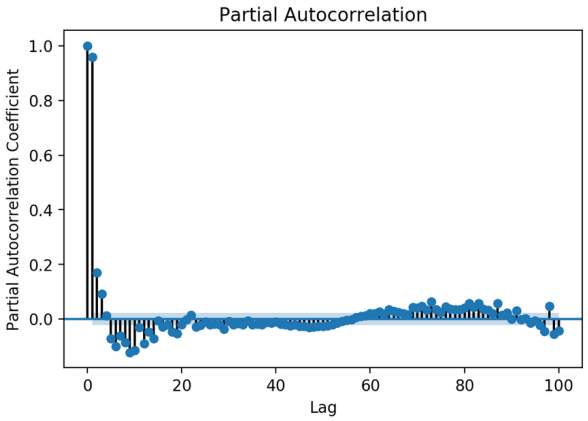
4.1 Testing and Evaluation Framework

The goal of the modelling task was to configure an algorithm to be able to predict output for the next hour and beyond from the latest output value, previous output values and weather-based predictions for the relevant timestamp. With this in mind, I decided to split the last 20% of the data off to be used as the testing set, meaning that all models were to be trained on 80% of the data and then tested on the rest. This resulted in roughly one year of training data with the models to be tested on the data for the beginning of 2019, which included months indicated to include very volatile energy output in Section 3 and should thus be innately difficult to forecast. The initial exploration and model search was performed on a single plant for efficiency reasons but was expanded to multiple plants after narrowing it down sufficiently.

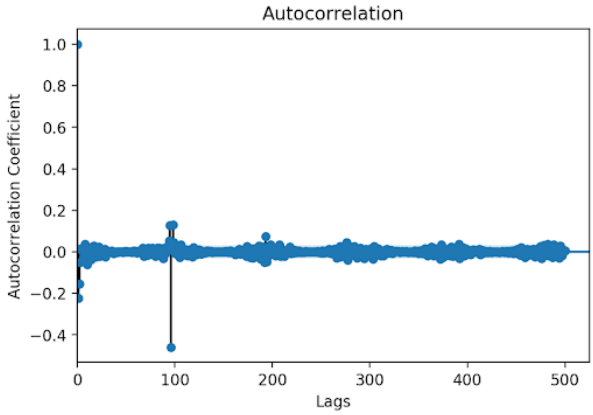
In terms of evaluating performance, I decided to look at errors based on time difference from the last output value. For example, I labelled the predicted value for the first quarter-hour



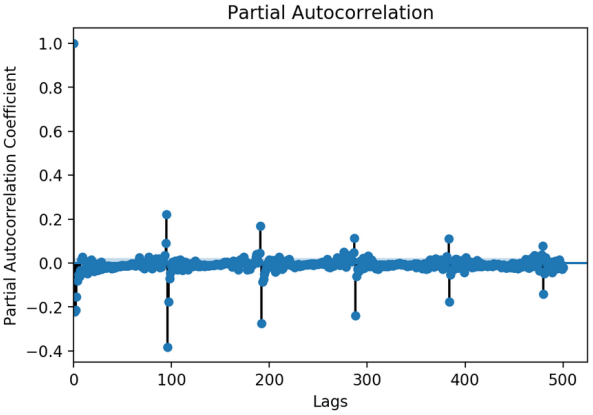
(a) ACF plot of raw infeed.



(b) PACF plot of raw infeed.



(c) ACF plot after regular and seasonal difference of length 96.



(d) PACF plot after regular and seasonal difference of length 96.

Figure 4: ACF and PACF plots for German PV plant output before and after differencing.

t+0, and evaluated models based on their t+0 performance for the entire testing set as well as t+1, t+2 etc. This was important because of the goal of stable performance 1 hour after the last energy output value (or t+3), but also necessary because each model was asked to produce predictions for multiple timesteps into the future at every timestamp of the testing set, meaning that every timestamp also had multiple predictions associated with them from various starting points. Grouping these prediction and their errors by relative distance from the original timestamp was more convenient to evaluate model performance based on said distance.

In terms of the actual error metric, the initial goal was to use normed errors such as MAPE or normed RMSE which is comprised as a percentage of the true value. Similar to the high relative Standard Deviation in the mornings in Figure 3, however, these errors were imprecise due to small morning values, which would give normed errors in the ten thousands or higher and greatly skew the average errors. Thus, I decided to stick with the regular RMSE and MAE for the entirety of the project, and used both to guide my model selection [4].

Lastly, after narrowing down the model search I also used the expanding and rolling window approach for model evaluation for all plants [2]. The expanding window approach takes a certain amount of data, trains the model and then evaluates on a subset of a given length. Then, it adds the window of data just used as the testing set to the training set and tests the model on the next subset until the end of the dataset. The rolling window approach simply creates another window for the subset of data on which the model is to be trained. For example, one might always trains the model on two weeks worth of data and evaluates on one for the entire length of the dataset. Both were used to test the stability of the final model and to find the optimal configuration of the chosen model based on a number of different factors.

4.2 Modelling Scope

As a next step, I was tasked with compiling a list of all possible approaches, techniques and other configurations. The table shown in Figure 5 shows all possible combinations of preprocessing steps, parameters, modelling techniques and time series modelling approaches. It is important to note that Machine Learning algorithms that are not time-series-specific need the manual creation of parameters such as previous values (Lags) when applied to time series data. The parameter creating and preprocessing steps will be discussed in Section 5 and the specific models will be discussed in Section 6.

Preprocessing Steps	Parameters	Model	Modelling Approach
Differencing	Lags (1,2,3..., 96*x)	Linear Regression	Recursive (use prediction as input for next timestep)
CSI Norm	Other Forecasts	Ridge Regression	Multiple models (one for each t+x)
Smoothing (Moving Avg, Kalman Filter?)	Quarter-hour Dummy Var.	SVM	Multiple outputs (from one model)
Different sizes of train sets?	Infeed/Forecast Ratio	Random Forest	Time Series Forecast then combine with other Forecasts
Relevant time indices (sunrise + x, sundown - x)	MA Terms	XGBoost	
		LSTM (or MLP/any type of NN)	
		(S)ARIMA	
		Gaussian Processes	
		Naive/Simple Techniques	

Figure 5: Table with all possible combinations of preprocessing steps, parameters, algorithms and modelling approaches.

The modelling approaches all attempt to solve the problem of forecasting multiple periods into the future. The time series method is fairly straight-forward, as it would use a seasonal ARIMA model to predict the plant's output from its previous out and then use a different algorithm to combine it with the weather-based forecasts. Most traditional time series methods have the innate ability to forecast multiple periods into the future.

The recursive approach is the approach traditional time series methods use to extent their predictions for multiple periods. After the initial forecast, the model simply takes the prediction as the latest true value and re-applies the algorithm for the next period. Theoretically, this means that the imprecision of the first prediction becomes part of the next prediction, meaning that logically the more predicted values that are included in making the next prediction, the more imprecise said prediction becomes. In other words, it amplifies the errors of the previous prediction. In addition, the algorithm is only trained on true values and not predictions, which could lead to some systematic errors. The multiple models approach attempts to fix this issue by fitting a separate model for each timestep removed from the initial prediction. In theory, the model at each timestep could work to mitigate the previous errors at the cost of computational efficiency. Lastly, some algorithms (e.g. Random Forests) have an innate ability to produce multiple outputs based on the initial input features which has the potential to be even more effective depending on the specific implementation [1].

5 Data Preprocessing

Prior to the modelling process, I had to transform the data into the right format. In this case, this process includes cleaning the data as well as potentially applying certain transformations and creating more features from the raw data.

5.1 Data Cleaning

The first step was to clean the raw data and check for inconsistencies. Aside from delimiter inconsistencies between files which initially lead to faulty values after parsing, there were no major inconsistencies or outliers in the actual values. The only inconsistencies were related to PV plant output values at night, where they should be equal to 0. In some cases, these values were small negative integers, which is likely due to measurement error. I set all negative values to 0. In addition, Next does not save the night values which are equal to 0 in order to save space. This leads to missing timestamps in the dataset. Even though the values are largely irrelevant, I wanted to create a continuous time index which would make it easier to carry out some of the other preprocessing steps such as differencing or feature engineering. In order to create this index, I expanded the timestamps to include all possible timestamps in 15 minute intervals between the first and last data point in the file, and inserted 0 for all values in the newly created rows.

5.2 Transformations

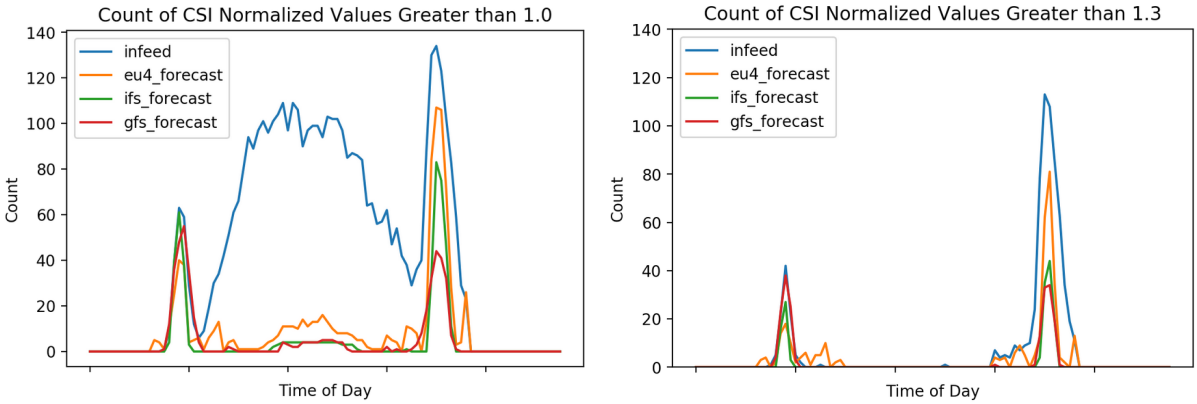
The next step in the preprocessing pipeline constitutes applying transformations to the data. In some cases such as LSTMs, transformations were necessary because the input needed to be normalized to [0,1]. For the most part, however, these transformations sought to address the challenges of the seasonality as well as the intra-day volatility of the plant's energy output. Seasonality might not seem like much of an issue in theory, but initial modelling attempts without prior transformations of the data showed systematic errors mostly due to the daily pattern of values after which we decided to address it.

Initially, I attempted to address the seasonality issue by using regular and seasonal differencing as described in Section 3. The drawbacks of differencing, however, are that it invalidates a small part of the dataset for the initial differencing and it adds complexity when transforming the predictions back in the end, especially in the case of both regular and seasonal differences.

The second, and much more effective approach for reducing seasonality was what we called Clear Sky Index (CSI) Norm. Essentially, the transformation simply divides the relevant time series by the respective CSI series, which would theoretically result in values between [0,1] and mimic the fraction of the maximum potential output that is being produced. In practice, it did work to reduce the seasonality of the energy output to an extent but it also produced outliers and

infinity values for hours where the CSI was 0 but the energy output was not. Since the infinity values were all in morning and evening time intervals adjacent to 0 output values, I decided to replace them with 0 values themselves as the actual input was negligible to begin with.

The other outliers were all CSI normed values above 1, which should theoretically be impossible. Figure 6 shows the total number of outliers with values above 1.0 and 1.3 for each quarter hour of the day for the 423 days worth of data. The raw values for several quarter-hour intervals each day exceed the corresponding raw CSI value for close to a quarter of the entire dataset. While there are a lot of outliers during the day, the more extreme outliers seem to happen in the morning periods for the same reasons as previously discussed in the Data Exploration section. For the remainder of the project, I decided to ignore the outliers in the middle of the day and instead focus on removing the outliers in the morning and evening hours. After some testing I decided to set all morning and evening outliers to 0 similar to the infinity values, as long as the normed value was above 1.3. In order to filter out the actual morning and evening values, I only replaced the values if the original CSI value was lesser or equal to 25% of the maximum CSI value for that day. This cleaning procedure resulted in no more CSI normalized values above 1.3 in the dataset. Both of these seasonality transformations were applied in conjunction with several of the algorithms, which will be discussed further in Section 6.



(a) Count of CSI normalized values above 1.0 grouped by time of day.

(b) Count of CSI normalized values above 1.3 grouped by time of day.

Figure 6: Histograms showing the frequency of outliers of CSI normalized values at different cutoffs grouped by time of day.

As for the volatility challenge, the discussed and attempted transformation included Moving Average and Exponential Smoothing methods. Both methods shared a similar disadvantage, as applying the transformations caused the resulting time series to trail behind the overall trend of the original time series. Ultimately, both methods did not offer enough volatility reduction in order to make up for this disadvantage, and the task of reducing the input volatility was to be solved in the modelling step rather than in the preprocessing step.

5.3 Feature Engineering

As a final step, it was also necessary to engineer and extract further features from the raw data. Aside from the time series approaches which can be applied directly to the series in question, most Machine Learning algorithms require feature engineering to prepare time series data for modelling. Similar to an auto-regressive approach on the energy output time series, I decided to include some of the most recent observations relevant to the current timestamp as features. In particular, I created features for up to ten of the most recent observations, as well as observations for the same timestamp on up to 4 days prior. These features can be obtained by simply pushing the series forward by x number of periods which was then referred to as Lag x or t-x.

In another attempt to combat the seasonality and volatility of the data, I also decided to include variables for the time, or rather quarter-hour of the day. I labelled each quarter-hour of each day from 1 through 96 and created a new column with the label. As some algorithms cannot handle categorical data with a separate weight per label, I decided to create dummy variables instead. The process of creating dummy variables creates a new feature for each label, and inserts a value of 1 if the label matches the feature and 0 otherwise. Instead of creating 96 new features, however, I decided to group the quarter-hours where all values were equal to 0 into one Night feature thus reducing the total number of new features. This step allows algorithms to set a constant value for each quarter-hour, which could help in stabilizing the predictions. I also ultimately decided to keep the initial labels feature with labels in integer format as it would allow some tree-based algorithms to create splits at certain times of the day.

Finally, I also included the weather-based forecast features, which were also transformed in the same way as the infeed time series. Figure 5 also references Infeed/Forecast Ratio and Moving Average Terms, but after some initial testing both of these parameter classes were deemed insignificant and will therefore not be described further. Thus, the features for much of the modelling process were transformed Lags, weather-based forecasts and the quarter-hour variables.

5.4 Time Restriction

Since dusk and dawn hours seemed to be volatile and difficult to forecast, we developed an approach which would remove night hours and a set number of hours at the beginning and end of every day from the training set. I called this method relevant time indices and applied it after all the other preprocessing steps. Essentially, the method would gather the sunrise and sundown times for every day in the training set, add or subtract the specified number of 15 minute periods from the time and remove all rows with time indices before or after these cutoffs for the specific day. The issue with this approach is that while it allows the prevention of forecasts for early morning hours, the values for the morning hours still affect predictions because they are included as Lag features for the initial periods after the cutoff. It seemed that this issue combined with reducing the test set ultimately had a negative effect, which will be discussed further in the following section.

6 Analysis & Results

With the dataset ready for modelling, the next step was to start training and evaluating the different algorithms for prediction. Instead of diving directly into the algorithms, it is important to document the general process I followed when narrowing my search, testing different algorithms and ultimate improving and tuning the final choice of algorithm for this particular problem. In the very beginning of testing, I created a Python script with a modelling class which had a preprocess(), fit() and predict() function similar to the scikit-learn package. These functions took the data and the initialized model as input as well as custom parameters indicating the specific preprocessing and modelling configurations. I created separate fit and predict functions for the recursive and multiple models approaches as well as for the walk-forward validation approach used in the later stages of modelling. This standardization of the process allowed me to run a large number of trials efficiently and in a coherent manner.

In terms of a general timeline, I first ran simple modelling tests with basic configuration for all algorithms on a single German power plant, allowing me to narrow them down to the top four algorithms in terms of prediction accuracy and seeming stability. After some initial hyperparameter tuning, I conducted several test series to figure out the best approach in terms of transformation and prediction method for each of the algorithms as well as the most promising algorithm overall. I applied grid search to find the optimal configuration for the algorithm and tested it on all other plants using both described methods of walk-forward validation. Finally, I looked into prediction accuracy in relation to training and testing set size and processing time in order to optimize the initial goals of a fast, stable, safe and independent algorithm.

6.1 Narrowing the scope

Initially, the algorithmic scope was set to be very broad on purpose, in order to not exclude any algorithm due to sheer bias. The overview table in Figure 5 lists 9 algorithms, some of which can be broken down further into several different algorithms as well such as Neural Networks. In order to get a sense of the viability of the different algorithms, I decided to implement all algorithms named with the bare minimum of preprocessing steps needed as well as a relative small number of parameters if required.

Even though the results were not very promising overall, it exposed some limits and weaknesses of certain algorithms. Some algorithms such as SARIMA or Gaussian Processes either threw a runtime error or had to be shut down due to incredibly long runtime. LSTMs and other neural networks required special preprocessing steps, more manual setup and long training times as well while producing results far below the other algorithms. I also experimented with naive forecasting and other simple techniques with fairly poor results comparatively. Finally, Linear Regression showed fairly promising results initially but Ridge Regression, which is a regularizable adaptation of Linear Regression, showed better results as a similar algorithm while also allowing

for further tuning.

6.2 Approach Comparisons

Considering the limitations for the above algorithms, I decided to move forward with Support Vector Machines, Random Forest, XGBoost and Ridge Regression. In order to evaluate both the algorithms as well as the other preprocessing and modelling approaches, I decided to specify an experimental setup under which all the algorithms were to be evaluated. Specifically, I only used the data from one German plant, with the testing set comprised of the last 20% of observations. The forecast horizon, i.e. the number of periods to be forecasted into the future, was set to 10 periods or two and a half hours. In addition, the predictions used in the overall accuracy calculations were subject to the same relevant indices cutoff used in some of the experiments, namely 8 periods after sunrise and 5 periods before sundown. Lastly, the parameters were standardized to include four day lags and four recent period lags in addition to the other forecasts and the dummy variables.

The different approaches used were simple recursive plus regular difference preprocessing, CSI normalization preprocessing, applying relevant indeces on top of CSI normalization and multiple models for each timestep also with CSI normalization. The reason the CSI Norm was used for most of the experiment is because early testing showed that it was far superior to differencing for all algorithms. As an example, Figure 7 shows the RMSE forecast error at different time steps from the initial prediction for XGBoost and all four approaches. The forecast error for CSI Norm is lower for all time steps shown. Furthermore, although the recursive CSI Norm approach has a slightly higher initial forecast error as compared to other methods, it seems to generalize the best multiple periods into the future, even better than the multiple models approach. This observation holds true for all other tested algorithms as well, which is why it was chosen as the approach going forward.

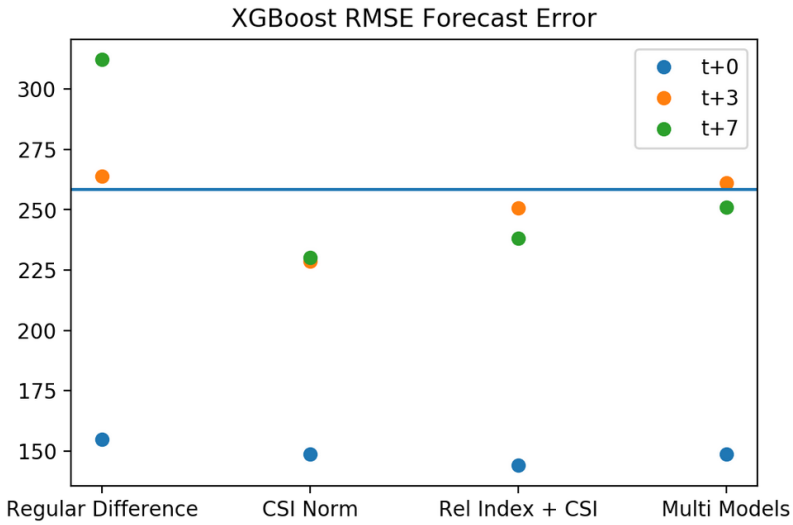
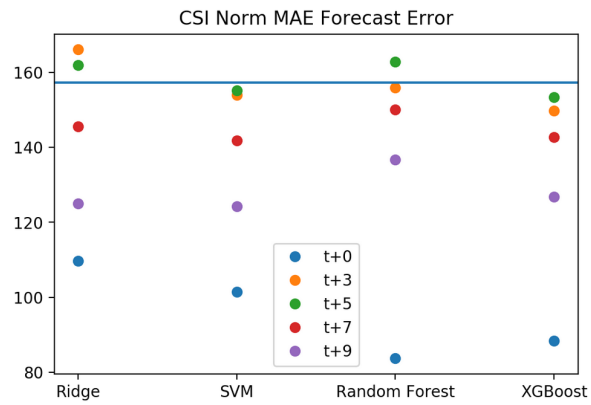
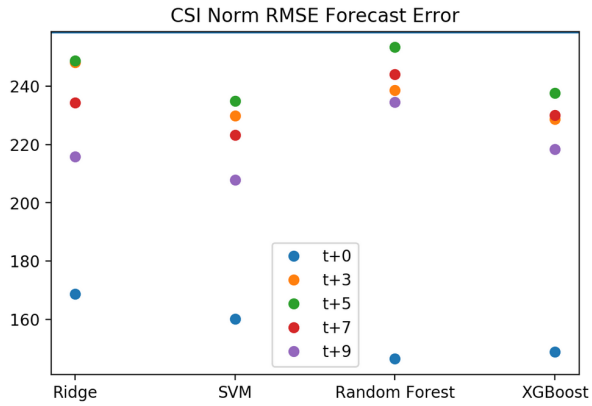


Figure 7: RMSE forecast error for the XGBoost Regressor and different approaches at different timesteps from the initial forecast. Note: Blue line represents the RMSE of the weather-based forecast with the lowest error metric.

When it came to comparing the distinct algorithms, all approaches again showed similar results. Figure 8 shows the forecasting error for all algorithms using the CSI Norm approach. Support Vector Machines and XGBoost Regression show lower forecasting errors for both error metrics. In addition, it is also important to note that both algorithms on average outperform the best weather-based forecast in terms of both MAE and RMSE for these time periods, which is promising consider the limited hyperparameter tuning which was performed for this experiment.



(a) MAE of forecasts using different algorithms at different time steps.



(b) RMSE of forecasts using different algorithms at different time steps.

Figure 8: Testing set forecasting errors for different algorithms after applying CSI normalization. Blue straight line signals lowest forecasting error for weather-based forecasts.

Lastly, another thing to note from these graphs is that the errors for some periods further in the future seems to be lower than periods closer to the initial prediction period. This phenomenon

is caused by the relevant indices approach, which essentially limits the t+x predictions for higher x to periods with lower values overall, i.e. afternoon and evening periods, which makes it seem like the errors are lower overall. Figure 9 illustrates this concept, as it groups the RMSE by time of day rather than just showing the overall average. Even though t+7 and t+9 show very similar error curves throughout the day, t+7 includes two more periods with relatively high average error and thus will have a higher overall RMSE. Another important observation is that t+7 and t+9 predictions still perform similarly or slightly better to the weather-based forecast.

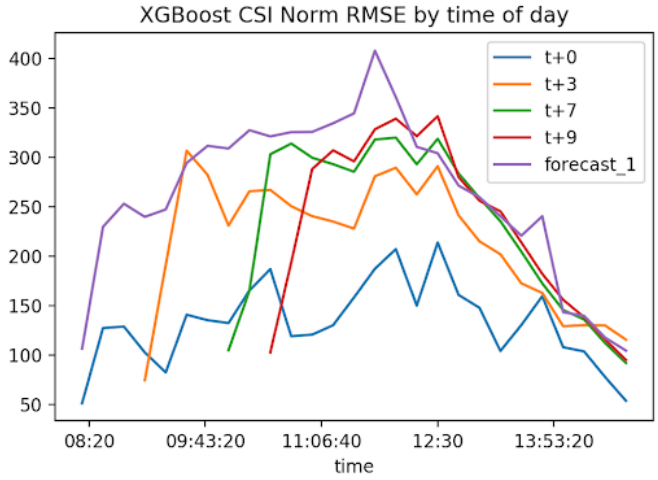


Figure 9: Line plot of the RMSE by time step of forecasts generated using the CSI Norm and XGBoost grouped by time of day.

Considering these results, I decided to further test Support Vector Machines and XGBoost using the expanding window walk-forward validation approach. In these tests, it became apparent the Support Vector Machines rely on careful parameter tuning in order to produce results comparable to XGBoost, with slight misconfigurations resulting in much larger errors. In addition, XGBoosts model training time was far superior to that of SVMs especially for large training sets. Both of these factors were important goals set in the initial framework, namely a fast and stable algorithm, which led to my decision to focus solely on XGBoost Regression for the remainder of the project.

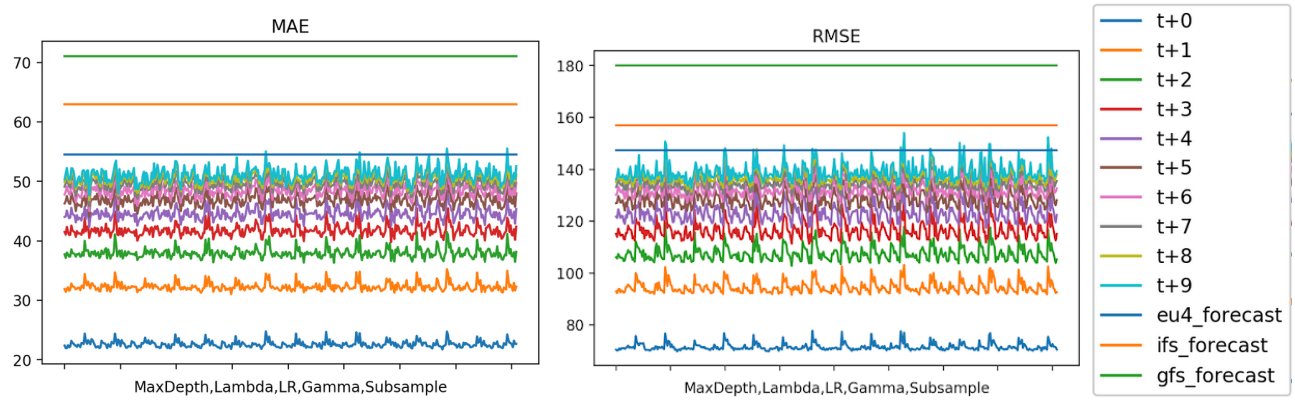
6.3 Algorithm Tuning

Before getting into the actual model hyperparameter tuning, I want to give some background on XGBoost to illuminate the importance and meaning of the parameters discussed in the later part of this section. XGBoost is an implementation of the gradient boosting decision tree algorithm, which describes an application of gradient descent to boosting of ensembled decision trees. Boosting itself refers to an ensembling technique which adds new models (individual trees) to mitigate the errors made by previously built models; gradient boosting refers to the use of gradient descent to minimize the loss function when training the algorithm [9]. The initial XGBoost paper was

published in 2016 and was developed as an open-source project with a focus on computational speed and model accuracy [3].

In order to find the optimal hyperparameters for the model, I decided to run an extensive parameter grid search on the parameters Max Depth, Lambda, Learning Rate, Gamma and Subsample. The reason for searching so many parameters at once is because changing individual parameter values can affect the optimal values for other parameters, which makes it near impossible to look at each parameter in a vacuum. In terms of the parameters in question, the Learning Rate governs the rate at which trees are updated between each iteration. The Max Depth and Gamma parameters both influence tree structure, namely setting the maximum number of consecutive splits as well as the minimum model improvement necessary to create a new split. Lambda applies L2 regularization and subsample specifies how much of the training set to use for each run of the algorithm; Both parameters aim to further prevent overfitting [10].

The initial results for the Grid Search can be seen in Figure 10. Although it is hard to identify the optimal mix of parameters from these graphs, it is important to note how stable the algorithm seems in terms of both error metrics while also displaying some clear patterns.

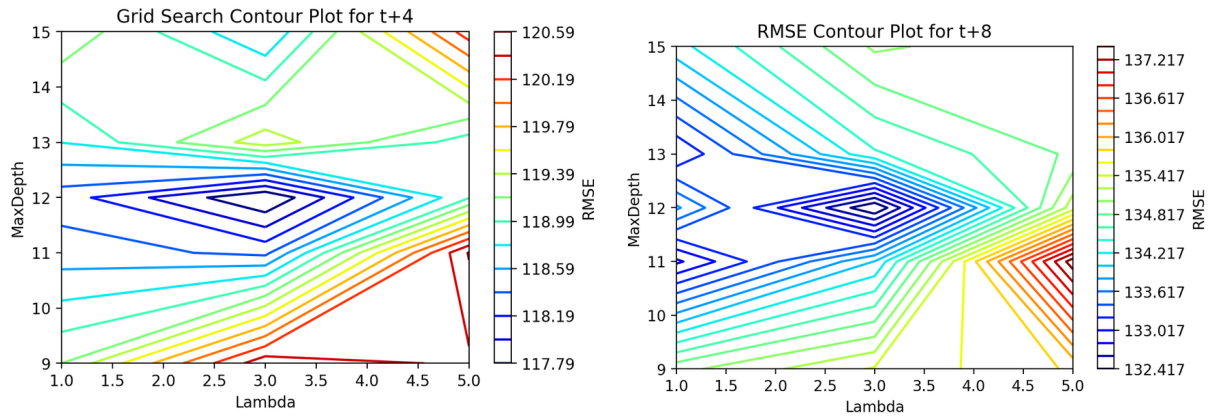


(a) MAE forecast error for different parameters combinations used in grid search.

(b) RMSE forecast error for different parameters combinations used in grid search.

Figure 10: Line plots of forecast errors generated using different hyperparameter combination during grid search.

After ordering and grouping these error metrics by the parameters Learning Rate, Gamma and Subsample, the optimal choice of value for each became easily apparent. For the remaining two parameters Lambda and Max Depth, I filtered out the error values with non-optimal other parameters and decided to graph contour plots at several forecast lengths to visualize the results. Figure 11 shows contour plots for the RMSE value for the t+4 and t+8 forecasting steps. Both plots display clear minima at a Max Depth of 12 and a Lambda of 3.



(a) Contour plot of RMSE at t+4 by Lambda and Max Depth XGBoost parameters.

(b) Contour plot of RMSE at t+8 by Lambda and Max Depth XGBoost parameters.

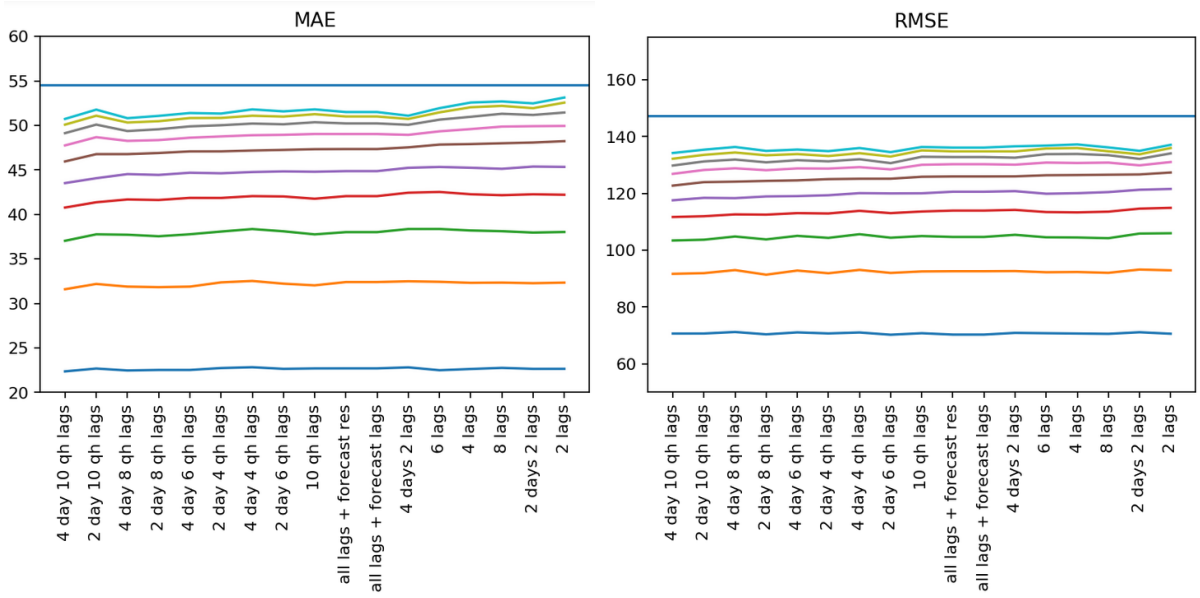
Figure 11: Contour plots showing the effect of different parameter values for Max Depth and Lambda on overall RMSE.

Thus, the final algorithm configuration included a Learning Rate of 0.1, Gamma of 0.5, Subsample of 1.0, Max Depth of 12 and Lambda of 3. This configuration was tested on the data for all remaining plants and used for the remainder of the project.

6.4 Final Framework Configuration

Besides optimizing the algorithm, I also attempted to optimize the included features and the optimal training size and frequency for refitting the model.

When considering the optimal features, I mostly focused on the different combinations of included lags but also included slight variations to the weather-based forecast such as also including lags or residuals for the most recent forecasts. Figure 12 shows the resulting error metrics for forecasting steps t to t+9. The graphs are ordered by t+3 forecasting error (last observation time plus one hour). Overall, it seems like these features only have a slight effect on the algorithm's ability to produce an accurate model. Still, more included lags show slightly higher accuracy. I did not try to include even more day lags, as the amount of data required to produce live forecasts would increase greatly with each day. Luckily, it seems that only including the ten most recent observations and no day lags as features does not decrease model performance by a lot. For now, however, I decided to include the features with the highest overall accuracy which is also the features with the maximum number of day and regular lags.



(a) Line plot of MAE at different time steps and for different feature sets.

(b) Line plot of RMSE at different time steps and for different feature sets.

Figure 12: Line plots showing the effects of different parameter combinations on prediction error. The feature sets are ordered by t+3 MAE and RMSE.

As for optimizing the training size, I first looked at training time relevant to training set size in isolation. Figure 13 shows that the training time increases linearly with the number of days included in the training set. XGBoost is an algorithm that was built mainly with efficiency in mind [9], which explains the training time of mere seconds relative to tens of thousands of rows. Considering a PV plant portfolio of more than 2000 plants, training time and training frequency remains in need of optimization.

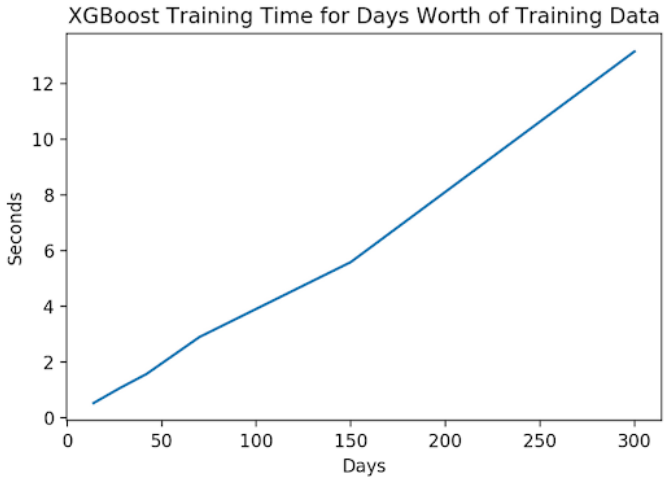
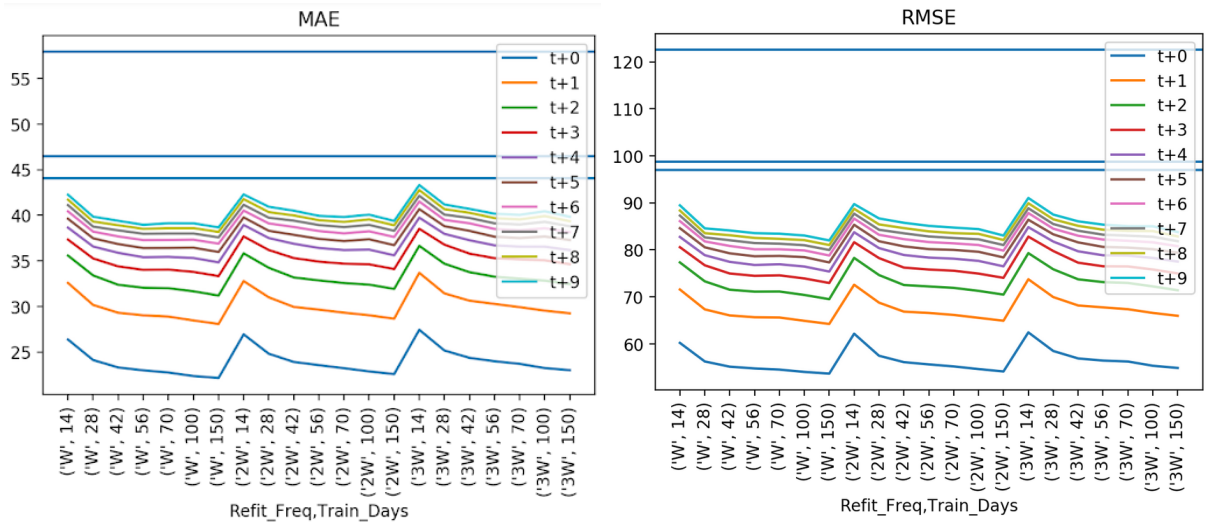


Figure 13: Line plot of XGBoost training time in seconds versus days worth of training data used to train the model.

Therefore, I decided to look at predictive accuracy relative to the size of the training set and the frequency at which the model is fitted anew. This experiment was carried out using rolling window walk-forward validation to get a better sense of the accuracy over time. Figure 14 shows the overall accuracy results of this experiment. As expected, a larger training size yields better accuracy. Yet, the improvements become fairly negligible after including around 56 days worth of data for this particular plant. In addition, the frequency of fitting seems to have an effect albeit very limited. Ultimately, these graphs can only be used to guide the decision regarding the optimal trade off between predictive accuracy and processing cost, yet luckily it seems that predictive accuracy does not suffer significantly when reducing the size of the training set and the frequency of updating the model.



(a) MAE at different time steps for different training and refitting window lengths.

(b) RMSE at different time steps for different training and refitting window lengths.

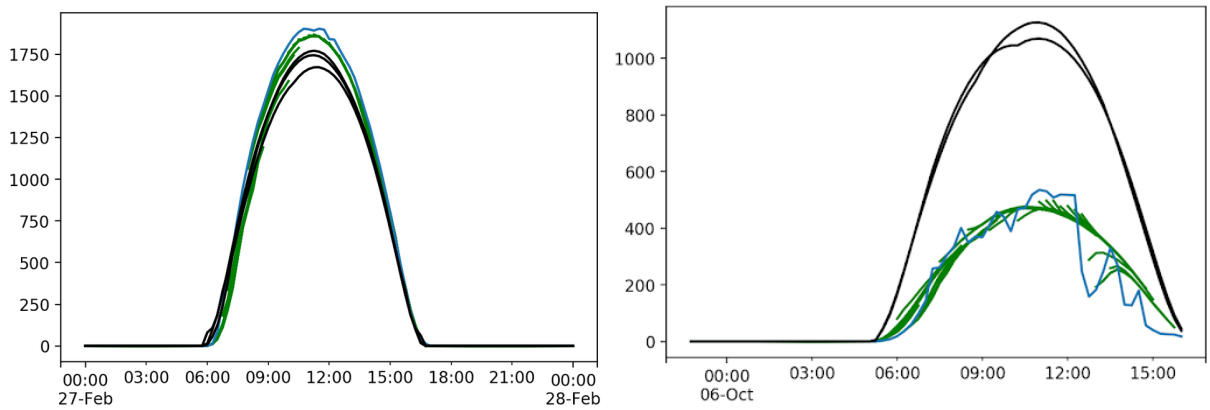
Figure 14: Error metrics for different rolling window configurations and at different time steps. The stright lines represent the weather-based forecasts.

6.5 Prediction Results

Revisiting the forecasting framework described in the introduction, we seem to have shown that the described approach satisfies three out of the four mentioned goals. Namely, the time to train the model and generate predictions is low, the trained models show stable, good performance which improve upon the previous forecasts and they are independent in the sense that it can be applied to a variety of parameter sets with similar performance. The last remaining goal was for the model to be safe, in other words that the produced forecasts do not show any unrealistic movements or sudden outliers in their prediction.

While this could certainly be answered through error and outlier metrics, I think it is important to combine the metrics with a visual indicator of model performance and stability. In Section

3, Figure 1 shows line graphs for two example days which illustrate certain issues within the raw dataset. Figure 15 features graphs for the same days including all 10-period forecasts generated for said days which were generated using the same rolling window walk-forward validation approach as discussed in the previous section. For both days, the green prediction lines follow the blue plant output much closer than the weather-based forecast. Especially the graph 15b shows a vast improvement, with the forecasts looking similar to a smoothed trend line for the energy output rather than following or nearing the weather-based forecasts. From these two days, the forecasting algorithm seems safe, especially when comparing it to the weather-based forecasts.

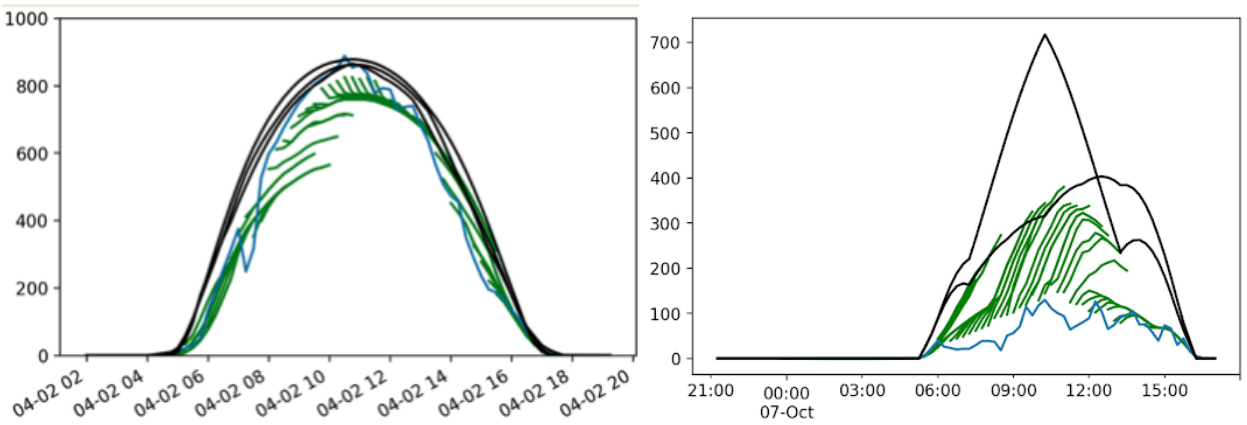


(a) Lineplot of actual values, previous forecasts and 10-period forecasts for every quarter hour on February 27.

(b) Lineplot of actual values, previous forecasts and 10-period forecasts for every quarter hour on October 6.

Figure 15: Predictions for days shown in Figure 1. Ten period predictions for every timestep are shown in green, actual values in blue, other forecasts in black.

Although those two days are good examples of the approach's stability, there are examples of days where the algorithm is not as accurate. Figure 16 shows two examples where the algorithm either fails to improve upon the weather-based forecasts or wrongly attempts to bridge the gap between the most recent output values and the weather-based forecasts. It is important to note that there does not exist any feedback mechanism between prediction sets, meaning that the prediction later in the day for the second day are not a reaction to previous prediction errors. While both days certainly indicate poor performance, the prediction series cannot be described as containing unrealistic movements or sudden outliers. Rather, I would see these days as indicators for possible areas of improvement.



(a) Lineplot of actual values, previous forecasts and 10-period forecasts for every quarter hour on April 2.

(b) Lineplot of actual values, previous forecasts and 10-period forecasts for every quarter hour on October 7.

Figure 16: Predictions for example days with poor accuracy. Ten period predictions for every timestep are shown in green, actual values in blue, other forecasts in black.

7 Conclusion & Outlook

Overall, the presented framework, modelling approach and algorithm configuration satisfies the stated goals at the beginning of the project. In particular, the chosen XGBoost algorithm is fast in both training and prediction, shows relatively stable and safe performance and is independent from specific variables or features. In addition, the modelling class and script provided at the end of the project should allow for easy implementation into live forecasting given the necessary data pipelines and integration. The next steps would be to implement the framework for a couple of plants with live data, monitor performance and then start adding it to more and more relevant plants.

Although the model fulfills all the previously stated goals, there are certainly areas for improvement as well as further approaches which could be tested in conjunction with the current framework. Specifically, the effect of using different cleaning methods after applying the CSI normalization has on overall model accuracy could be tested. In addition, while it might be difficult to resolve the issues with morning and evening values of the Clear Sky Index, an investigation into the reason for some of the faulty values during the day in some plants could lead to a more precise Clear Sky Index, which in turn might have a positive effect on the predictive ability on the model. The reasons for the focus on the CSI in this section is due to the remaining issues that are still present in the current approach as well as the high importance of this metric within the framework overall.

In terms of further approaches to be used in conjunction with the current modelling framework, one area that I did not have time to explore further was a metric assessing the quality of

or confidence in the weather-based forecasts. While I did include recent forecast residuals in the feature search, there are plenty of other approaches to this issue. For example, one could develop a metric which assesses the relative confidence in the forecasted values based on the time elapsed since the last weather update or distance from the set forecast times. This metric could be used alongside the actual values and allow the model an extra option to decide how much weight to give a certain forecast at a given time and confidence level. Given the relative imprecision of the weather data and weather-based forecasts, I think this approach or similar ones could lead to a further overall improvement.

References

[1] Jason Brownlee. 4 strategies for multi-step time series forecasting, 2017.

[2] Jason Brownlee. Simple time series forecasting models, 2017.

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[4] Rob Hyndman and George Athanasopoulos. Evaluating forecast accuracy, 2013.

[5] NIST. Engineering statistics handbook: Autocorrelation, 2013.

[6] NIST. Engineering statistics handbook: Autocorrelation plot, 2013.

[7] NIST. Engineering statistics handbook: Partial autocorrelation plot, 2013.

[8] Alexander Phinikarides, George Makrides, Bastian Zinsser, Markus Schubert, and George Georghiou. Analysis of photovoltaic system performance time series: Seasonality and performance loss. *Renewable Energy*, 77:51–63, 05 2015.

[9] xgboost developers. Introduction to boosted trees, 2019.

[10] xgboost developers. Xgboost parameters, 2019.