Machine Learning Engineer Nanodegree

Capstone Project

Jan Engelke

July 29, 2017

Dependence of the soiling rate of solar concentrators on weather parameters

Content

1. Definition

1.1. Project Overview and domain background

Concentrating Solar Power plants (CSP plants) capture and concentrate sunlight and transfer its energy to a heat transfer fluid. Currently there are installed about 5000 MW of CSP power worldwide (source: cspplaza.com). It is predicted that due to the rapid development of emerging markets including Morocco, South Africa and China the CSP capacity will have a huge increase in the next years.

Figure 1: Solar Through Collector, source: www.dlr.de

Soiling (= "getting dirty") of these solar concentrators (= back-coated glass mirrors) is an important issue that significantly reduces plant efficiency and causes high cleaning costs for the plant operators. For a lot of potential plant sites there is no soiling data available. Also, in state-of-the-art cost-effectiveness studies during the planning phase of a CSP plant the soiling issue is not considered in sufficient detail: the soiling rate is assumed as a constant factor independent from plant location, without variation during the year and constant during the entire plant life. A recent study by (Wolfertsstetter, 2016) does an in-depth analysis on the effects of soiling on CSP plants. Soiling is dependent on several environmental factors like wind speed and direction, condensation and aerosol particle concentration. All of these parameters vary during the time of a year and strongly depend on the location of the plant. In the work of (Wolfertsstetter, 2016), a new measurement device is designed, that measures the cleanliness of solar concentrator (mirror) probe. During 3 years the cleanliness of the mirror has been logged together with the above-mentioned weather factors. The soiling rate is then found as the time derivative of the cleanliness.

Dr. Wolfertstetter and DLR e.V. (German Aerospace Center) kindly agreed to provide the measurement data and domain information and thus make this capstone project possible.

1.2. Problem- and Solution Statement

1.2.1. Problem description

In the work of (Wolfertsstetter, 2016) a prediction model for the soiling rate is built. The model is based on linear regression. The independent parameters (features) are the weather parameters like temperature, humidity, windspeed, mirror orientation and others. The dependent variable (label) is the soiling rate. As a result of the regression analysis it was found that the Pearson correlation coefficient (PCC) was > 0.3 for none of the features. None of the features shows strong linear correlation with the dependent variable. Only 4 of 43 input features showed a PCC > 0.2 with the labels, and all the other features showed a PCC < 0.2 with the labels. It is estimated that the model can make reliable predictions for a maximum of 2 months only. The aim of this work is to find a better prediction model.

1.2.2. Solution description

As the soiling rate consists of floating point values and requires no classification, the problem to solve is a regression problem. Solving the problem means to provide a model that accurately predicts the soiling rate dependent on the meteorological features.

The following steps are taken to solve the problem and create an accurate model:

- First, a label to be used with supervised learning is created. The label should be free of trends, or seasonalities, in order to avoid time series problems. While the cleanliness could be used as label available in 1-min resolution, it shows a strong decreasing trend, as shown Figure 4. To remove the trend, the time derivative of the cleanliness (soiling rate) has to be used.
- Data preparation of the data set
- An appropriate splitting method is chosen to divide the data set in test and validation data and optionally perform cross validation.
- Regression algorithms are to be applied, like Linear Regression, SVM, Decision Trees, AdaBoost and XgBoost, and MLP Regression NN. If available, the feature importance functionality of the algorithm should be used to gather information on the importance of the weather parameters.
- Create Learning curves in order to understand the behavior of the learner.
- Parameter optimization for the algorithm that is best suited, aiming to minimize the prediction errors
- Perform a PCA with the best-performing algorithm in order to reduce feature dimensions while capturing as much variance as possible. The hope here is that regression algorithms might work better on the PCA-transformed data set than with the original features.

1.3. Metrics

The Pearson coefficient reveals only linear dependencies and thus cannot be used as an indicator for non-linear dependencies. The R2 score is more appropriate for this analysis, as well as the mean squared error.

1.3.1. Mean squared error

The mean square error or MSE is a risk metric corresponding to the expected value of the quadratic error or loss (sckit-learn.org - mean squared error, 2017). The MSE incorporates both the variance and the bias of the learner. For an unbiased estimator, the MSE is the variance of the estimator (Wikipedia - mean squared error, 2017)

1.3.2. R2 score

The R2 score function computes R^2, the coefficient of determination. In terms of machine learning the R2 score compares a model to the simplest possible model, the mean:

$$R^2 = 1 - \frac{MSE(model)}{MSE(mean)}$$

MSE(model) and MSE(mean) being the mean squared errors of the performing model and of the mean, respectively. The MSE(mean) can also be seen as the variance of the data.

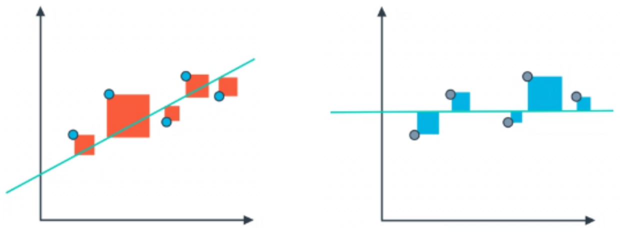


image source (Udacity, 2017)

Figure 2: MSE(model) on the left and MSE(mean) on the right, assuming a linear regression model in this example

Thus, the R2 score gets close to 1 either if the MSE of the model gets very small or the variance of the labeled data is very big.

When comparing R2 scores from train/validation- and test data sets, it is important to consider the variance of both data sets.

In more general terms the R2 shows the amount of variance in the labels, or dependent variable, that is predictable from the features. A R2 score of 1 indicates a perfect fit of model and data. A score of 0 indicates that the model is just as good as one that predicted always the mean of the data. Negative R2 scores indicate that the model performs worse than one that would just be predicting the mean. For non-linear models and/or data with underlying non-linear models the R2 score can become arbitrarily negative.

2. Analysis

2.1. Data Exploration

The main data set provides 1-min time resolution for all 43 measurement channels (features). It contains 874k raw-data points. The measurements were performed over approximately 1.5 years. Figure 3 shows the number of NaN values for each feature. The data set contains a huge number of NaN values that have to be processed in the data preparation step.

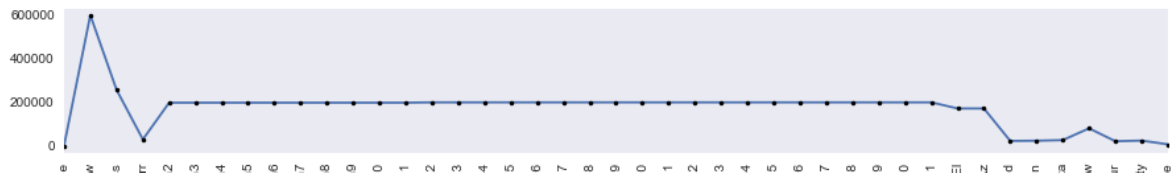


Figure 3: number of NaN values for each feature

There is a second data set available with 1-day resolution and 544 data points. Due to its small size and some negative pre-tests it is not considered for the supervised learning task anymore.

2.1.1. Understanding cleanliness and soiling rate

The cleanliness is measured in minute resolution and could be used as label. However, the measured raw data of the cleanliness is heavy bias and variance due to several optical and atmospheric effects (see Figure 4).

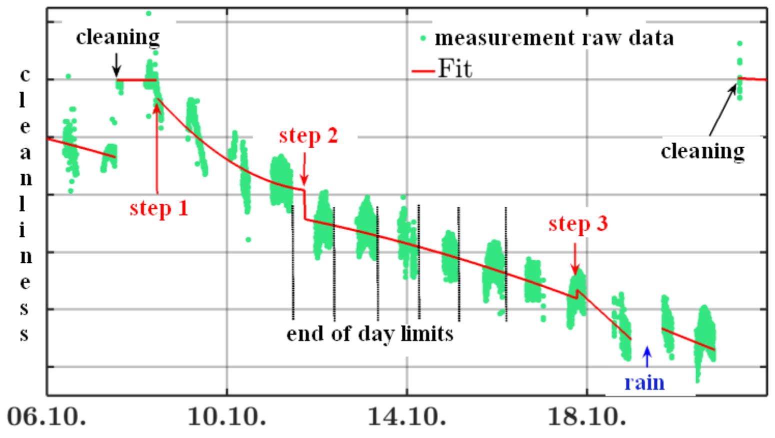


Figure 4: 1-min resolution data set cleanliness measurements (green dots) and the data fit (red curve) in order to create the 1-day resolution soiling-rate dataset. The figure includes 3 cleaning events that cause steps in the fitted curves. Rain effects the cleanliness, too.

Also, the cleanliness is affected by "cleaning events", when the mirrors are cleaned. In this case a step appears in the data after the cleaning. These steps depend on the applied cleaning schedule and cannot be predicted. And, most importantly, the cleanliness shows a time trend. The soiling rate is the time derivative of the cleanliness and independent of the applied experimental protocol. It does not show an obvious trend. In Figure 4 the soiling rate can be seen as the slope of the fitted (red) line. In order to calculate the soiling rate, a corrected version of the cleanliness is used. The correction includes removing the steps and fitting splines rather to use the original data. The whole process is done by domain experts at DLR and its steps are explained in (Wolfertsstetter, 2016). The correction is not scope of this work.

The corrected cleanliness is chosen as base feature for the creation of the soiling rate.

2.2. Exploratory Visualization

2.2.1. Feature and label distributions

In order to work well, most machine learning algorithms need the features as well as the labels to be distributed normally. Figure 5 shows some of the features' distributions after being preprocessed. The preprocessing steps are explained in section 3.1. As can be seen, some of the distributions are close to normal, while some others, like dew, are not at all normally distributed.

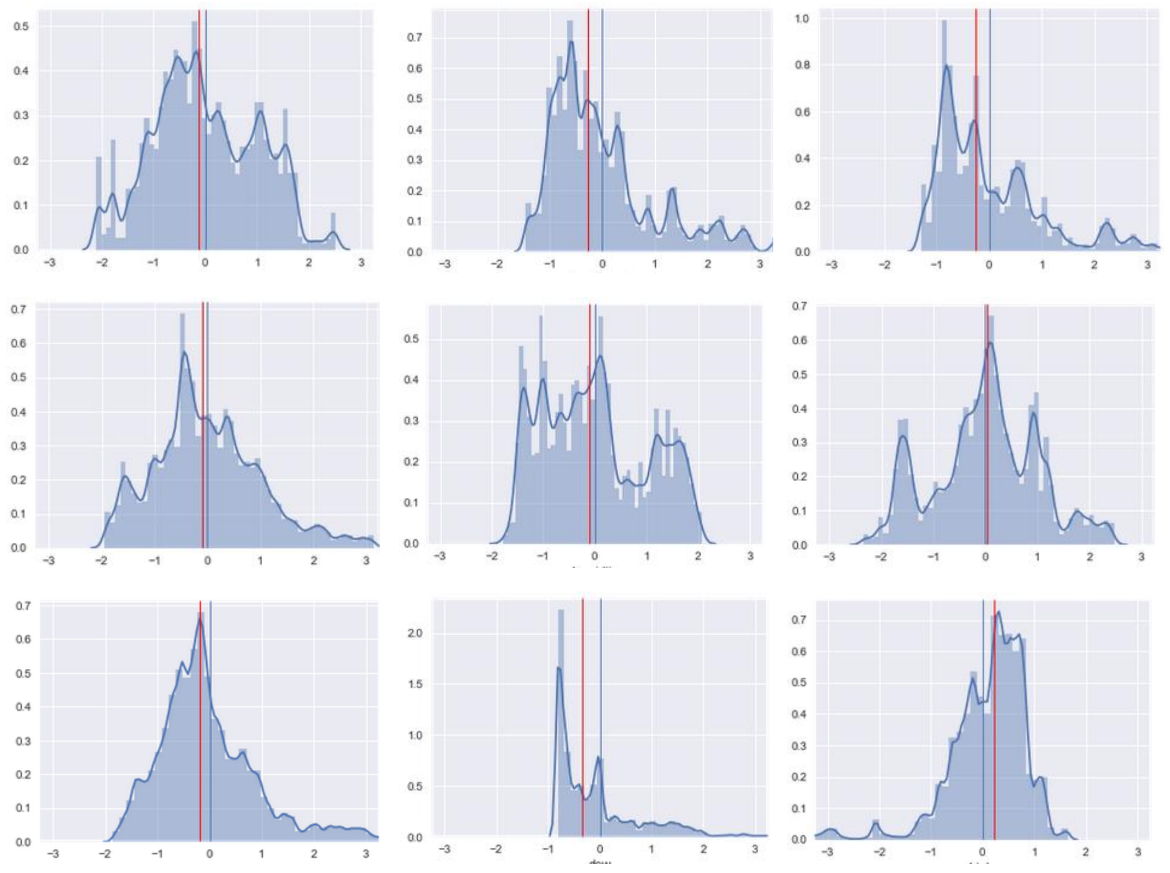


Figure 5: distribution plots for some of the input features and the label (soiling rate, "sr_highres")

The data distribution depends heavily on the data preprocessing like outlier removal, moving average and standardization- and normalization techniques. With the present data set, none of these techniques lead to a very good normal distribution, however. Also, advanced methods like log-transformations or Box-Cox transformations where not able to transform these distributions to normal. It is assumed that with more future measurement data the feature distributions will automatically get closer to a normal distribution.

2.2.2. Cleanliness and soiling rate

The corrected cleanliness provides the base for the creation of the label, the soiling rate. Figure 6 shows both the corrected cleanliness and the calculated soiling rate. The corrected cleanliness is a curve "patched together" by experts, based on domain knowledge and raw cleanliness data, as described in section 2.1.1. The raw cleanliness data is only available for a few hours during the day. For this time, the correction is done by removing measurement biases and fitting a spline to the raw data. During the night and morning hours no raw data is available and the corrected cleanliness is created by intra- and extrapolation.

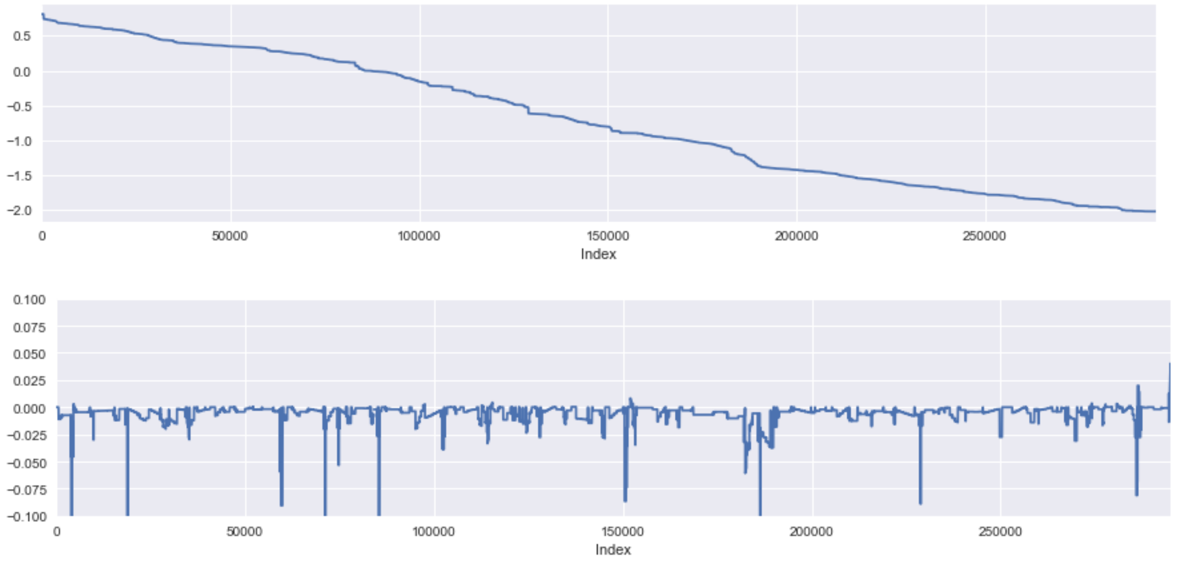


Figure 6: corrected cleanliness and calculated soiling rate, shown for the day time hours with DNI >200W/m^2, whole data set, recorded between 06/2013 and 08/2014

As the corrected cleanliness is a patched curve, the soiling rate shows a lot of what looks like strange artefacts.

2.3. Algorithms and techniques

2.3.1. day-based soiling rate creation

As explained in the last section, the soiling rate is calculated from the corrected cleanliness curve that is contained in the data set. The corrected cleanliness curve is provided for both moments with actual cleanliness raw data (during day time with sufficient solar radiation) and moments without cleanliness raw data (see Figure 4). However, tests showed that the corrected cleanliness should only be used for training when actual cleanliness raw data was available (about 30% of all data). When all of the corrected cleanliness data is used, the learner performance drops significantly. This seems caused by the fact that if no raw cleanliness data is available, the corrected cleanliness was inter- or extrapolated, and thus not accurate. In order to extract these subsets of data of corrected cleanliness, an algorithm was created. Its pseudocode is shown in Figure 7.

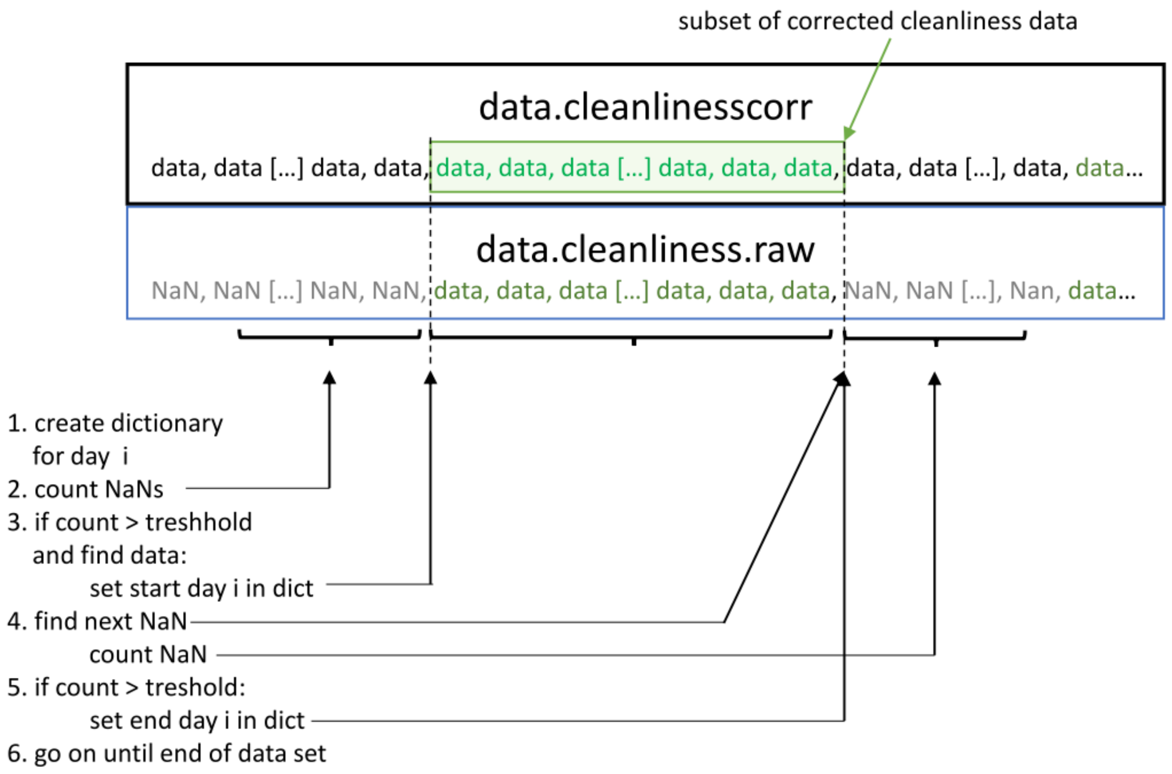

subset of corrected cleanliness data
data.cleanlinesscorr
data, data [...] data, data, data, data, data [...] data, data, data, data, data [...], data, data...
data.cleanliness.raw
NaN, NaN [...] NaN, NaN, data, data, data [...] data, data, data, NaN, NaN [...], Nan, data...
1. create dictionary
for day i
2. count NaNs
3. if count > treshhold
and find data:
set start day i in dict
4. find next NaN
count NaN
5. if count > treshold:
set end day i in dict
6. go on until end of data set

Figure 7: pseudo code for the algorithm that creates a dictionary for the extraction of daily subsets from the corrected cleanliness

Once the dictionary with all time windows is created, a number of steps are performed:

- a subset of the data frame is created, containing all data for the first time window
- small NaN gaps are filled with the daily mean in the corrected cleanliness data
- the soiling rate for the subset is calculated
- a moving average is performed on the soiling rate
- NaN gaps, caused by the moving average, are filled again with the daily mean
- the soiling rate is written into the subset

This is repeated for all time windows in the dictionary and after each subset operation the new subset is concatenated with the one that already exists. Thus, a new data frame is created that contains only the data recorded during the day and the newly created soiling rate.

While it might seem easier to just drop all data rows that contain NaN in the raw cleanliness in order to create the new data frame, in this case the soiling rate would contain artefact values at the transition from one day to the next. These would have to be deleted, making it necessary again to know the positions of all day-to-day transitions. Also, the calculation of the daily mean values, used to fill the gaps, would be more complicated. The method presented here makes it easy to perform any desired soiling rate operation on a daily base, not on the whole data set.

2.3.2. Moving Average

2.3.3. Outlier removal

2.3.4. Standardization

2.3.5. Gradient Boosting algorithms

Some of the algorithms that work best on the problem presented in this work, are gradient boosting- or gradient tree boosting algorithms. These algorithms use decision trees (rule sets) that act as weak learners (weak learner = slightly better than random guessing) and are used in a stage-wise additive model. This additive model adds tens or hundreds of decision trees to minimize a loss function. In stage-wise additive models, only one learner is added at at time and once a weak learner is added, it remains unchanged (). The loss function must be differentiable and could be – for regression tasks - the (mean) squared error, for example

2.4. Benchmarks

In (Musango, 2016) a "multilayer feed forward neural network" is proposed to estimate the cleanliness of CSP reflectors and an R2 score of 0.95 for the independent test data set is reported.

The work of (Musango, 2016) provides a useful base, as it tries to prove that MLP NNs (and thus other machine learning algorithms) can be used to solve this type of problems.

3. Methodology

3.1. Data Preprocessing

Several steps are to be performed before the supervised learning tasks can be implemented. They can be divided into general preprocessing steps and steps to prepare the supervised learning. Figure 8 shows the general preprocessing steps. These steps can be adjusted. The size off the moving average window, is set to 1 week, centered, moving average. The value was chosen because the cleaning interval of the solar mirrors uses to be 1 to 2 weeks and so the moving average and the average cleanliness of the solar collectors are in the same range.

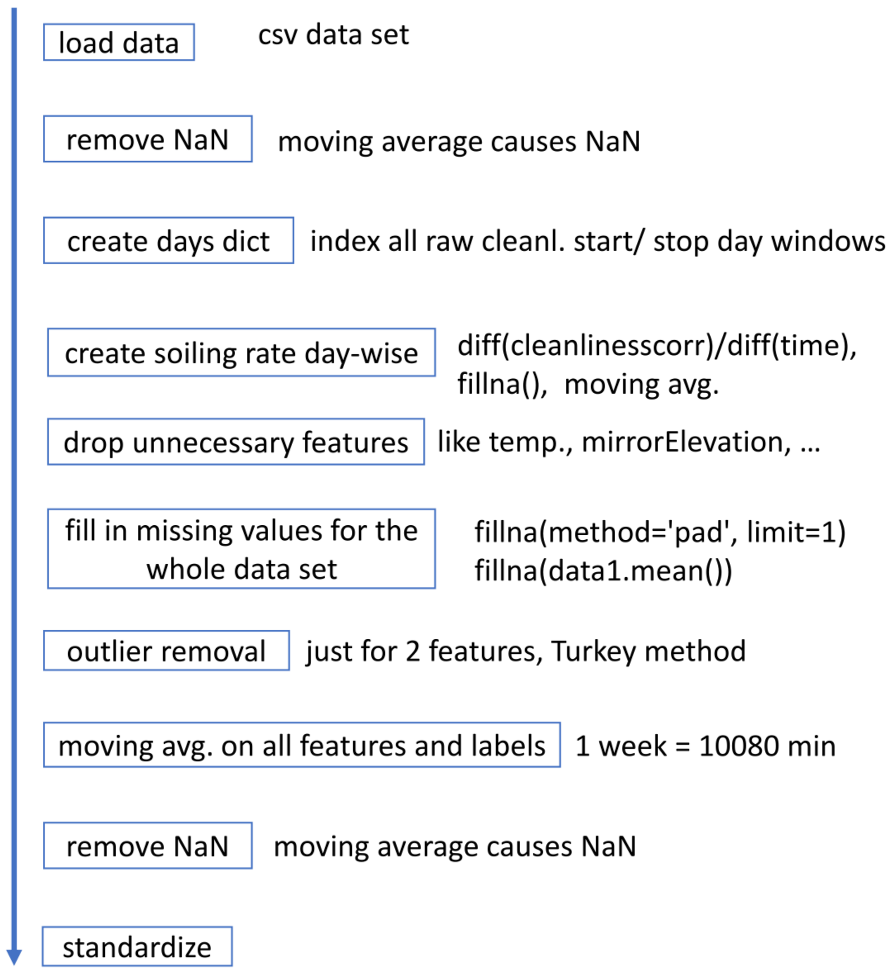

load data
csv data set
remove NaN
moving average causes NaN
create days dict
index all raw cleanl. start/ stop day windows
create soiling rate day-wise
diff(cleanlinesscorr)/diff(time), fillna(), moving avg.
drop unnecessary features
like temp., mirrorElevation, ...
fill in missing values for the whole data set
fillna(method='pad', limit=1) fillna(data1.mean())
outlier removal
just for 2 features, Turkey method
moving avg. on all features and labels
1 week = 10080 min
remove NaN
moving average causes NaN
standardize

Figure 8: first part of the preprocessing steps

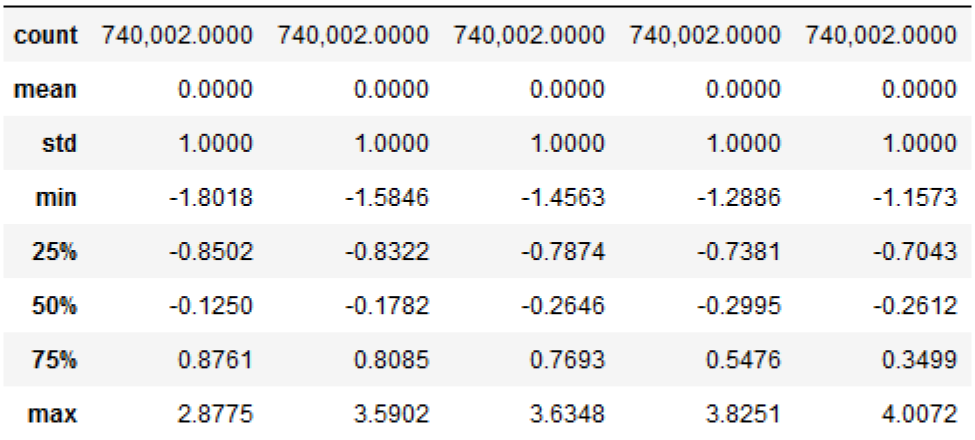
Table 1: Preprocessing parameter

property	**parameter(s)**
data set start / stop	2013/06/11
removed features	[...]
used features	[...] (35 features total)
daily soiling rate calculation: - fill NaN values method: - moving average	 fill with daily mean 10min, centered

fill NaN values method (all features and labels)	padding 1 value to the right, then fill all remaining gaps with the mean
Outlier removal	Tukey method, performed on 'f0xx' and 'f0cc'
moving average on whole data set including soiling rate	10080min, centered

After that the data set is ready and its basic statistic properties can be displayed, as shown in Table 2.

Table 2: basic statistics for the data set with the raw data



count	740,002.0000	740,002.0000	740,002.0000	740,002.0000	740,002.0000
mean	0.0000	0.0000	0.0000	0.0000	0.0000
std	1.0000	1.0000	1.0000	1.0000	1.0000
min	-1.8018	-1.5846	-1.4563	-1.2886	-1.1573
25%	-0.8502	-0.8322	-0.7874	-0.7381	-0.7043
50%	-0.1250	-0.1782	-0.2646	-0.2995	-0.2612
75%	0.8761	0.8085	0.7693	0.5476	0.3499
max	2.8775	3.5902	3.6348	3.8251	4.0072

In Table 2 some features and their summary statistics are shown. It is seen that the data set contains 740000 standardized data points. In a next step, the data set is prepared for the supervised learning task. To do so the data is first splitted into train-/validation data and test data.

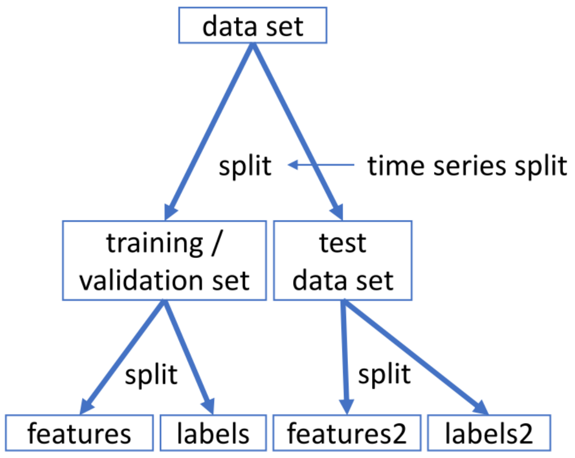


Figure 9: data splitting to prepare the supervised learning task

As shown in Figure 9 the data set is first splitted into the train-/validation set and the test set. The learning algorithm trains and validates using k-Fold cross validation, shuffle split or a similar technique. Here, shuffle split is used. After the training is complete, the test set is used in order to see how well the algorithm generalizes to new, unseen data. In order to improve by training, the learning algorithm needs the features (independent, input data) and the labels (dependent, output data) to perform the regression.

Table 3: key data of the splitted data sets

	Train / Validation data set		**test data set**
	training set	**validation set**	
start date	2013-06-26		2014-08-04
stop date	2014-08-04		2014-11-24
Variance	1.003		0.282
number of data points	157500	52500	60082

Table 3 shows the key data of the splitted data sets. Note the even though the train/validation set comprises 6 times more days than the test set, it has just twice the number of data points.

3.2. Implementation

3.2.1. Shuffle split cross validation

One basic implementation of the supervised learning task in this work is shown in Figure 10. The partition of the data set, that is assigned for training and validation, is used to create a number of distinct - training and assigned validation - sets. In this work, this number is usually set to 10, meaning we perform 10-fold cross validation or shuffled split cross validation with 10 splits.

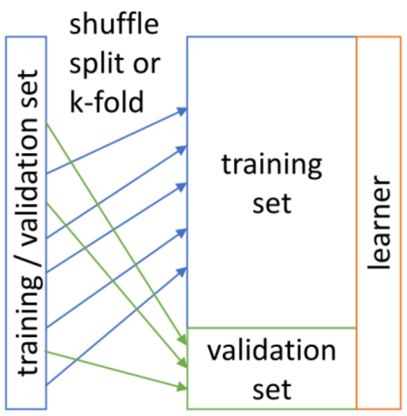


Figure 10: basic supervised learning task using k-fold cross validation

The only difference between shuffled splits and k-fold is that k-fold does use every data point only once while shuffled splits can use data points repeatedly in training- and validation sets. This makes learners using shuffled splits a little bit more prone to over-fitting, while on the upside shuffled splits provides some more data points in each split. In this work both split methods where tested and shuffled splits often provided slightly better results and hence is preferred to k-fold.

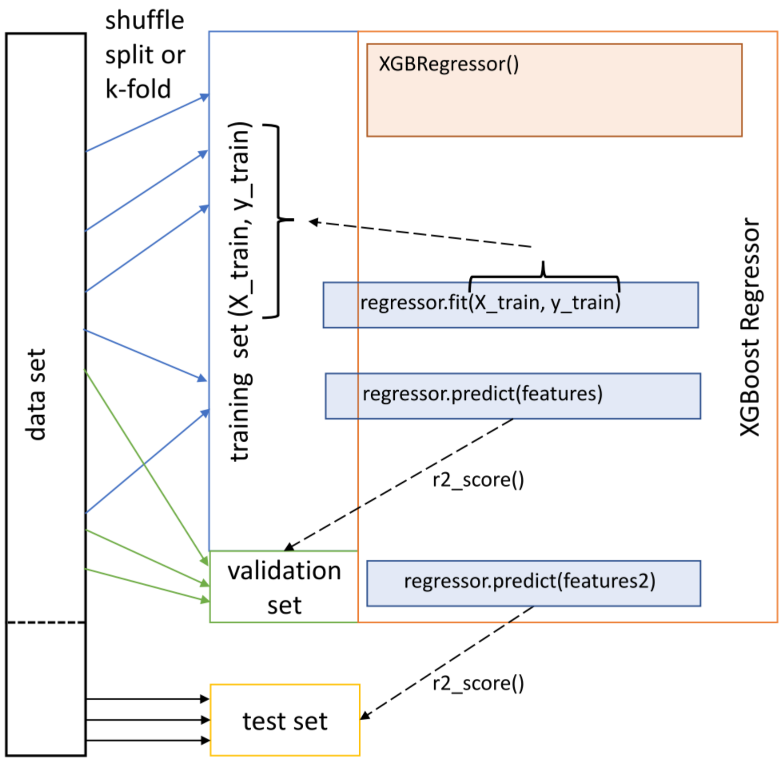

shuffle
split or
k-fold
data set
training set (X_train, y_train)
XGBRegressor()
regressor.fit(X_train, y_train)
regressor.predict(features)
r2_score()
validation
set
regressor.predict(features2)
r2_score()
test set
XGBoost Regressor

Figure 11: extended supervised learning task using train/validation data and an independent test set

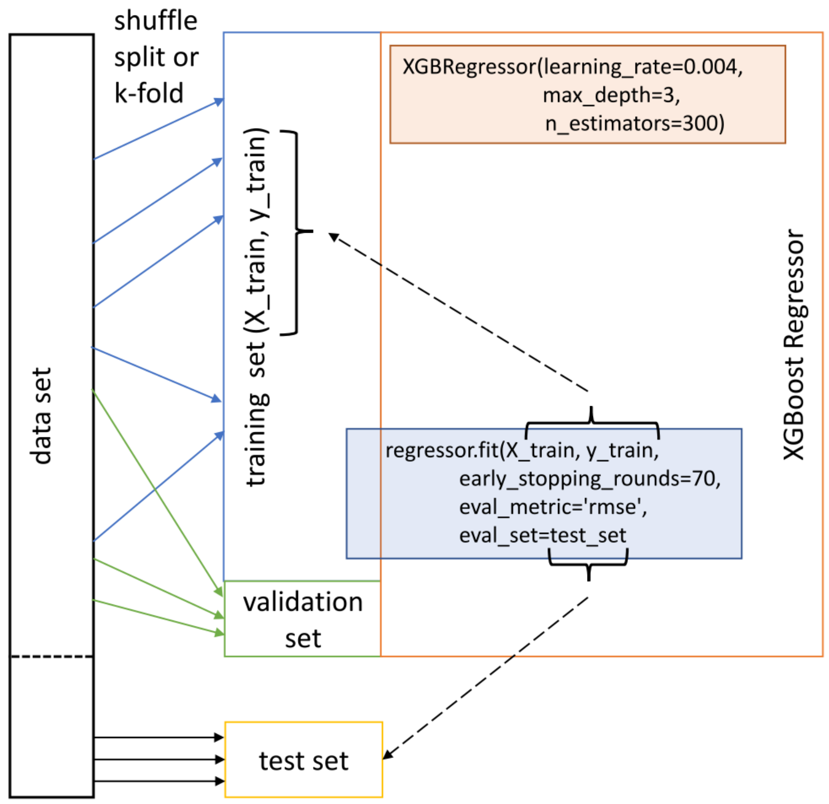
In a next step, the chosen model's generalization ability on new data has to be checked. Thus, an independent data set is introduced, the test data set. While the algorithms use parts of the validation sets for training during cross validation, the test set is truly independent and first shown to the learner after training and cross validation is finished.

Figure 11 shows the implementation. The introduction of the test set revealed strong overfitting behavior of the spot-checked algorithms:

3.2.2. Early Stopping functionality

The XGBoost algorithm provides a very practical functionality called early stopping.

> *(Early stopping) … avoids overfitting by attempting to automatically select the inflection point where performance on the test dataset starts to decrease while performance on the training dataset continues to improve as the model starts to overfit." (Brownlee, XGBoost with Python - Gradient Boosted Trees with XGBoost and scikit-learn, 2017)*



```
XGBRegressor(learning_rate=0.004,
             max_depth=3,
             n_estimators=300)
```

```
regressor.fit(X_train, y_train,
        early_stopping_rounds=70,
        eval_metric='rmse',
        eval_set=test_set
```

Figure 12: using XGBoost and early stopping

When using early stopping, the algorithms stops fitting more trees, if the chosen evaluation metric on the test set stops to improve for a number of steps. In Figure 12 is shown that the evaluation metric is the root mean squared error. Figure 13 shows the root mean squared error on both validation and test sets during the fit process, using early stopping.

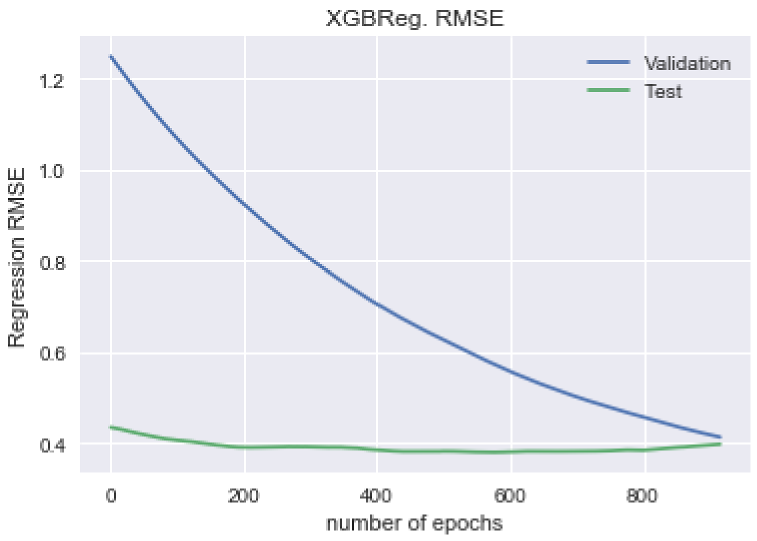


Figure 13: learning curves of the XGBoost algorithm on validation and test set

These curves are called learning curves. In the example shown in Figure 13, after finding the lowest RMSE on the test data set, the algorithm continues for 350 steps (early stopping rounds) before stopping the fit. It can also be seen that the lowest RMSE on the test set is reached after about 550 epochs (550 fitted trees), while the error on the validation set still decreases strongly. The early stopping rounds parameter was chosen like this to achieve the same MSE for train/validation- and test set.

3.2.3. Principal Component Analysis

The PCA is implemented in the following way

3.3. Refinement

Several parameters of the XGBoost algorithm were adjusted to achieve better results (Brownlee, XGBoost Tuning, 2017). The learning rate was reduced from 0.1 to 0.002 and the maximum number of estimators (number of trees) was increased from 100 to 1200. This combination usually increases the generalization performance at the cost of increased runtime (Note that due to early stopping the fit might stop i.e. after 900 trees, even if the number of trees is set to 1200). The early stopping rounds parameter was set to 320. Next, depth of the trees was reduced from 4 to 3, also to reduce overfitting and improve generalization capabilities. No further tuning was done.

4. Results

4.1. Model Evaluation and Validation

The learning algorithms used with the basic implementation are shown in Figure 14. Apart from these algorithms, SVM regression was also tested, but the time-to-fit was so long that the tests had to be interrupted and the SVM was discarded from the spot check.

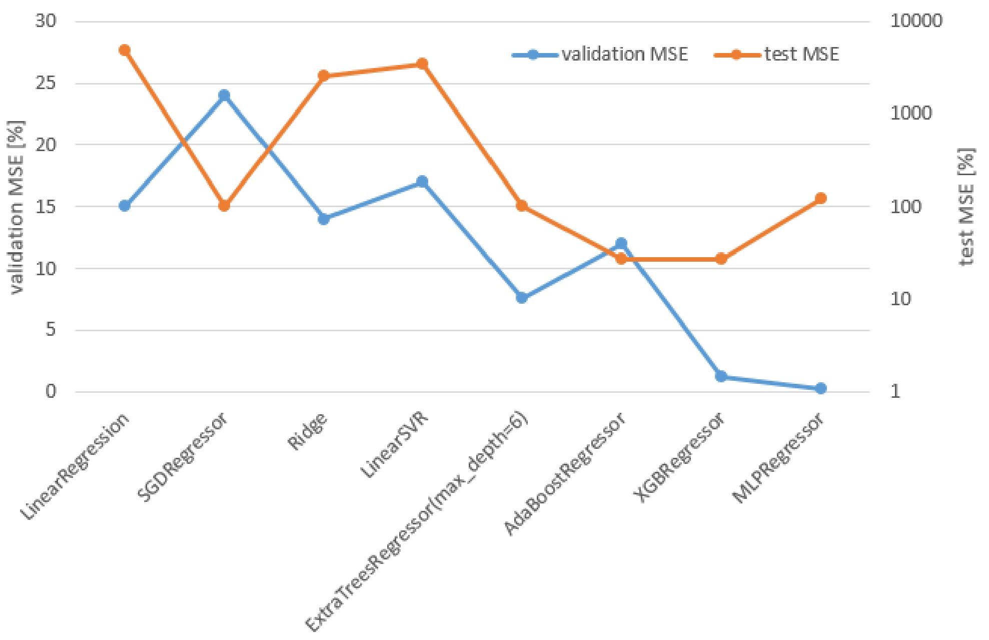


Figure 14: spot check, validation and test mean squared error of several supervised learning algorithms

Out of these algorithms XGBoost (Extreme Gradient Boosting) and the MLP regressor (Multilayer Perceptron Neural Network) reached the lowest MSE on the train/validation set, around 1.25% for XGBoost and 0.18% for the MLPR. As just stated, more important is the performance on the independent test set. There, AdaBoost and XGBoost got the lowest MSE with about 27%. The MLPR performed worse with 124%. Thus AdBoost und XGBoost seem to generalize better to unseen data than the MLP regressor, for example.

It seems that boosted trees algorithms in general perform well on this type of problems, as the ExtraTrees regressor also got a relatively low MSE of 100% on the test set. Linear algorithms like Linear regression or Linear SVR on the contrary were not able to capture the underlying model of the data and got high MSE up to 4800% on the test set.

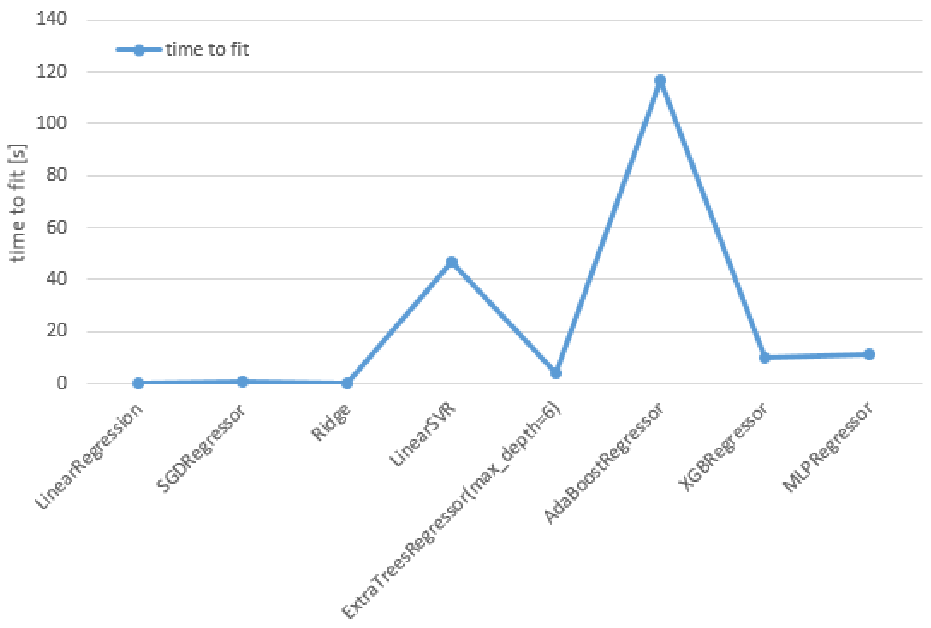


Figure 15: spot check, time to fit needed for the training data, for 1 split, 157500 data points, 35 features

As AdaBoost and XG Bost are the best performing algorithms on this data set, the time to fit, shown in Figure 15, is compared. It results that the XGBoost regressor is a lot faster in fitting the model, with 11.4s compared to 117s of the AdaBoost regressor. This is possible due to the parallelization of tree construction using all available CPU cores (multithreading)

Other advantages of XGBoost over other learners are, that, like AdaBoost, XGBoost provide feature importance functionality and XGBoost alone accepts sparse data (NaN values), which is a big advantage when working with the present meteorology data. Thus, XGBoost is chosen as the algorithm to work with.

4.1.1. Short- mid- and long-term prediction models

The learning tasks are divided into 3 learning tasks: long-term, mid-term and short-term prediction. The following table characterizes these terms in detail:

Table 4: soiling rate learning task classification

	training / validation time span data	**test time span data**
long-term prediction	> 2 years	6 months to several years
mid-term prediction	< 2 years	< 6 months
short term prediction	< 4 months	< 1 month

4.1.1.1. Long-term model performance

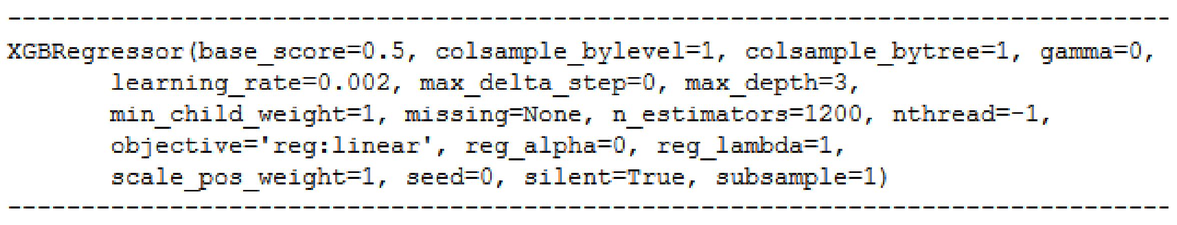
Long-term training brings the advantage of lots of available data. It is supposed that the features and thus the soiling rate will show cycles with one-year periodicity. Maybe there are even more cycles hidden in the data. Also, there could be a global trend hidden in the data. These occurrences of seasonality and trends reduce the performance of the regression learners and thus have to be identified and removed before training the model. At the moment of writing this work, no long-term data sets where available and with the available data no seasonality or trend could be identified. The long-term prediction performance of the learner could not be evaluated.

4.1.1.2. Mid-term model performance

The train/validation test set comprises 13 months of data and the test set 3.5 months of data. As can be seen in Table 3 and Figure 17, one clue of these data subsets is, that the variance of the train/validation set is 3.5 times higher than the variance of the test set.

Figure 16 shows the parameters that can be adjusted in XGBoost. Parameters found to work well on this data set and where changed from its standard are:

- learning_rate = 0.002 (standard = 0.1)
- max_depth = 3 (standard = 4)
- n_estimators = 1200 (standard = 100)
- earlyStopRounds = 320



```
--------------------------------------------------------------------------------
XGBRegressor(base_score=0.5, colsample_bylevel=1, colsample_bytree=1, gamma=0,
       learning_rate=0.002, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=1200, nthread=-1,
       objective='reg:linear', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1)
--------------------------------------------------------------------------------
```

Figure 16: final model parameters as shown in Python

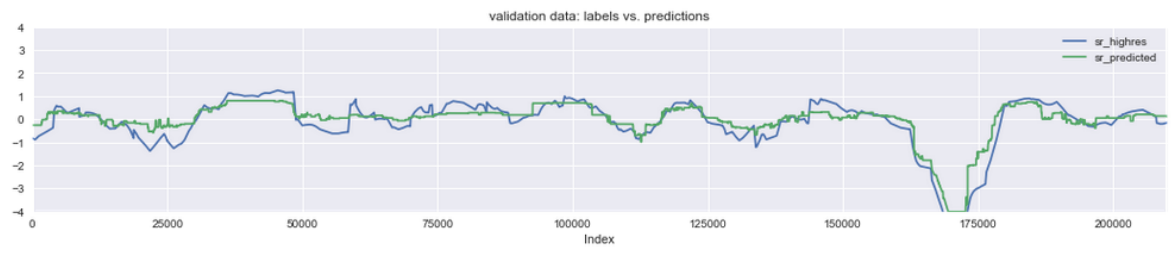
Lowering of the learning rate combined with increasing the number of estimators (the number of trees) usually reduces overfitting at the cost of longer algorithm runtimes. As described in section 3.2 and Figure 11, the early stopping functionality can be used to stop the fitting process as soon as the lowest error von the test set is reached.

Table 5: Results for the learning task

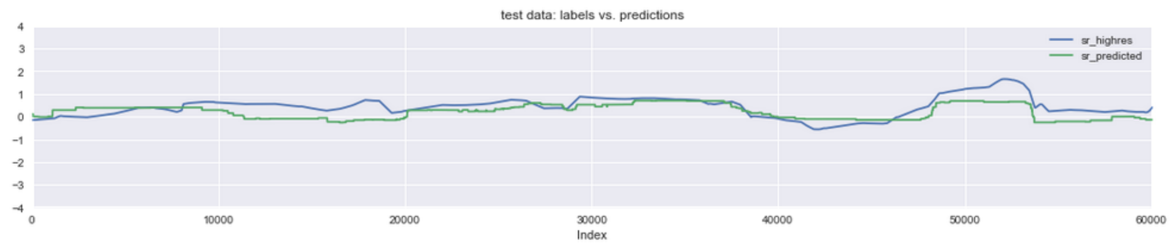
	Train / Validation data set	**test data set**
MSE [%]	17.7	15.6
r2 score	0.848	0.16
soiling rate mean	-0.122	0.428
predicted mean	-0.017	0.219

deviation predicted / soiling rate [%]	-86.8	-48

The mean squared error (MSE) for both train/validation and test sets are 17% and 16% respectively. The r2 score for the train/valid. set is 0.13 and 0.85 for the train/valid. set. As a next quality measure, the mean of the soiling rate introduced. When predicting the soiling rate over a period of time, it is important that the average soiling rate is accurately predicted, because the total soiling of the power plant's solar concentrators over a period of time is very important. In this test, it can be seen, that the deviation of the soiling rate for the train/valid. set deviates -80% (the predicted soiling rate is 80% below the actual soiling rate) and for the test set the deviation is -52%.



```
('validation set r2 score:', 0.84820547688330328)
validation set mean squared error: 17.70%
sr mean: -0.122379074689 | predicted mean: -0.0169649347663 | pred. deviation from sr: -86.1373892478%
```



```
('test set r2 score:', 0.15552652201571771)
test set mean squared error: 15.55%
sr mean: 0.427742180433 | predicted mean: 0.219372272491 | pred. deviation from sr: -48.7139023162%
```

Figure 17: result plot for the mid-term learning task, using the advanced model

Figure 17 shows the result of the training and testing process. In the upper plot the train/validation run and below the test run is shown. Looking at the plots in Figure 17 it can be seen, that the model's prediction sticks closer to 0 than the actual values. It can be said that the model underestimates the deviation of the soiling rate from the mean. This is also an important point in (Musango, 2016) where it is stated:

"The neural network underestimated soiling measurements with maximum percentage difference between the measured and estimated cleanliness of about 5.4%."

Even though (Musango, 2016) used the cleanliness as label and not the soiling rate, the reason for this underestimation in both works could be caused by the data distribution of the features and labels, as discussed in section 2.2.1 and Figure 5. The soiling rate, for example, is not very normally distributed. There are several peaks far from the mean, that could lead to bias in the model's predictions. Also, the features that are important for the prediction, do deviate from a normal distribution. Figure 18 shows the feature importance for the above created mid-term model. Lots of the important features in Figure 18 are not normally distributed.

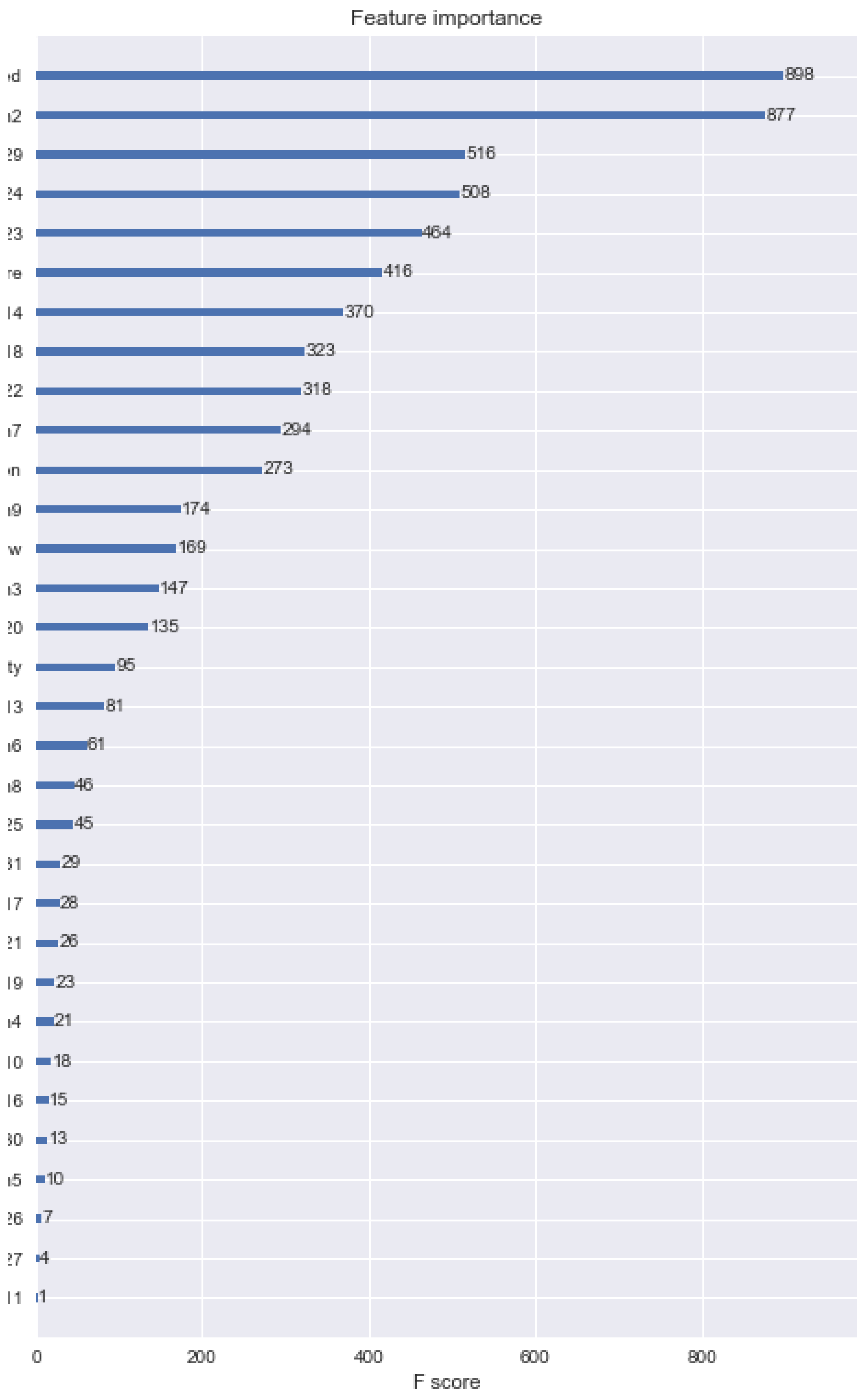


Figure 18: feature importance

…

Test were performed using the PCA to reduce feature dimensionality. The tested configurations were from 6 to 12 components, where 6 components already showed an explained variance of 98,6%. However, using the PCA the MSE from Table 5 dropped from 17% to 50% for the train/valid. set and from 15.6% to 18.5% for the test set. Hence the PCA optimizations was note pursued any further.

4.1.1.3. Short-term model performance

No tests were done for short-term model performance. It is estimated that the short-term model performance is equal or better than the mid-term performance. A very interesting application of the short-term model is presented in section 5.3.

4.2. Justification

The high R2 score reported in (Musango, 2016) could not be reached in the present work. This is assumed to be due to 2 main factors: First, the features used in the benchmark are different to the features used here (see Table 6). Second, one of the features used by (Musango, 2016), is the direct normal irradiation of the sun (DNI). In this work and in (Wolfertsstetter, 2016) the assumption is made, that the DNI helps explaining the high variance in the cleanliness raw data. Due to the measurement device working principle the raw data spread (shown in Figure 4) is indeed caused be the DNI. Thus, the DNI feature rather helps to explain the measurement device variance (high) than actual cleanliness variance (low, compared to the measurement variance).

Table 6: comparison of the (Musango, 2016) setup and the present setup

	Musango data set	**this data set**
Data set size:	n/a	270082 data points (used data)
preprocessing procedures	n/a	see section 3.1
splitting technique	n/a	see section 3.1
feature distributions	n/a	see section 2.2.1
number of features	5	35
features directly related to cleanliness / soiling	0	29 ccc channels
features indirectly related to cleanliness / soiling	3 (xxx1, xxx12, xx3)	4 (…)
R2 score (test set)	0.95	0.16
MSE (test set)	n/a	16 %

In order to compare the benchmark results with the ones presented here, the following aspects from Table 5 have to be taken into consideration: The number of data points used for training and testing is not reported. The data preprocessing is not detailed: No information on the data spitting- and cross validation technique is given, noise reduction is mentioned, but not explained (use of moving average, etc.) and no information on the feature distributions is given.

5. Conclusion

(approx. 1-2 pages)

5.1. Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

- *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

5.2. Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*
- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

5.3. Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

- *Are there further improvements that could be made on the algorithms or techniques you used in this project?*
- *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
- *If you used your final solution as the new benchmark, do you think an even better solution exists?*

- use XGBoost with NaNs
- achieve better normal normal distribution of the data, using Box-Cox
- use several years of data
- remove seasonality in multi-year data set
- improve outlier detection (Tukey method removes too many data points)
- improve the day-dict creation
- improve soiling rate calculation
- MDMS implementation

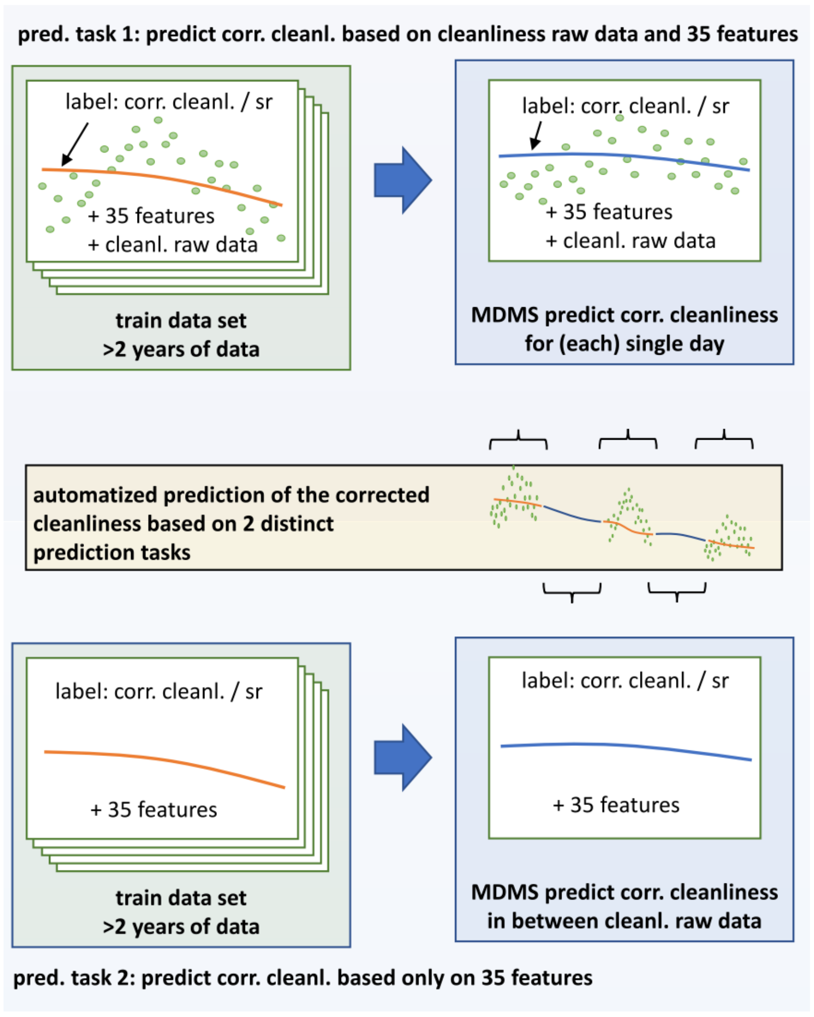

pred. task 1: predict corr. cleanl. based on cleanliness raw data and 35 features
label: corr. cleanl. / sr
+ 35 features
+ cleanl. raw data
train data set
>2 years of data
label: corr. cleanl. / sr
+ 35 features
+ cleanl. raw data
MDMS predict corr. cleanliness for (each) single day
automatized prediction of the corrected cleanliness based on 2 distinct prediction tasks
label: corr. cleanl. / sr
+ 35 features
train data set
>2 years of data
label: corr. cleanl. / sr
+ 35 features
MDMS predict corr. cleanliness in between cleanl. raw data
pred. task 2: predict corr. cleanl. based only on 35 features

Figure 19: automatization of MDMS creation of corrected cleanliness

6. Bibliography

(2014, 01 16). Retrieved from Highlighting Outliers in your Data with the Tukey Method: http://datapigtechnologies.com/blog/index.php/highlighting-outliers-in-your-data-with-the-tukey-method/

Brownlee, J. (2017). XGBoost Tuning. In J. Brownlee, *XGBoost With Python* (pp. 75-97).

Brownlee, J. (2017). *XGBoost with Python - Gradient Boosted Trees with XGBoost and scikit-learn.*

Musango, J. (2016, December). Measuring and modelling the influence of weather. Stellenbosch University.

scipy - Pearson coefficient. (2017, 08 25). Retrieved from https://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.stats.pearsonr.html

sckit-learn.org - mean squared error. (2017, 08 25). Retrieved from http://scikit-learn.org/stable/modules/model_evaluation.html

Udacity. (2017). *Udacity Machine Learning Nanodegree.* Retrieved from https://www.udacity.com/course/machine-learning-engineer-nanodegree--nd009

Wikipedia - mean squared error. (2017, 08 25). Retrieved from https://en.wikipedia.org/wiki/Mean_squared_error

Wolfertsstetter, F. (2016, January). Dr. *Effects of Soiling on Concentrating Solar Power Plants*. DLR e.V., German Aerospace Center, Almeria.