Author: Iva Eftimska

Date: 9.10.2023

Overview of three algorithms for forecasting

Examine the train and test data

The train set starts on **13/10/2020 10:20:00** and ends on **31/12/2022 23:55:00**. The test set starts on **1/1/2023 00:20:00** and ends on **31/5/2023 21:55:00**.

Table 1.1 and 1.2 show the number of samples in the raw/processed data, the missing values in the raw/processed data and the samples that fall outside the valid range for the training and test set respectively.

Raw data is when rows with a data validity of less than 32 (these values are not valid according to the user's needs) are not removed and values outside the valid range are also not removed.

Processed data here means that rows with a data validity of less than 32 are removed. Values outside the valid range are not removed here, only an additional column is displayed to see how many samples are outside the valid range in the data.

Valid values for **temperature are: [-40°C, 50°C]** and for **wind direction are: [0°, 360°]**. In both the processed and raw data, the duplicate values are removed in such a way that the last duplicate value is retained in order to have the same structure for comparison in the tables.

In Figure 1 you can see how many missing timestamps there are in the training and test data. The top row shows the missing timestamps for temperature, while the bottom row shows the missing timestamps for wind direction.

Missing values are filled with the previous non-missing value. The maximum range of missing days in our data is **4 days 2:30:00** (**21/08/2021 10:45:00 to 25-08-2021 13:10:00**).

Train data	Raw data	Processed data	Missing values in raw data	Missing values in processed data	Out of valid range
AMBIENT TEMPERATURE	229360	228497	6063	4659	0
WIND DIRECTION	229360	224558	6063	8597	0

Table 1.1 Raw, processed data, missing values and samples outside the valid range for train set

Test data	Raw data	Processed data	Missing values in raw data	Missing values in processed data	Out of valid range
AMBIENT TEMPERATURE	43011	43011	449	449	0
WIND DIRECTION	43011	39794	449	3666	0

Table 1.2 Raw, processed data, missing values and samples outside the valid range for test set

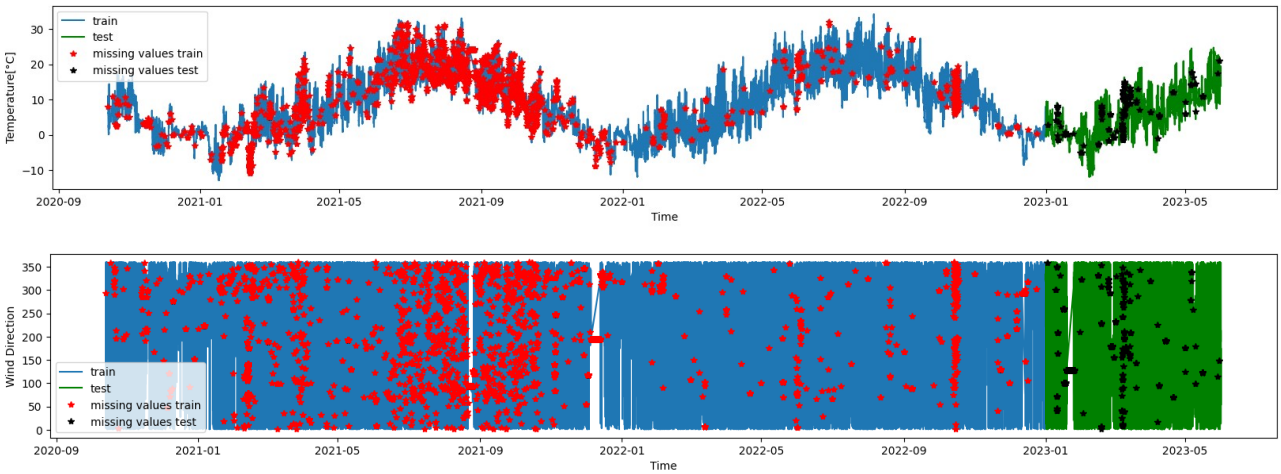

Figure 1: Missing values on train and test set

Building models

I used the mlforecast library, which implements a variety of regression algorithms and neural network algorithms that are compatible with the scikit-learn API. For all three algorithms (**RandomForestRegressor, XGBoost, LightGBM**) I used the recursive strategy for multi-step forecasting that means, a model is trained once and in the first iteration the model gives the first prediction, in the second iteration the previous predicted value is used as a feature for the prediction of the next value and so on, so that all future predictions are interdependent and the error accumulates.

To use the regression algorithms, the time series need to be transformed into a supervised learning problem, some features need to be selected as inputs and the target is the actual value at that exact time. First, I used the previous 24 lags (24*5 min) as features, but for the RandomForest regressor, only the first lag (5 min) was the most important and it kept repeating, so the predictions were constant. Then I used a function from AutoGluon that returns a list of lags, date features and seasonality depending on the frequency of the data. I used all these automatically generated features for each of the algorithms, also adding a target transformation to remove the seasonality returned by the function.

The RandomForest regressor cannot extrapolate if the data has a trend. So I thought that maybe seasonality could also mess up the RandomForest regressor and just removed the seasonality.

Hyperparameter optimisation is not used for any of the algorithms, so the default values are used.

Since the decision trees are not sensitive to scaling of the inputs, the features are not scaled for any of the algorithms.

All the automatically generated features I used are listed below in Table 2.

Date features	Lags	Seasonality
[minute of hour, hour of day, day of week, day of year]	[1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 22, 23, 24, 25, 26, 34, 35, 36, 37, 38, 287, 288, 289, 575, 576, 577, 863, 864, 865, 1151, 1152, 1153]	Lag 288

Table 2: Features used to train the model

I used the timestamps as indices to train the model. A regular training set was needed for all models, so I filled in the missing values using the ffill method (padded with the previous known value), but then when the model predicted some timestamps that were not present in the real test data, I simply removed them.

Missing data has some impact on the performance of the model because if I use, for example, the previous 24 lags (the previous 2 hours) and in those 2 hours some data is missing, i.e. 10:15, 10:20, 10:30, the model understands that the first lag of 10:30 is 10:20 and therefore uses that value for the lag and not the value of 10:25 which is missing.

I should also mention that it is worth doing manual feature engineering to see which features are suitable for particular models and to improve performance.

Table 3 shows the RAM and memory usage for training each model and predictions for 72 hours using only temperature as a single input and single output.

	RandomForest	XGBoost	LightGBM
RAM	5097 MB	517 MB	543 MB
Time	Approx. 8 min	Under 1 min	Under 1 min

Table 3: RAM and memory usage for training

Figure 2 shows the results for the 72 hour prediction. **The error is calculated by subtracting the actual test data from the predicted data.**

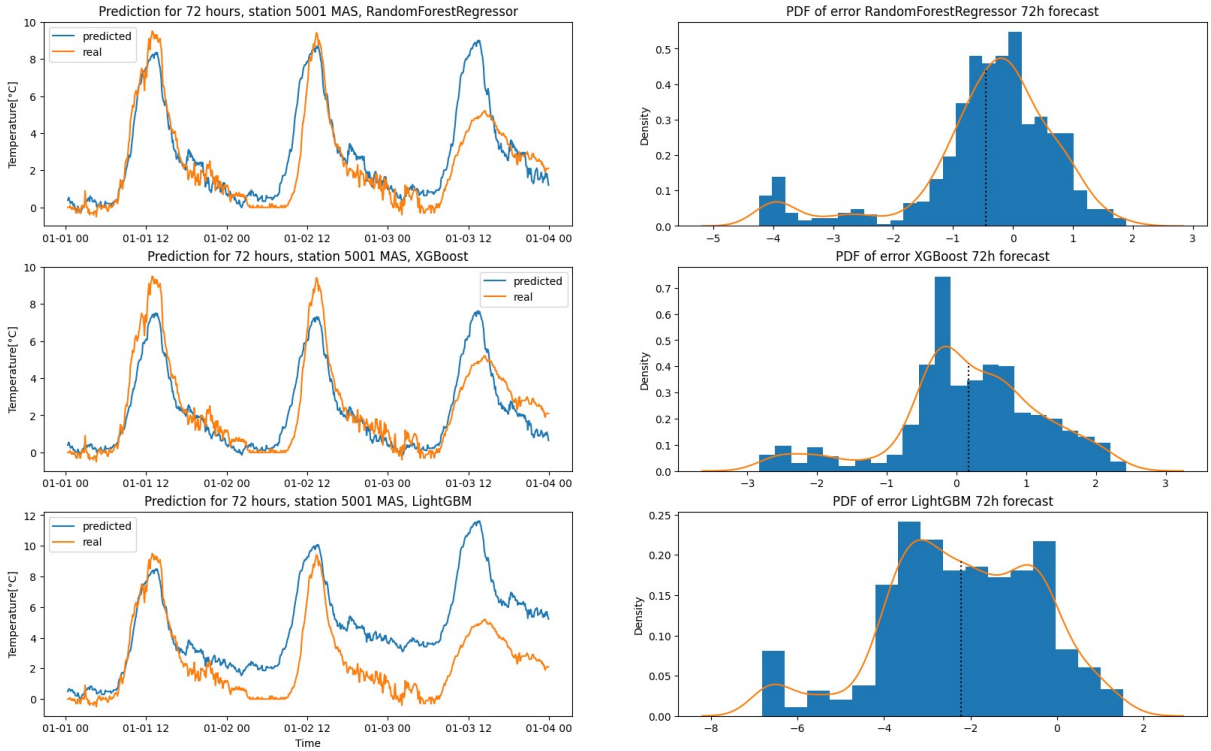

Figure 2: Forecasting 72 h for station 5001

We also tried different forecast horizons and the results are shown in Figure 3.

Here, the predictions for the 1 hour horizon are identical to the predictions for the first hour of 12 hour predictions since this is a recursive strategy for multi-step forecasting.

The single-input, single-output model is used all the time, with the only input parameter, the temperature.

The probability density function of the error for the different horizons is shown in Figure 4. **Again, the error is calculated by subtracting the real test data from the predicted data.**

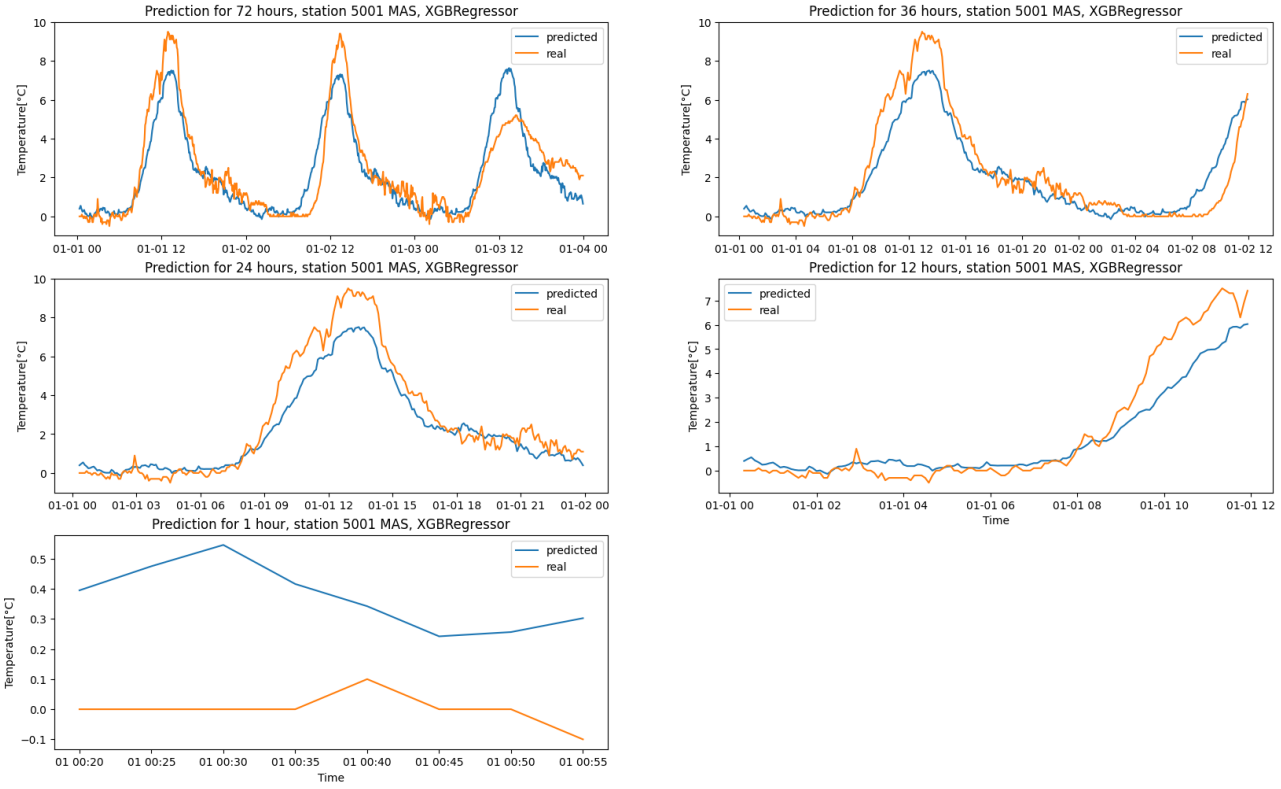

Figure 3: Different forecast horizons with XGBRegressor, station 5001

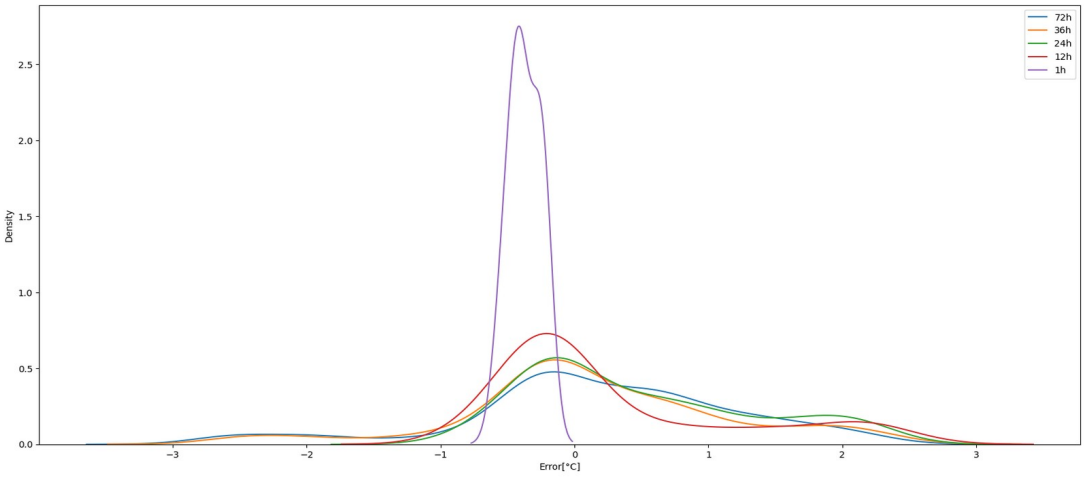

Figure 4: Probability density function of error for different forecast horizons

AutoGluon – AUTOML library

When I used AutoGluon, an AUTOML(Auto Machine Learning) library that automatically trains multiple models with extracted features, scales as needed, examines seasonality and trend, and removes them, the top three models for the 72-hour forecast results are shown in Figure 5. The ranking is from best to worst. Recursive Tabular is the same as LightGBM, they are gradient boosting methods.

	model	score_val	pred_time_val	fit_time_marginal	fit_order
0	WeightedEnsemble	-1.098399	13.303031	2.432689	12
1	RecursiveTabular	-1.121096	10.990061	95.321797	5
2	TemporalFusionTransformer	-1.220085	0.156522	1193.663618	9
3	DeepAR/T3	-1.267079	2.156449	590.039935	8
4	DeepAR/T1	-1.306425	2.129925	557.814489	6
5	AutoARIMA	-1.307831	219.565418	0.142409	11
6	DeepAR/T2	-1.343045	2.130736	553.854599	7
7	SeasonalNaive	-1.343703	2.890483	0.112233	2
8	Theta	-1.418068	13.964581	0.112226	3
9	AutoETS	-1.419983	13.215934	0.110342	4
10	Naive	-1.445472	3.604721	0.112055	1
11	PatchTST	-1.503644	0.155343	473.006383	10

	model	score_val	pred_time_val	fit_time_marginal	fit_order
0	WeightedEnsemble	-1.098399	13.303031	2.432689	12
1	RecursiveTabular	-1.121096	10.990061	95.321797	5
2	TemporalFusionTransformer	-1.220085	0.156522	1193.663618	9
3	DeepAR/T3	-1.267079	2.156449	590.039935	8
4	DeepAR/T1	-1.306425	2.129925	557.814489	6
5	AutoARIMA	-1.307831	219.565418	0.142409	11
6	DeepAR/T2	-1.343045	2.130736	553.854599	7
7	SeasonalNaive	-1.343703	2.890483	0.112233	2
8	Theta	-1.418068	13.964581	0.112226	3
9	AutoETS	-1.419983	13.215934	0.110342	4
10	Naive	-1.445472	3.604721	0.112055	1
11	PatchTST	-1.503644	0.155343	473.006383	10

Figure 5: AutoGluon results for 72 h forecasting

Perhaps these top 3 models are suitable for this task, but here we are only concerned with the prediction of temperature as a model with a single input and a single output, so that the time series is treated as a univariate time series.

Figure 6: Memory usage for direct strategy 72 h forecasting

When I tried to use the direct strategy for multi-step forecasting, in this case for the 72-hour forecast, 864 models were trained, each model giving a forecast, the first model predicts the next 5 minutes, the other the next 10 minutes, and so on.

After 5 days with the direct strategy, 252 GB RAM and about 500 GB SWAP were used, so the OS itself killed the process, but there are some options for distributed training and perhaps this is also worth trying. Figure 6 shows the memory usage when I tried the direct approach for multi-step forecasting as a single input, single output model.

AutoGluon does not support multivariate time series, but other libraries do. If I use all stations as a multivariate problem and try to predict two parameters at once, I don't know how much RAM and time the training will cost.

Data imputation

I must mention here that the end of the train data is 2022-31-12 23:55:00, then the model predicts from 2023-01-01 00:00:00 and onwards. Since we don't have values for 2023-01-01 00:00:00 to 2023-01-01 00:20:00 in the real test data, I just removed them to compare the predictions with the real test data.

For the 10-minute forecast, the model would give two values: 01-01 00:00 and 01-01 00:05. I did not use this horizon because I could not compare the test data (values we do not have) with the model predictions.

I could add another sample 2023-01-01 00:20:00 to the train and retrain again. Then the model would make further predictions based on this sample further, but this is not fully representative as we are changing our train data and the predictions would be biassed compared to other horizons as the train data does not match the other horizons.

It is possible to add an additional parameter to predict function, the parameter new_data, and the predictions will continue from the end of new_data, depending on the chosen prediction horizon. For example, if I want a

predicted value on 2023-01-01 00:25:00, I would add two samples, the last sample of training data and the first sample of test data (2023-01-01 00:20:00) in new_data and if I choose a horizon of one (5 minutes prediction), I get the predicted value on 2023-01-01 00:25:00.

I just need to find out if the size of new_data affects the quality of the predictions and if adding train samples to new_data can lead to overfitting, since the model was trained on this data and new_data should be unseen data. I think that I could not predict the value on 2023-01-01 00:20:00 because in new_data I should add samples before 2023-01-01 00:20:00 and we do not have these values.

There are several imputation strategies to fill in the missing data. Some algorithms have been developed for this task, you train a model and the model provides the values for the missing timestamps. The downside is that we can not compare whether the model's results are reasonable because we don’t have values to compare.

Another approach that might be tried would be if we just predict the temperature as a model with a single input and a single output. Since the temperature changes slowly, if there are gaps, i.e. missing data of only 5 minutes, we could fill them with the previous known value and fill gaps that are more than 5 minutes (or some other threshold) with the values provided by a trained model to fill the missing data.

An imputation strategy would be necessary to improve the performance of the model, especially if we go further and implement variants with multiple inputs and one output or with multiple inputs and multiple outputs, because we see that more data is missing for wind direction than for temperature, and all this missing data can cause the model not to learn well.

Future work

- I have to do a bit of manual feature engineering to find out which features are best suited for regression models. In some of the libraries there is an option to check online how each feature affects the performance of the model during training.
- I also need to find a solution to deal with missing data. If I ignore the missing data, I need to find a way to minimise how this affects the performance of the model.
- I would also like to see the performance of the model when the missing values are filled with a suitable trained model that gives outputs with filled missing values.
- I would also like to train a model for probabilistic forecasts that gives confidence intervals for each prediction, or somehow combine this model with the model for point forecasts.
- To find a way to combine several models, i.e the top three models given by AutoGluon, there is a way in one of the libraries to combine several models with some weights. You first give some initial values for these weights and then you can train a model that optimises the weights to get a better prediction result.

- I also want to try multiple-inputs single-output models, multiple-inputs multiple-output models, LSTM model or similar variants that provide all forecasts at once for multi-step forecasting. For example, in 1-hour forecasting, all 1-hour forecast values can be predicted at once without using a direct or recursive strategy for multi-step forecasting.
- To see if spatial correlation can affect the performance of the model, I can add some static features (that do not change in time) to the inputs, such as the location of the stations, and the model can give similar predictions for similar locations. **All libraries (mlforecast, skforecast, darts)** have this option implemented. I also want to see how the input parameters correlate with each other and how this affects the performance of the model.
- I need to review the literature to see what others have done so as not to repeat it and compare the performance of the model with them.
- The hyperparameter optimisation will be done for the selected model.
- If I have time, I would like to try Wind Topo for wind speed and wind direction.