Name: Simon Min Olafsson
Programme: Software Design
Course: Big Data Management Autumn 2023

Data Pipelines & Data Analytics Lifecycle

Forecasting the Wind Power Production in Orkney

Which steps (preprocessing, retraining, evaluation) does your pipeline include?

Handling of missing values:

To handle the missing values and what feature engineering to be utilized, I first used interpolation to handle the missing timestamps and creating linear values in context of merging with the power dataframe. One of the downsides by that is the interpolated values will then only follow assumed linear relationship between two datapoints, so for comparison matters, I also created the same pipeline runs on a dataset, where the power and wind dataframe was joined by an inner join for the last 500 days to have a dataset with more representative data.

Using dictionary instead of onehotencoding for categorization of the direction feature:

At first, I wanted to use onehotencoding in the preprocessing steps of the pipeline on the direction feature to scale and normalize the data to be more numerical related to the target feature, but I went with creating a dictionary instead. There were 16 unique directions in the dataset, so I categorized each direction from 1-16 and afterwards it was much easier to prompt the azure VM for predictions on integers in the wind direction instead of using onehotencoding.

Retraining steps & evaluation:

I chose SVR and LR models because of the mostly linear relationship between the feature columns and the target variable. The selected hyperparameters was chosen to tune and enhance each model performance in relation to the specific dataset. Using TimeSplitSeries, I was able to cross-validate each model through different splits and their metrics for each split. The best model is

determined based on the lowest evaluation scores with MLflow logging each iteration and metrics for each model evaluation and their respective hyperparameters.

What is the format of the data once it reaches the model?

The chosen features to predict power generation were "speed" and "direction" by assembling them into an input set X. This set represents the model's input features.

The target column "Total" for power generation was selected and stored as Y, as this is the variable to predict. The data format, before being fed into the model, is in a tabular format, specifically X holds a dataframe with columns for speed and direction, whereas y contains a series representing the target variable.

How did you align the data from the two data sources?

When running the two dataframes, wind and power, it's clear to see that there's a big gap between the timestamps in the wind dataframe compared to the power dataframe. It makes sense, that there is not a timestamp for every minute in the wind table, as the speed and direction doesn't change every minute, but in the context of merging the data from the two dataframes, it's a problem with so few timestamps in the wind data.

A viable option was to use interpolation to estimate the timestamps, that was missing in the timeframe compared to the power dataframe. Interpolation is a method to estimate missing values in a dataset by taking approximations that are calculated based on the existing data between two timestamps in the wind table. The accuracy of the interpolated values will be highly affected by only have one timestamp per every third hour available, and with linear interpolated values, those values between two datapoints are created with an assumption of straight-line relationship between those two points.

I created two dataframes to run on models on, one that contained the linear interpolated values and one that only contained the original timestamps in the power dataframe. In the latter, I increased the retrieval size to the last 500 days instead of 90 to ensure that the dataset was not too small. Those two were compared in the end of my mlflow project to see their performance and

prediction residual values, where it was clear that the model that was trained on the dataset with interpolated values performed better overall in the logged metrics. The size of the dataset was a huge factor, as the interpolated contained 17.000 rows whereas the one without interpolated values only contained 3800 rows.

time	ANM	Non-ANM	Total
2023-07-29 22:34:00+00:00	0.200271	3.124	3.324271
2023-07-29 22:35:00+00:00	0.207018	2.958	3.165018
2023-07-29 22:36:00+00:00	0.177399	2.938	3.115399
2023-07-29 22:37:00+00:00	0.188297	2.979	3.167297
2023-07-29 22:38:00+00:00	0.115458	2.872	2.987458
...	...	...	...
2023-11-26 22:29:00+00:00	11.955361	13.544	25.499361
2023-11-26 22:30:00+00:00	10.910618	12.811	23.721618
2023-11-26 22:31:00+00:00	10.903605	13.199	24.102605
2023-11-26 22:32:00+00:00	10.940333	13.250	24.190333
2023-11-26 22:33:00+00:00	10.612055	13.066	23.678055

time	Direction	Lead_hours	Source_time	Speed
2023-07-30 00:00:00+00:00	ESE	1	1690664400	3.12928
2023-07-30 03:00:00+00:00	ESE	1	1690675200	0.89408
2023-07-30 06:00:00+00:00	NW	1	1690686000	0.89408
2023-07-30 09:00:00+00:00	N	1	1690696800	0.89408
2023-07-30 12:00:00+00:00	N	1	1690707600	3.12928
...	...	...	...	...
2023-11-26 09:00:00+00:00	WNW	1	1700982000	5.81152
2023-11-26 12:00:00+00:00	WNW	1	1700992800	8.04672
...				
2023-11-26 18:00:00+00:00	N	1	1701014400	8.94080
2023-11-26 21:00:00+00:00	NNE	1	1701025200	12.07008

Figures: Power generation table (left) and the wind forecast table (right)

time	Speed	Direction	Total
2023-07-30 00:00:00+00:00	3.129280	3	2.192881
2023-07-30 00:10:00+00:00	3.005102	3	2.142532
2023-07-30 00:20:00+00:00	2.880924	3	2.166578
2023-07-30 00:30:00+00:00	2.756747	3	1.938750
2023-07-30 00:40:00+00:00	2.632569	3	1.881516
...	...	...	...
2023-11-26 20:20:00+00:00	11.374684	6	13.860357
2023-11-26 20:30:00+00:00	11.548533	6	14.401018
2023-11-26 20:40:00+00:00	11.722382	5	12.214954
2023-11-26 20:50:00+00:00	11.896231	5	11.126244
2023-11-26 21:00:00+00:00	12.070080	5	12.970429

Figure: Merged dataframe containing relevant columns for model training.

Name: Simon Min Olafsson
Programme: Software Design
Course: Big Data Management Autumn 2023

How did you decide on the type of model and hyperparameters?

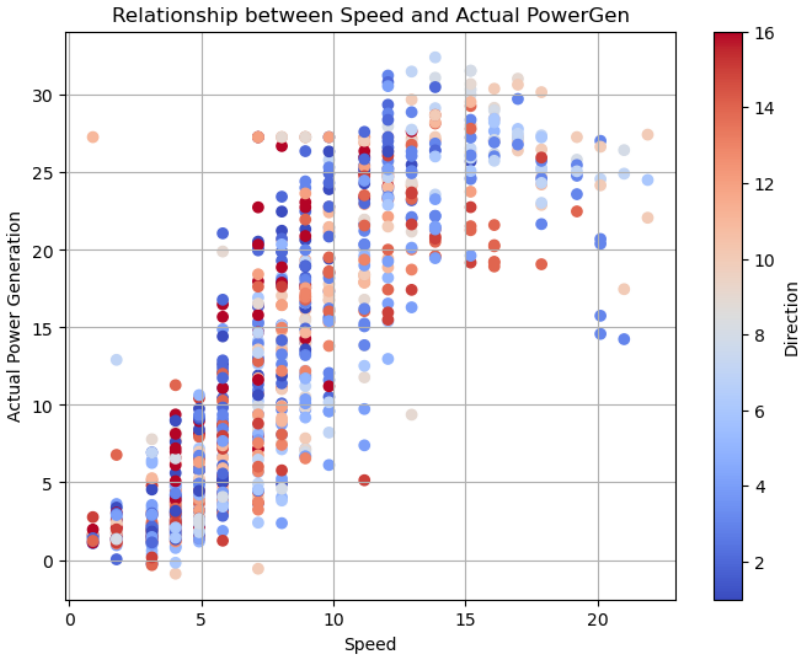


Figure: "Power Curve" – displays the relationship between wind speed and power generation

As displayed in the assignment description, the "power curve" displays the relationship between wind speed and power generation in a mostly linear fashion as well as some non-linear relationships. Therefore, it seemed reasonable to choose Linear Regression as a baseline model as the relationship between the feature columns and the target column is mostly linear. Another great model is Support Vector Regression, that is essentially very good at capturing non-linear relationships as well as it's hyperparameter options, where you are able to choose linear or radial basis function (great for capturing non-linear relationships between features and target variables)

In terms of which hyperparameters that were chosen, I chose to tune my LR model with:

- **Fit_intercept:** in linear regression context, the intercept is where the regression line crosses the y-axis. Let's say that the target variable power is above zero, even though speed and direction is zero, then it's good to fit the intercept to better capture starting point / baseline of generated power, as this could lead up to better accuracy.

- **Normalize:** changes the variables to have a similar scaling. It's useful when speed and direction are measured in different units or scales. Essentially equalizes each feature columns impact on the final result
- **Copy_X:** ensuring the original input data's integrity is not corrupted

Hyper parameters for the SVR:

- **Kernel:** as explained earlier, this parameter is great for capturing linear and non-linear relationships
- **C:** balances the model's prioritization between fitting the training data versus wider margin
- **Gamma:** determines the importance of a single training data sample.
- **Epsilon:** specification of tolerance where no penalty is given to errors
- **Shrinking:** Allows the use of certain heuristics in terms of skipping samples in training iterations
- **Max_iter:** if the convergence criteria is not met within this limit, then the process stops

How do you compare the newly trained model with the stored version?

To compare different models and different hyperparameters within the same model, there were different metrics that could be utilized. I found it suitable to use:

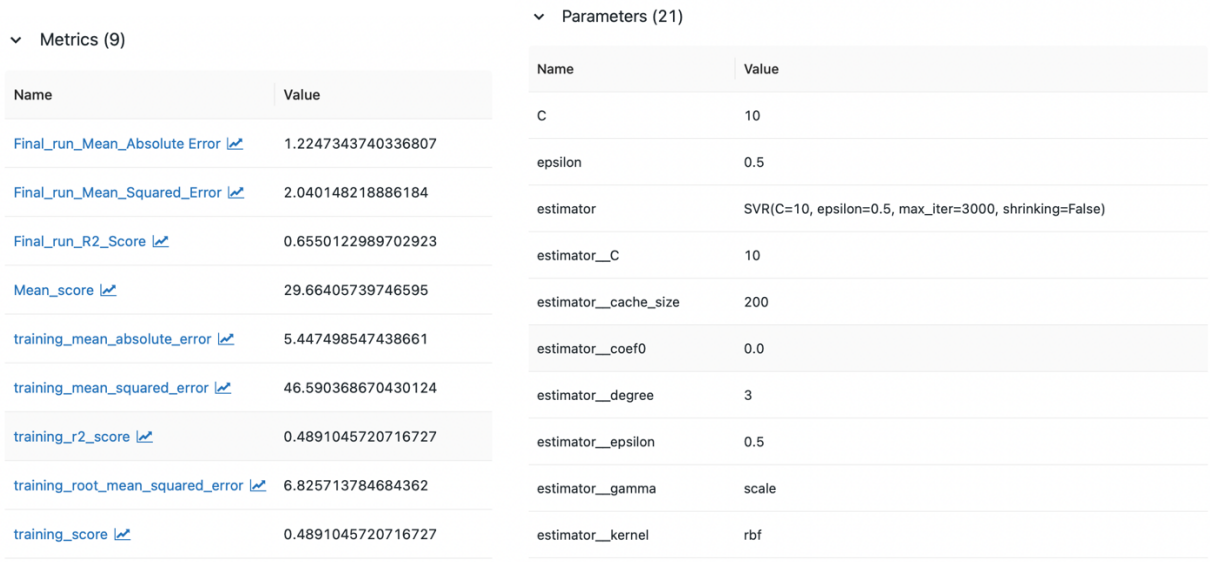
- **Mean Absolute Error:** Average absolute difference between predicted and actual values.
- **Mean Squared Error:** Average of squared differences between predicted and actual values.
- **R2 score:** how well the model fits the data on a scale from 0 to 1. Higher value indicates a higher precision and better explanation and prediction on the data by the model

For each run, all of these metrics were stored in a "Mean avg score" variable and kept in a list to keep track of which one had the lowest overall mean score. The model with the lowest overall score had a separate run with further estimation of these metrics to determine its precision on a bigger training set. Each metric was logged in MLflow and were evaluated by comparative analysis, hyperparameter tuning and understanding the model's behavior. This tracking and logging in MLflow made comparison and decision making easy, almost automatically.

Name: Simon Min Olafsson
Programme: Software Design
Course: Big Data Management Autumn 2023

```
mlflow.sklearn.log_model(best_model, f"{best_model_name}_best_model")  # Logging the best model
mlflow.log_params(best_params)  # Logging the best parameters
mlflow.log_metric("Mean_score", best_score)  # Logging the best score
```

Figure: after the best model I found – tell MLFlow to log its parameters



Metrics (9)

Name	Value
Final_run_Mean_Absolute Error	1.2247343740336807
Final_run_Mean_Squared_Error	2.040148218886184
Final_run_R2_Score	0.6550122989702923
Mean_score	29.66405739746595
training_mean_absolute_error	5.447498547438661
training_mean_squared_error	46.590368670430124
training_r2_score	0.4891045720716727
training_root_mean_squared_error	6.825713784684362
training_score	0.4891045720716727

Parameters (21)

Name	Value
C	10
epsilon	0.5
estimator	SVR(C=10, epsilon=0.5, max_iter=3000, shrinking=False)
estimator__C	10
estimator__cache_size	200
estimator__coef0	0.0
estimator__degree	3
estimator__epsilon	0.5
estimator__gamma	scale
estimator__kernel	rbf

Figure: MLFlow UI representation of the best models logged metrics and hyperparameters

How could the pipeline/system be improved?

Using a different cross-validation strategy such as GridSearch within the pipeline could've been interesting to see how much a performance optimization we would've got. There are also many different preprocessing options such as categorizing the "Speed" column into low, medium, and high speed etc. to see if the accuracy on the target variables would increase. There is also the option of using more different type of estimators such as Random Forest Regression or Gaussian Process Regression, that's especially great for time-series data.

How would you determine if the wind direction is a useful feature for the model?

To determine the relationship between the feature columns, but also between the feature and target columns. It's highly interesting to see if a change in one variable can cause or associate with changes in another. Therefore, I made a correlation matrix to see the strength and relationship

between the feature columns. You can see that direction generally have a very low influence on each of the other feature columns whereas speed highly affects the total power generated column.

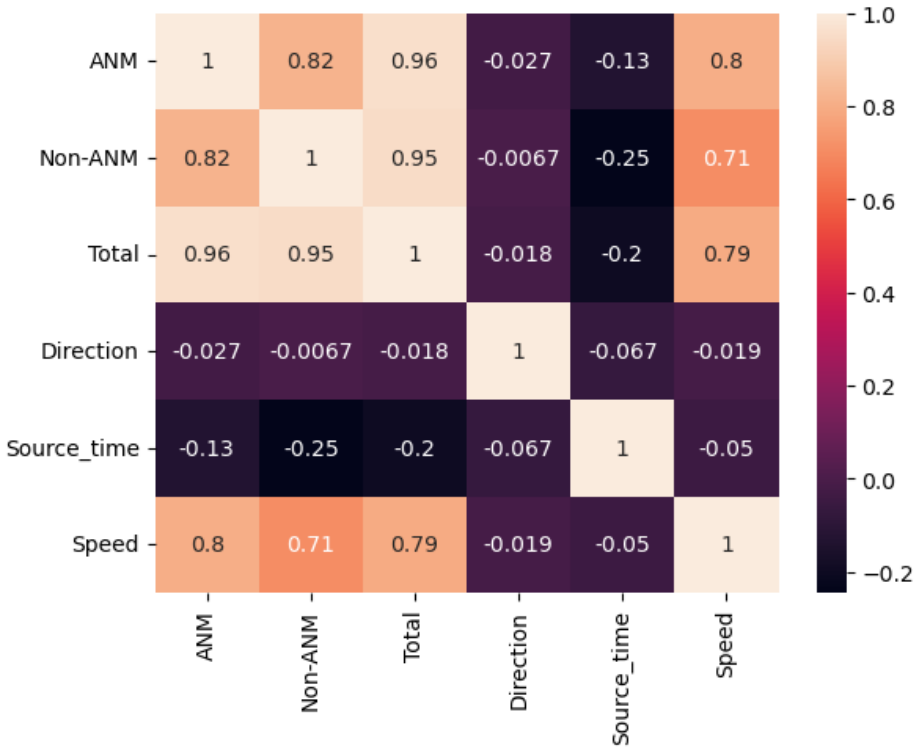


Figure: Correlation matrix of all the columns

How the evaluation errors change in terms of the cross-validation parameters?

The TimeSeriesSplit method divides the data into sequential parts for training and testing while the number of splits is set to 5 in my system. When the 5 splits have been run, the best model is found and that model is going to get trained on 80% of the original training set which changes the eval metrics dramatically, so the cross-validation parameters is crucial in the tuning process and how the model and its hyperparameters is going to get evaluated in the end. The different model's metric results change dramatically through some of the splits and hyperparameters, as some of them gets a negative R2-score which highly indicates overfitting and overall, very poor performance. When a negative R2 score is present, that model will get a penalty of 100 in the

overall score for that certain model and hyperparameters, so it won't be considered as a viable option.

```
# Calculating and storing the metrics for this fold
for metric_name, func, _ in metrics:
    score = (func(truth, predictions))  # Calculating metric score
    if score < 0 and metric_name == "R2":
        scores.append(100) #giving high penalty is R2 is below zero
    else:
        scores.append(score)
```

Figure: If the R2 score is below 0 – high penalty for that exact model and hyperparameter tuning

The advantages of packaging the experiments/models in the MLFlow formats and a comparison with other reproducibility options.

```
mlflow.sklearn.autolog()  # Enabling automatic logging for scikit-learn
mlflow.set_tracking_uri("http://127.0.0.1:5000")  # Setting MLFlow tracking URI to local host
mlflow.set_experiment("with interpolation")  # Setting the MLFlow experiment name
```

Figure: MLFlow experiment tracking and setting up the server to track the logging

By using MLflow, I was able to track and log every experiment, both with interpolated dataframes and non-interpolated for comparison reasons. The option to create a MLflow server for UI purposes is also a very easy way for tracking and model selection, including data visualization (metrics and parameters).

It's very easy to allow local packaging of saved models and runs in terms of reproducibility and integrity control as well as consistency in formats. When running cross validation series, it's also important to ensure capability and consistency in recording and managing all the experiment details, that facilitates the reproduction of these experiments. MLflow is great as a standardized logging framework to organize experiments, making it very easy to reproduce them.

There are many other reproducibility options like Jupyter Notebook, GIT, Docker, and they have very different ways of ensuring availability, consistency, packaging, compatibility options and

dependencies in different ways than MLflow, but for pipeline experiments, MLflow is a very viable solution.

What are the main reasons to deploy your model, making it available for others?

As machine learning is a power hungry, CPU intensive task with big datasets, its often a great advantage to be able to use remote computers for deployment, usually referred to as “the cloud”. The benefits are that its already pre-trained models for certain task, where big data analysis can be utilized on different datasets quickly for companies. The cloud allows us to scale our machine learning projects up and down as needed. The cloud access generally speeds up the machine learning lifecycle and it’s an inexpensive solution as you’re only paying for consumption of the remote machines. Below is shown how I deployed my model after installing MLFlow on my azure VM and how I can send objects with specific feature columns as wind and direction to get a prediction of power generation immediately.

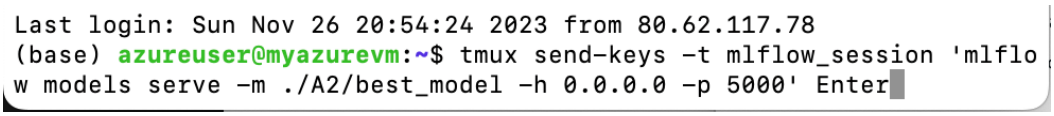


Figure: how I enabled MLFlow to run the model in the cloud using tmux

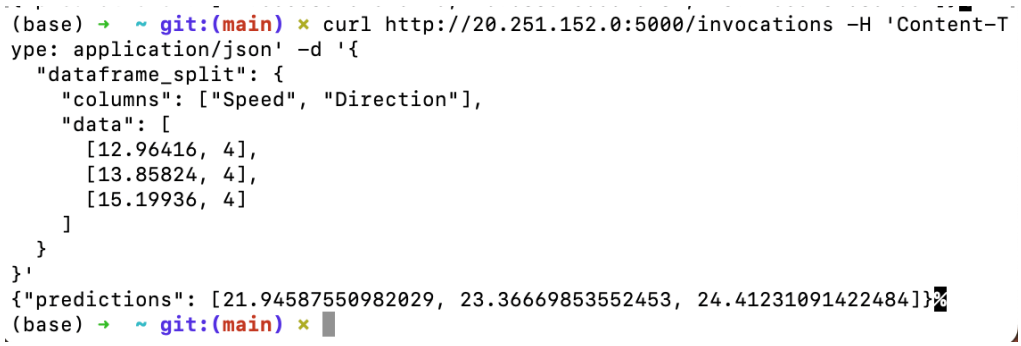


Figure: how to get predictions from the cloud VM on certain wind speed and wind directions from my host terminal using an object input format