Big Data Management

Assignement 2 - Forecasting the Wind Power Production

Matteo Mormile | matmo@itu.dk | GitHub repository

23 November 2023

Contents

1 Introduction

Starting with the power produced by a wind farm in the Okrney Islands and the weather conditions in the area, the goal of the analysis conducted is to identify and develop a machine learning model that can predict power production based on weather forecasts.
To do this we will go through several steps including:

- the analysis and visualization of available data
- the identification of models that could meet our requirements
- the training of models with different parameters (we will use MLflow)
- the validation of the models
- the release of a prediction model that is available to everyone

2 The Data

2.1 Get and merge the data

We will work with data from two main sources:

1. **Renewable Power Generation in Orkney**: Sourced from Scottish and Southern Electricity Networks (SSEN), this data is stored in InfluxDB. The "Generation" measurement includes parameters like time of measurement and total renewable power generation in MegaWatts.
2. **Weather Forecasts for Orkney**: Obtained from the UK MetOffice, forecasts are stored in the "MetForecasts" measurement in InfluxDB. Key elements include target time, wind speed, wind direction, source time, and forecast horizon.

Once we obtain the data in our code by querying the InfluxDB database, with a SQL-like language, we can immediately see that the sampling rate is different. If the power detection occurs every minute, the weather forecast, on the other hand, has a 3-hour gap.
To do a more complete analysis in addition to doing a simple join between the two datasets we will also try to analyze a dataset obtained by joining the weather forecast with the average energy produced over the last 3 hours instead of the point energy (*joined_ df* and *mean_ df* respectively).

2.2 Visualize the data

We use some graphs to help us better understand the relationship between speed and energy produced, and as a result, determine which ML methods are optimal for this particular scenario. Specifically, we observe that the energy produced and wind speed form a sigmoid relationship. Additionally, we observe that direction affects energy production. This is evident in the first graph, where the average energy produced varies when comparing similar velocity averages, as well as in the following graphs, where direction significantly affects energy production even at low speeds. We can see from the data that including direction in our model is beneficial.

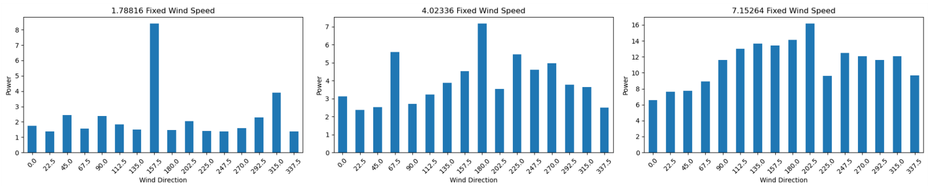


Figure 1: Direction also influences the energy produced

3 Identificate our models

Based on the relationship that emerged between speed and generated power, we identified 4 possible machine learning algorithms that might best fit our problem. Specifically in the case of polynomial regression we will evaluate with and without direction. In this case, the models have fixed parameters; however, we will subsequently alter them and use MLflow to plot the results. The following algorithms are then used:

- **Polynomial Regression without Direction**: The basis of this model is polynomial regression, a kind of linear regression in which an nth-degree polynomial is used to model the relationship between the independent variable (input) and the dependent variable (output). We will create and evaluate this model with two different data sets, one with simple join and the other averaging the energy over the 3 hours.
- **Polynomial Regression with Direction**: By converting cardinal points to degrees, directional information is integrated. This addition makes it possible for the model to account for directionality in its forecasts, improving its capacity to represent more complex relationships.
- **Random Forest Regressor**: During training, this learning technique creates several decision trees and combines their predictions to produce more reliable and accurate outcomes. It can handle non-linear relationships and automatically weighs the relevance of features.
- **Multi-layer Perceptron Regressor**: The multi-layer perceptron (MLP) regressor, which is based on a neural network design, is made up of several layers of nodes (neurons) and trains using backpropagation.

By using the matplotlib phyton library we can have an example of how the models work. By dividing the dataset into training and test data we can observe in blue the actual energy values while in red those predicted by the different models.

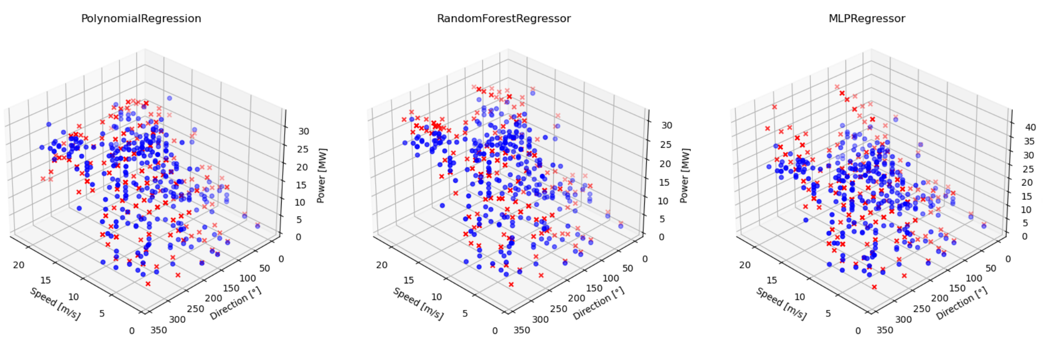


Figure 2: Result prediction of models that also use direction

4 Tracking our experiments with MLFlow

By using this system, however, we realize how our algorithms need many parameters to build the models, parameters that we are not certain about and that we must modify to obtain the best model (such as the degree of the polynomial, the depth of the tree, or the number of levels). To compare our models, we will use the MLFlow library, which allows us to save them for later comparison of error values on predictions.
In particular, we will keep track of these error:

- **AE (Absolute Error)**: represents the absolute difference between the model predictions and the actual values in your dataset.
- **MSE (Mean Squared Error)**: the average of the squared differences between the model predictions and the actual values. A lower MSE indicates higher model accuracy.
- **RMSE (Root Mean Squared Error)**: is a measure of the standard deviation of the model errors. It is calculated as the square root of MSE.
- **R² (Coefficient of Determination)**: represents the percentage of variation in the dependent variable that can be explained by the independent variables in the model. A value closer to 1 indicates a good fit of the model to the data.

To use MLflow, we need to first start a server instance using the command:

```
mlflow server --host 127.0.0.1 --port 5000
```

Once this is done, in our Python code, before initiating the model creation and data prediction, we will connect to the server and enable autologging of parameters and metrics for our models. After doing this, it's sufficient to create our models with different parameters (such as polynomial degree or maximum number of layers), grouping them by the type of ML algorithm. Once this is done, by connecting to the MLflow server (`127.0.0.1:5000`), we can find the results of your runs along with their respective error values.

5 Compare the models

By submitting the two different datasets to the algorithm that does not use wind direction, we observe that the one obtained from the direct join, without using the mean, turns out to be the one that scored better. We then decide to use only this dataset for the other more complicated models.

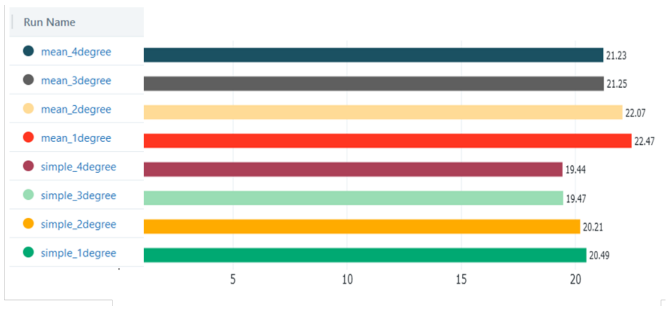
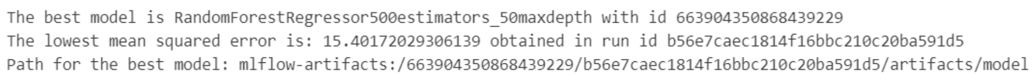


Figure 3: Polynomial Regression without direction on 2 different dataset

By querying the mlflow server via code (we can also use graphs as before) we can observe how, among the runs of the models that consider direction, the Random Forest-based model was the best one.

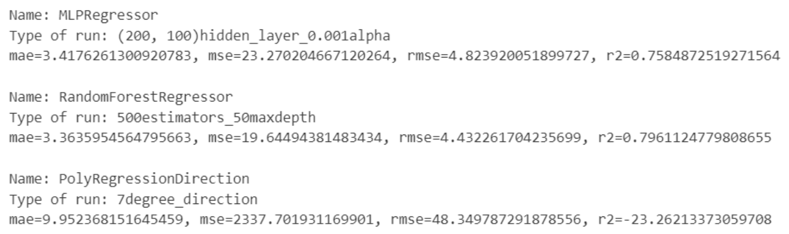


```
The best model is RandomForestRegressor500estimators_50maxdepth with id 663904350868439229
The lowest mean squared error is: 15.40172029306139 obtained in run id b56e7caec1814f16bbc210c20ba591d5
Path for the best model: mlflow-artifacts:/663904350868439229/b56e7caec1814f16bbc210c20ba591d5/artifacts/model
```

Figure 4: Model with the lowest mean squared error

5.1 Validation and overfitting problem

For additional model confirmation, we decided to use a subset of data not present in either the training or testing set to make new predictions and recalculate the errors. In particular, the models used for these predictions are the best model for each of the 3 algorithms including direction.
Analyzing the results, we observe that RandomForest once again proves to be the best model. MLPRegressor also performs well, but we observe significant errors with Polynomial Regression.



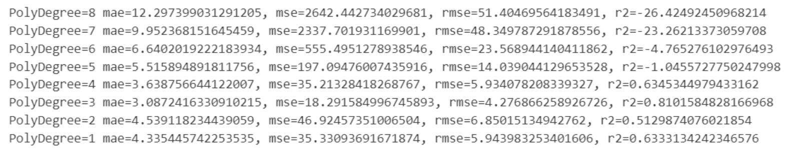
```
Name: MLPRegressor
Type of run: (200, 100)hidden_layer_0.001alpha
mae=3.4176261300920783, mse=23.270204667120264, rmse=4.823920051899727, r2=0.7584872519271564

Name: RandomForestRegressor
Type of run: 500estimators_50maxdepth
mae=3.3635954564795663, mse=19.64494381483434, rmse=4.432261704235699, r2=0.7961124779808655

Name: PolyRegressionDirection
Type of run: 7degree_direction
mae=9.952368151645459, mse=2337.701931169901, rmse=48.349787291878556, r2=-23.26213373059708
```

Figure 5: Model validation on new data

Examining the models of this algorithm, we notice that in this case, decreasing the polynomial degree leads to a decrease in prediction error.
This event can be identified as an overfitting error that occurs when a model becomes overly specific to the training data, essentially memorizing it, instead of learning the underlying patterns that can be applied to new examples.

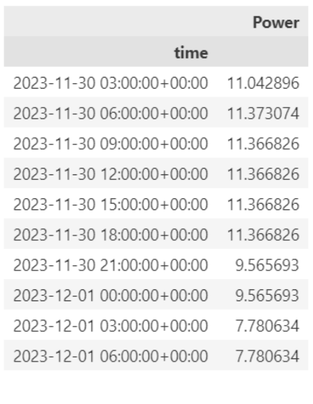


```
PolyDegree=8 mae=12.297399031291205, mse=2642.442734029681, rmse=51.40469564183491, r2=-26.42492450968214
PolyDegree=7 mae=9.952368151645459, mse=2337.701931169901, rmse=48.349787291878556, r2=-23.26213373059708
PolyDegree=6 mae=6.6402019222183934, mse=555.4951278938546, rmse=23.568944140411862, r2=-4.765276102976493
PolyDegree=5 mae=5.515894891811756, mse=197.09476007435916, rmse=14.039044129653528, r2=-1.0455727750247998
PolyDegree=4 mae=3.638756644122007, mse=35.21328418268767, rmse=5.934078208339327, r2=0.6345344979433162
PolyDegree=3 mae=3.0872416330910215, mse=18.291584996745893, rmse=4.276866258926726, r2=0.8101584828166968
PolyDegree=2 mae=4.539118234439059, mse=46.92457351006504, rmse=6.85015134942762, r2=0.5129874076021854
PolyDegree=1 mae=4.335445742253535, mse=35.33093691671874, rmse=5.943983253401606, r2=0.6333134242346576
```

Figure 6: Overfitting problem

6 Make prediction

After validating our model, we can finally use it to make predictions on future energy production values. To do this, we just need to retrieve our model and provide it with the weather forecast values for the upcoming days, obtaining the predicted energy production values.



	Power
time	
2023-11-30 03:00:00+00:00	11.042896
2023-11-30 06:00:00+00:00	11.373074
2023-11-30 09:00:00+00:00	11.366826
2023-11-30 12:00:00+00:00	11.366826
2023-11-30 15:00:00+00:00	11.366826
2023-11-30 18:00:00+00:00	11.366826
2023-11-30 21:00:00+00:00	9.565693
2023-12-01 00:00:00+00:00	9.565693
2023-12-01 03:00:00+00:00	7.780634
2023-12-01 06:00:00+00:00	7.780634

Figure 7: Using the saved model to predict energy production

7 Deploy the model on remote machine

However, using our model in this way doesn't make it available to others. To do that, we will use a virtual machine created through Microsoft Azure to serve our model. The steps to follow are:

- Create a virtual machine on Microsoft Azure, paying attention to the key pair that we will use to connect via SSH.
- Enable traffic on the port we will use to serve the model
- Connect via SSH protocol, install the conda package manager on the machine, and create a new environment with the required packages.

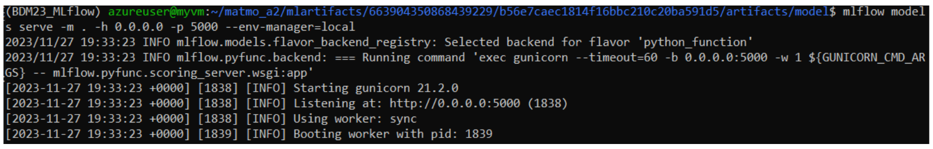
```
ssh -i /home/matteo/.ssh/id_rsa azureuser@51.120.121.101

(BDM23_MLflow) azureuser@myvm:~$
```

- Clone the repository on VM from GitHub

```
git clone https://github.itu.dk/BDM-Autumn-2023/matmo_a2.git
```

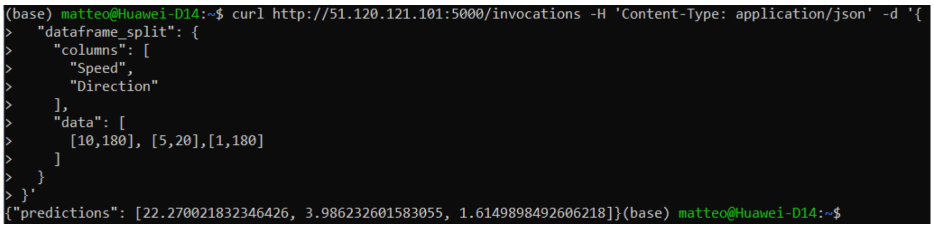
- Move inside the folder that contains the model specification and serve it. We will use the one indicated as the best:
  /matmo_a2/mlartifacts/663904350868439229/b56e7caec1814f16bbc210c20ba591d5 /artifacts/model



```
(BDM23_MLflow) azureuser@myvm:~/matmo_a2/mlartifacts/663904350868439229/b56e7caec1814f16bbc210c20ba591d5/artifacts/model$ mlflow model
s serve -m . -h 0.0.0.0 -p 5000 --env-manager=local
2023/11/27 19:33:23 INFO mlflow.models.flavor_backend_registry: Selected backend for flavor 'python_function'
2023/11/27 19:33:23 INFO mlflow.pyfunc.backend: === Running command 'exec gunicorn --timeout=60 -b 0.0.0.0:5000 -w 1 ${GUNICORN_CMD_AR
GS} -- mlflow.pyfunc.scoring_server.wsgi:app'
[2023-11-27 19:33:23 +0000] [1838] [INFO] Starting gunicorn 21.2.0
[2023-11-27 19:33:23 +0000] [1838] [INFO] Listening at: http://0.0.0.0:5000 (1838)
[2023-11-27 19:33:23 +0000] [1838] [INFO] Using worker: sync
[2023-11-27 19:33:23 +0000] [1839] [INFO] Booting worker with pid: 1839
```

Figure 8: Serve the model on 51.120.121.101:5000

- We can then remotely query our model to get predictions about the power output of our wind farm.



```
(base) matteo@Huawei-D14:~$ curl http://51.120.121.101:5000/invocations -H 'Content-Type: application/json' -d '{
>   "dataframe_split": {
>     "columns": [
>       "Speed",
>       "Direction"
>     ],
>     "data": [
>       [10,180], [5,20],[1,180]
>     ]
>   }
> }'
{"predictions": [22.270021832346426, 3.986232601583055, 1.6149898492606218]}(base) matteo@Huawei-D14:~$
```

Figure 9: Query the model using the curl command

So we have seen how, starting from the wind farm data we were able to analyze several ML models, choose the best one and make it public available but also with the ability to easily recreate it on another machine using the MLflow project. However, the development cycle of an ML model does not stop there, once deployed it must also be monitored so that the results are always reliable.