

Assignment 1: Forecasting Pipelines

Forecasting the Wind Power Production in Orkney

September, 2022

1 INTRODUCTION

In this assignment you will design and implement a retraining pipeline (in the non-interleaved sense of the word) - including data fetching, preprocessing, evaluation and model storing - for a small wind energy forecasting system using `sklearn`. You'll be using data from Orkney, an archipelago off the north eastern coast of Scotland. Orkney has around 20,000 inhabitants and they produce around $\sim 120\%$ of their annual energy consumption in wind energy.

This document will give you an extremely brief intro to wind power forecasting, describe the data you will be working with, list the requirements for the assignment and provide suggested reading and useful links. This document also includes the `template.py` file to help you get started.

1.1 Wind power forecasting

As you might expect, the estimated wind speed is the primary input to any model that seeks to estimate wind power production. If we make a scatter plot of wind speed against power output, it might look like [Figure 1.1](#). If you squint at the data, you'll see a sigmoid-ish curve, called the "power curve". The power curve is essential for wind power forecasting, as it describes the relationship between wind speeds and power output.

But the wind might meet different environments, depending on which direction the wind is coming from, affecting the wind speed or creating turbulence. So we should probably include the wind direction as a feature in our model, increasing the dimensionality of the input.

The wind direction is naturally circular and is currently encoded as a string in our data. However, our models only accept numerical input, so we need to process the data to extract useful features. We can either map the text string to degrees or radians, or encode them as a categorical (or "one-hot") vector. This is up to you to decide.

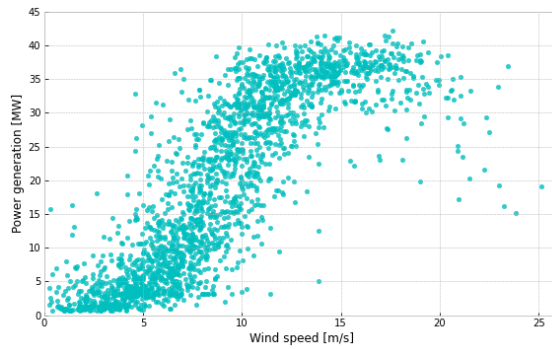


Figure 1.1: Wind speed plotted against power generation, showing a "power curve" for Orkney

2 THE DATA

You will be working with data from two sources:

1. Orkney's renewable power generation - Sourced from [Scottish and Souther Electricity Networks \(SEN\)](#)
2. Weather forecasts for Orkney- Sourced from the [UK MetOffice](#)

The data is stored in an [InfluxDB](#), which is a non-relational time-series database. InfluxDB can be queried using [InfluxQL](#), a "SQL-like" query language for time-series data. InfluxDB does not have tables with rows and columns, instead data is stored in *measurements* with *fields* and *tags*¹.

2.1 Renewable energy generation

The power generation data is sampled every minute from a public website and is stored in the measurement **Generation** with the following "schema":

```

1 name: Generation
2 Key      Type
3 -----
4 time     float    (Time of measurement)
5 ANM      float    (Not relevant for this assignment)
6 Non-ANM  float    (Not relevant for this assignment)
7 Total    float    (Renewable power generation in MegaWatts)
```

2.2 Weather forecasts

The weather forecasts are sourced from the UK governmental weather service, the MetOffice. For forecasts we talk about **Source time**, the time at which the forecast was generated, **target time**, the time that is forecasted, and **lead time** or **forecast horizon**, the difference between the source time and the target time. There are 3 hours between each timestamp.

The forecasts are stored in the measurement **MetForecasts** with the following "schema":

```

1 name: MetForecasts
2 Key      Type
3 -----
4 time     float    (Target time of forecasts)
5 Speed    float    (Wind speed in M/S)
6 Direction string    (Wind direction, e.g. "S" or "NW")
7 Source_time integer  (Time of forecast generation)
8 Lead_hours string    (Forecast horizon in hours, actually a tag)
```

¹Tags are always strings and are indexed, while fields are not indexed

3 REQUIREMENTS AND HAND-IN

3.1 System requirements

For this assignment you should create the data retraining pipeline, including an `sklearn` Pipeline for preprocessing, in a system that:

- Reads the latest data from the InfluxDB
- Prepares the data for model training, including
 - Aligning the timestamps of the two data sources (e.g. through resampling or an inner join)
 - Handling missing data
 - Altering the wind direction to be a usable feature (by mapping to radians, encoding as categorical, or other)
 - Scaling the data to be within a set range
- Trains a (few) regression model(s) of your choice, (e.g. `LinearRegression`)
- Saves the best performing model to disk

Once you have saved a model, your work is not done just yet. In a production setting, models should not be static, but kept up to date, once new and possibly better training data appears in the database. Imagine creating and saving a model today, and perhaps in a few weeks, you run your code again and train a new model. Is it better than the first model? To answer that you should also:

- Compares the newly trained model with the currently saved model, and picks the best performing model.
- Saves the best performing model to disk (decide if you think deleting the old is a good idea)
- Uses the current best model to forecast future (wind) power production

Template code for the data fetching is provided in `template.py`, so you primarily need to work on preprocessing, training, evaluation and storage.

3.2 Hand-in

You should hand in a report describing your solution, including which design choices and trade-offs you made. We value concise and well formulated arguments with supplementary code examples. Here are some questions for inspirations:

- Which steps does your (preprocessing, retraining, evaluation) pipeline include?
- What is the format of the data once it reaches the model?
- How did you align the data from the two data sources?
- How did you decide on the type of model and hyperparameters?
- How do you compare the newly trained model with the stored version?
- How could the pipeline/system be improved?
- How would you determine if the wind direction is a useful feature for the model?
- Currently the system fetches data for the previous 90 days worth of data:
 - How would you determine if this is a good interval?
 - What are the trade-offs when deciding on the interval?
 - Would accuracy necessarily increase by including more data?

The maximum length for the report handed in is five pages.

3.3 Suggested reading and useful links

- Page 66-76 of [Hands-On Machine Learning](#), especially the difference between estimators, transformers and predictors.
- [User guide for sklearn pipelines](#)
- [sklearns Pipeline API](#)

3.4 Notes and hints

The questions in this section are optional

- How you encode the wind direction is up to you. Some possible options are:
 - [OneHotEncoding](#)
 - Transforming to degrees or radians with a [custom transformer](#)
 - Converting speed and direction to a 2-dimensional vector: [Explanation](#), [example](#).
- From the `sklearn` documentation on [Pipelines](#): *All estimators in a pipeline, except the last one, must be transformers (i.e. must have a transform method). The last estimator may be any type (transformer, classifier, etc.).* This means that you can have your model (predictor) as the last step in your pipeline and have whole process from new data to prediction in one object with the `predict`-method. This is also useful when storing and loading trained models.
- The power curve in [Figure 1.1](#) does not look very linear, does it? You can address this by using a non-linear model, or by adding a [PolynomialFeatures](#) transformer to the wind speed input. Which approach gives you the best results? And which degree of polynomial would you use?
- If you want to apply different transformations to different columns (e.g. categorical and numerical), you can use the [ColumnTransformer](#).
- Note that this is *time series data*, which means that it is not `iid`. You need to take this into account when splitting the data. You can either use a non-shuffled split, or something like [TimeSeriesSplit](#).
- Training and evaluating sequence-oblivious models is simpler, but there might be some accuracy to gain in using a sequence prediction model, like [ARIMA](#), or a [Recurrent Neural Network](#).

4 CODE TEMPLATE

```
1 #####
2 ## Getting the data
3
4 from influxdb import InfluxDBClient # install via "pip install influxdb"
5 import pandas as pd
6
7 client = InfluxDBClient(host='influxus.itu.dk', port=8086, username='lsda',
8                          password='icanonlyread')
9 client.switch_database('orkney')
10
11 def get_df(results):
12     values = results.raw["series"][0]["values"]
13     columns = results.raw["series"][0]["columns"]
14     df = pd.DataFrame(values, columns=columns).set_index("time")
15     df.index = pd.to_datetime(df.index) # Convert to datetime-index
16     return df
17
18 # Get the last 90 days of power generation data
19 generation = client.query(
```

```

19     "SELECT * FROM Generation where time > now()-90d"
20     ) # Query written in InfluxQL
21
22 # Get the last 90 days of weather forecasts with the shortest lead time
23 wind = client.query(
24     "SELECT * FROM MetForecasts where time > now()-90d and time <= now()
25     and Lead_hours = '1'"
26     ) # Query written in InfluxQL
27
28 gen_df = get_df(generation)
29 wind_df = get_df(wind)
30
31
32 #####
33 ## Preprocess the data / Compose your pipeline
34
35 from sklearn.pipeline import Pipeline
36
37 # Align the data frames
38
39 pipeline = Pipeline([
40     # Here you can add your preprocessing transformers
41     # And you can add your model as the final step
42 ])
43
44 # Fit the pipeline
45
46 # Load stored model and compare with newly trained
47 # and store the best one
48
49 #####
50 ## Do forecasting with the best one
51
52 # Get all future forecasts regardless of lead time
53 forecasts = client.query(
54     "SELECT * FROM MetForecasts where time > now()"
55     ) # Query written in InfluxQL
56 for_df = get_df(forecasts)
57
58 # Limit to only the newest source time
59 newest_source_time = for_df["Source_time"].max()
60 newest_forecasts = for_df.loc[for_df["Source_time"] == newest_source_time].
61     copy()
62
63 # Preprocess the forecasts and do predictions in one fell swoop
64 # using your best pipeline.
65 pipeline.predict(newest_forecasts)

```