PREDICTING WIND ENERGY PRODUCTION WITH SCIKIT-LEARN

• INTRODUCTION

Nowadays, electricity networks of advanced countries rely more and more in non-operable renewable energy sources, mainly wind and solar. However, in order to integrate energy sources in the electricity network, it is required that the amount of energy to be generated to be forecasted 24 hours in advance, so that energy plants connected to the electricity network can be planned and prepared to meet supply and demand during the next day (For more details, check "Electricity Market" at Wikipedia).

This is not an issue for traditional energy sources (gas, oil, hydropower, …) because they can be generated at will (by burning more gas, for example). But solar and wind energies are not under the control of the energy operator (i.e. they are non-operable), because they depend on the weather. Therefore, they must be forecasted with high accuracy. This can be achieved to some extent by accurate weather forecasts. The *Global Forecast System* (GFS, USA) and the *European Centre for Medium-Range Weather Forecasts* (ECMWF) are two of the most important Numerical Weather Prediction models (NWP) for this purpose.

Yet, although NWP's are very good at predicting accurately variables like "100-meter U wind component", related to wind speed, the relation between those variables and the electricity actually produced is not straightforward. Machine Learning models can be used for this task.

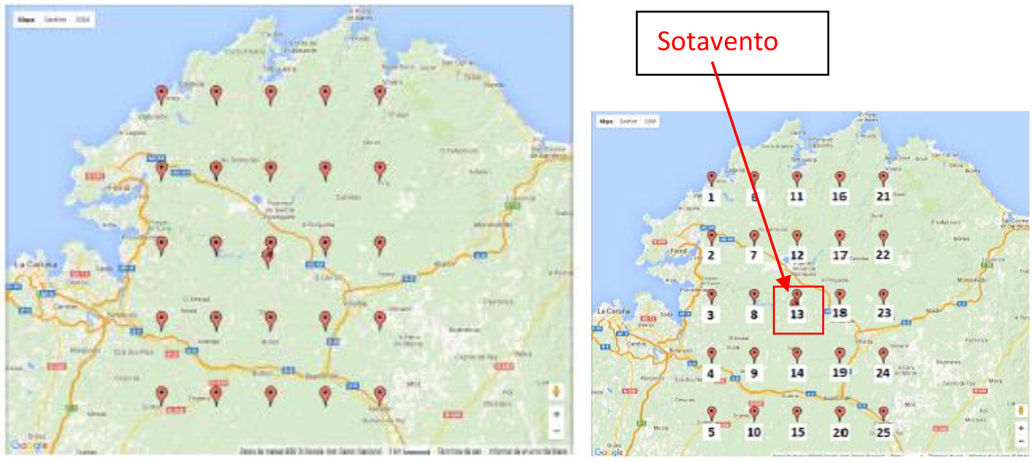
In particular, we are going to use meteorological variables forecasted by ECMWF (http://www.ecmwf.int/) as input attributes to a machine learning model that is able to estimate how much energy is going to be produced at the Sotavento experimental wind farm (http://www.sotaventogalicia.com/en).

Sotavento wind farm.

More concretely, we intend to train a machine learning model *f*, so that:

- Given the 00:00am ECMWF forecast for variables $A_{6:00}$, $B_{6:00}$, $C_{6:00}$, … at 6:00 am (i.e. six hours in advance)
- $f(A_{6:00}, B_{6:00}, C_{6:00}, \ldots)$ = electricity generated at Sotavento at 6:00

We will assume that we are not experts on wind energy generation (not too far away from the truth, actually). This means we are not sure which meteorological variables are the most relevant, so we will use many of them, and let the machine learning models and attribute selection algorithms select the relevant ones. Specifically, 22 variables will be used. Some of them are clearly related to wind energy production (like “100 metre U wind component”), others not so clearly (“Leaf area index, high vegetation”). Also, it is common practice to use the value of those variables, not just at the location of interest (Sotavento in this case), but at points in a grid around Sotavento. A 5x5 grid will be used in this case.



5x5 grid around Sotavento.

Therefore, each meteorological variable has been instantiated at 25 different locations (location 13 is actually Sotavento). That is why, for instance, attribute iews appears 25 times in the dataset (*iews.1, iews.2, …, iews.13, …, iews.25*). Therefore, the dataset contains 22*25 = 550 input attributes.

- ## WHAT TO DO:

- **(0) Setup: (0.5 points)**
  - Read the data (a Pandas dataframe) with:

    ```
    import pandas as pd
    data = pd.read_pickle('wind_pickle.pickle')
    ```

  - Historical data is available both from ECMWF (for the meteorological variables) and Sotavento (for energy production) from 2005 to 2010. The dataset has many columns. **Energy** is the response variable and must be kept. **Steps, month, day, hour should be removed, they cannot be used for training the models**. You can remove them already.
  - Now, you are going to modify artificially your dataset, so that there are some missing values in your data. In order to do that, first set the seed of the random number generator to your NA number:

    ```
    import numpy as np
    my_NIA = <PUT YOUR NIA NUMBER HERE>
    np.random.seed(my_NIA)
    ```

    and then, choose 10% of the columns randomly (excluding **year** and **energy**), and put 5% of missing values at random places in the chosen columns.

  - Save the modified dataset. This is the dataset that you are going to use from now on.

- We are going to use train/test for model evaluation (outer) and train/validation for hyper-parameter tuning (inner). Therefore, you should get three datasets, as follows. You can use column **year** for partitioning.
  - **i.** Train partition: data belonging to 2005 and 2006.
  - **ii.** Validation partition (inner): 2007 and 2008 data.
  - **iii.** Test partition (outer): 2009 and 2010.
- Column **year** should be removed now and data should be transformed to Numpy matrices, so that they can be used by scikit-learn.

- **(1) Model selection and hyper-parameter tuning: (1.5 points)**
  - Train and evaluate (outer) the following methods:
    - **i.** *KNN*, *Regression Trees*, and *SVMs* with default hyper-parameters
    - **ii.** *KNN*, *Regression Trees*, and *SVMs* with hyper-parameter tuning done with *RandomizedSearch*. The imputation method ('mean' or 'median') can be one of the hyper-parameters.
    - **iii.** Which one of the six approaches gives the best results (according to the test set)?
  - Notes:
    - i. Mean Absolute Error (MAE) will be used as metric.
    - ii. Please, remember that methods such as KNN and SVM need scaling and imputation (use pipelines for that). Trees do not.
    - iii. Also note that we are doing things a little differently than explained in class for creating the training, validation, and testing partitions. We are taking data in temporal order: we are not randomly shuffling our data. Therefore, there is no need to use train_test_split or KFold.
    - iv. For hyper-parameter tuning, you will need to use **PredefinedSplit** with no shuffling.

- **(2) Attribute selection: (1.5 points)**
  - Here, the only machine learning method that is going to be used is KNN. You will need to use imputation, but you can use the imputation method that worked best for KNN in the previous section.
  - You are going to try three pipelines:
    - i. One that does feature selection (*SelectKBest*) (and KNN)
    - ii. Another that does Principal Component Analysis PCA (and KNN)
    - iii. Another that uses both feature selection and PCA (and KNN)
  - Once the pipelines have been defined, do hyper-parameter tuning with them, using as hyper-parameters, the following:
    - Number of KNN neighbors
    - Number of features to be selected
    - Number of PCA components to be selected
  - Which of the three pipelines gives the best error? (according to the test set) Is there any improvement over results of the previous section?
  - Please, extract the names of the attributes selected by the first pipeline (feature selection). Does the method tend to select attributes which belong to the Sotavento location? What locations are more often preferred?

3. WHAT TO HAND IN:
All you need to hand in is your ipython notebook. But please, use some of the cells to make comments about what you are doing, your results, and the conclusions of your results. A good report, with tables and conclusions will be graded better.

If your group has two members, please write your names at the beginning of the notebook.

ATTRIBUTE NAMES

2 metre temperature

10 metre U wind component

10 metre V wind component

100 metre U wind component

100 metre V wind component

Convective available potential energy

Forecast logarithm of surface roughness for heat

Forecast surface roughness

Instantaneous eastward turbulent surface stress

Instantaneous northward turbulent surface

Leaf area index, high vegetation

Leaf area index, low vegetation

Neutral wind at 10 m u-component

Neutral wind at 10 m v-component

Soil temperature level 1

Soil temperature level 2

Soil temperature level 3

Soil temperature level 4

Surface pressure

Vertical integral of temperatura

Vertical integral of divergence of kinetic energy

Vertical integral of water vapour