Wind Turbine Prediction

Capstone Project

Dorado Vázquez, José M.
24/05/2020

1.- Definition

Project Overview

In Wind Turbines Scada Systems measure and save data's like wind speed, wind direction, generated power etc. for 10 minutes intervals. This file was taken from a wind turbine's scada system that is working and generating power in Turkey.

The data's in the file are:

- Date/Time: (for 10 minutes intervals).
- LV ActivePower (kW): The power generated by the turbine for that moment.
- Wind Speed (m/s): The wind speed at the hub height of the turbine (the wind speed that turbine use for electricity generation)
- TheoreticalPowerCurve (KWh): The theoretical power values that the turbine generates with that wind speed which is given by the turbine manufacturer.
- Wind Direction (°): The wind direction at the hub height of the turbine (wind turbines turn to this direction automaticly).

The dataset is located on Kaggle:

https://www.kaggle.com/berkerisen/wind-turbine-scada-dataset

For a more efficient prediction, both meteorological and location data as well as more temporal data are missing from the dataset.

Problem Statement

The objective of this kernel is the prediction of the energy produced by a wind turbine. For this estimation, we use the data provided in the dataset: wind speed and direction, generated power, and theoretical power that should produce (data provided by the manufacturer).

This is a supervised learning problem, since they provide us with the target variable. It should be noted that they do not provide us with some relevant data, such as: location, temperature, humidity, rain, air density, technical stops, breakdowns...

Both energy consumption and energy production forecasts are very important from an economic point of view for a company, since it is no longer possible to store them, they need to know approximately the amount they must buy or produce when demand exceeds production by renewable means.

Metrics

The metrics used in this project are r r2 mse:

$$\hat{R}^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2} = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

R^2 measures how far the data are from the model's predicted values compare to how far the data are from the mean.
MSE measures how far the data are from the model's predicted values.
The difference between how far the data are from the model's predicted values and how far the data are from the mean is the improvement in prediction from the regression model.

R^2

- proportional improvement in prediction of the **regression model**, compared to the **mean model** (model predicting all given samples as mean value).
- interpreted as the proportion of total variance that is explained by the model.
- **relative measure** of fit whereas MSEMSE is an **absolute measure** of fit
- often easier to interpret since it doesn't depend on the scale of the data.
  - It doesn't matter if the output values are very large or very small,
- always has values between -∞ and 1.
- There are situations in which a high R2R2 is not necessary or relevant.
- When the interest is in the **relationship between variables**, not in prediction, the R2R2 is less important.

MSE

- Sensitive to outliers
- Has the same units as the response variable.

- Lower values of MSEMSE indicate better fit.
- Actually, it’s hard to realize if our model is good or not by looking at the absolute values of MSEMSE or MSEMSE.
- We would probably want to measure how much our model is better than the constant baseline.

Disadvantage of MSE:

- If we make a single very bad prediction, taking the square will make the error even worse and
    - it may skew the metric towards overestimating the model’s badness.
- That is a particularly problematic behaviour if we have noisy data (data that for whatever reason is not entirely reliable)
- On the other hand, if all the errors are **smaller than 1**, than it affects in the opposite direction: we may underestimate the model’s badness.

Notes:

- For the R-Squared Fit, the closer the value is to 1, the better the performance of our model.

- RMSE / MAE is used to evaluate the variance in errors. Also, the value itself does not tell you much. You must compare different models to harvest from the RMSE / MAE.

- However, R-Squared does not need to be compared between different models. If the R-Squared / R-Squared adjusted is 0.10, we can recognize that the model is not doing a great job in modeling.

2.- Analisys

Data exploration

The data's in the file are:

- Date/Time (for 10 minutes intervals)
- LV ActivePower (kW): The power generated by the turbine for that moment
- Wind Speed (m/s): The wind speed at the hub height of the turbine (the wind speed that turbine use for electricity generation)
- Theoretical*Power*Curve (KWh): The theoretical power values that the turbine generates with that wind speed which is given by the turbine manufacturer
- Wind Direction (°): The wind direction at the hub height of the turbine (wind turbines turn to this direction automaticly)

Regardless of the features provided in the dataset, we can create others:

- suitable wind speed: Wind turbines start operating when the wind reaches a speed of 3 to 4 meters per second, and reaches maximum electricity production with a wind of about 13 to 14 meters per second. If the wind is very strong, for example 25 meters per second as average speed for 10 minutes, the wind turbines stop for safety reasons.
- loss: difference between the Teorical Powercurve and ActivePower.
- efficient use hours: hours of use with efficient wind.

We have only used these columns to illustrate the operation and behaviour of wind turbines.

We have not used them to make the predictions since they are highly correlated with those provided in the dataset: they are linear representations of the previous ones.

The data is included in a single 3.79 Mg file, which we will later divide into training and test sets.

This file, which contains 5 columns including the target variable, has 50530 rows.

The rows are temporary readings over a year, taken every day at 10 minute intervals.

So the first thing we did was to convert the date and time into the dataset index.

It should also be noted that there were no missing or outliers in the data.

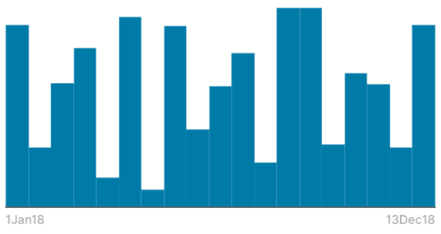
We also don't use featue selection, as is obvious.

For predictions we use the sklearn.StandardScaler along with the return algorithms.

Exploratory Visualization

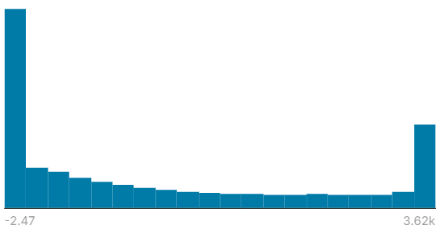
Below we show the bar charts of each column to see its distribution (data obtained from the kaggle dataset page):

Date/Time



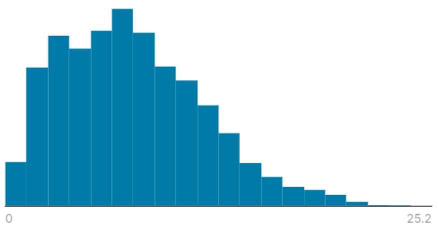
Valid 🟩	20.0k	40%
Mismatched 🟧	30.5k	60%
Missing 🟥	0	0%
Minimum	1Jan18	
Mean	23Jun18	
Maximum	13Dec18	

# LV ActivePower (kW)



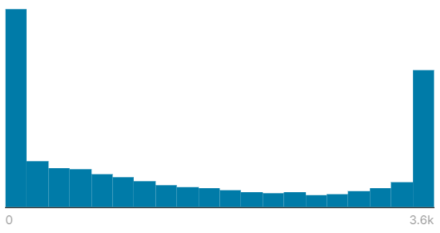
Valid 🟩	50.5k	100%
Mismatched 🟧	0	0%
Missing 🟥	0	0%
Mean	1.31k	
Std. Deviation	1.31k	
Quantiles	-2.47	Min
	50.7	25%
	826	50%
	2.48k	75%
	3.62k	Max

# Wind Speed (m/s)



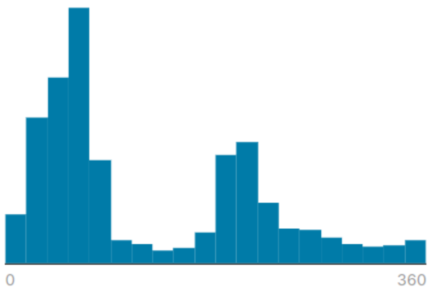
Valid 🟩	50.5k	100%
Mismatched 🟧	0	0%
Missing 🟥	0	0%
Mean	7.56	
Std. Deviation	4.23	
Quantiles	0	Min
	4.2	25%
	7.1	50%
	10.3	75%
	25.2	Max

# Theoretical_Power_Curve (KWh)



Valid 🟩	50.5k	100%
Mismatched 🟧	0	0%
Missing 🟥	0	0%
Mean	1.49k	
Std. Deviation	1.37k	
Quantiles	0	Min
	161	25%
	1.06k	50%
	2.97k	75%
	3.6k	Max

# Wind Direction (°)



Valid 🟩	50.5k	100%
Mismatched 🟧	0	0%
Missing 🟥	0	0%
Mean	124	
Std. Deviation	93.4	
Quantiles	0	Min
	49.3	25%
	73.7	50%
	202	75%
	360	Max

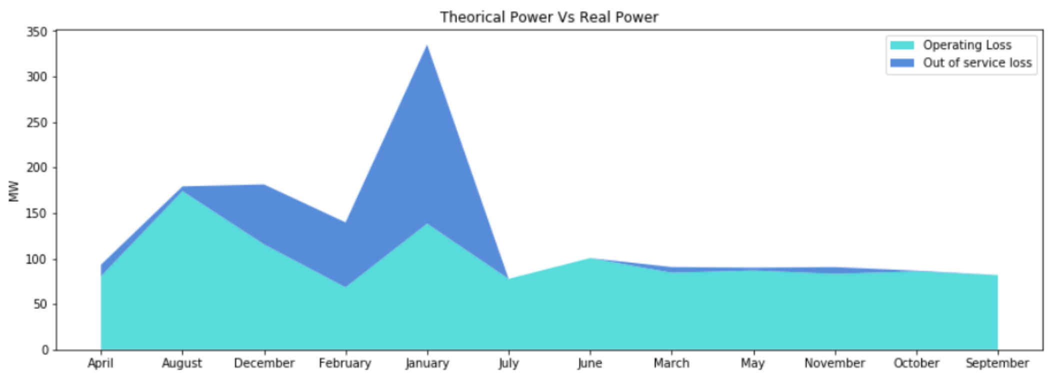
As we can see the LV Active Power and Theorical_Power_curve columns have a higher concentration of values at their extremes, especially at their initial point, which indicates that the power produced by the turbine was mostly below 25% or above 75%.

We also observe that the wind speed column is biased to the right, with most of its values grouped in the first three quantiles, so there are not many readings with very strong winds, which in this case have their maximum approximate in 25 m/s.

As shown in the analysis by month: the coldest months (autumn-winter, northern hemisphere) present a higher production of energy compared to the hottest months, with the exception of August.

We will now analyse the differences between actual and expected production:

- Hours of operations where the turbine did not operate but the speed was adequate.
- Fluid-dynamic and mechanical inefficiencies, maintenance, breakdowns...



We see how in the winter months, energy losses increase considerably. An aspect in favor of low temperatures is that the air is more dense, which translates into a greater potential for energy production that must be taken advantage of. In January and December, the fluid dynamics worsen with the accumulation of ice on the blades.

Algorithms and Techniques

To train this model there are some better algorithms than others, however since the amount of data is reduced we have used a lot of them to make the predictions, mainly based on trees and gradient decreasing:

- **LinearRegression:** fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.
- **ElasticNet:** Linear regression with combined L1 and L2 priors as regularizer.
- **Lasso:** Technically the Lasso model is optimizing the same objective function as the Elastic Net with l1_ratio=1.0
- **BayesianRidge:** Bayesian regression techniques can be used to include regularization parameters in the estimation procedure: the regularization parameter is not set in a hard sense but tuned to the data at hand.

- **LassoLarsIC:** regression algorithm for high-dimensional data, developed by Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani. LARS is similar to forward stepwise regression. At each step, it finds the feature most correlated with the target. When there are multiple features having equal correlation, instead of continuing along the same feature, it proceeds in a direction equiangular between the features.
- **RandomForestRegressor:** A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.
- **GradientBoostingRegressor:** Gradient Tree Boosting or Gradient Boosted Decision Trees (GBDT) is a generalization of boosting to arbitrary differentiable loss functions. GBDT is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems in a variety of areas including Web search ranking and ecology.
- **ExtraTreesRegressor:** randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule.
- **KernelRidge:** Kernel ridge regression (KRR) [M2012] combines Ridge regression and classification (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function in the original space.
- The form of the model learned by KernelRidge is identical to support vector regression (SVR). However, different loss functions are used: KRR uses squared error loss while support vector regression uses -insensitive loss, both combined with l2 regularization. In contrast to SVR, fitting KernelRidge can be done in closed-form and is typically faster for medium-sized datasets. On the other hand, the learned model is non-sparse and thus slower than SVR, which learns a sparse model for , at prediction-time.
- **DecisionTreeRegressor:** are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
- **KNeighborsRegressor:** Neighbors-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based on the mean of the labels of its nearest neighbors.
- **XGBRegressor:** is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) ... A wide range of applications: Can be used to solve regression, classification, ranking, and user-defined prediction problems
- **LGBMRegressor:** machine Learning is the fastest growing field in the world. Everyday there will be a launch of bunch of new algorithms, some of those fails and

some achieve the peak of success. Today, I am touching one of the most successful machine learning algorithm, Light GBM.

- **SVR:** The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function ignores samples whose prediction is close to their target.
- **Nural Networks** are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules.
- **arima:** 'Auto Regressive Integrated Moving Average' is actually a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values.
- **RNN:** the recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. ... Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable-length sequences of inputs.

3.- Methodology

Data Preprocessing

The pre-processing of data in this dataset did not require processes that are normally performed, since: they do not have missing data or atypical values.

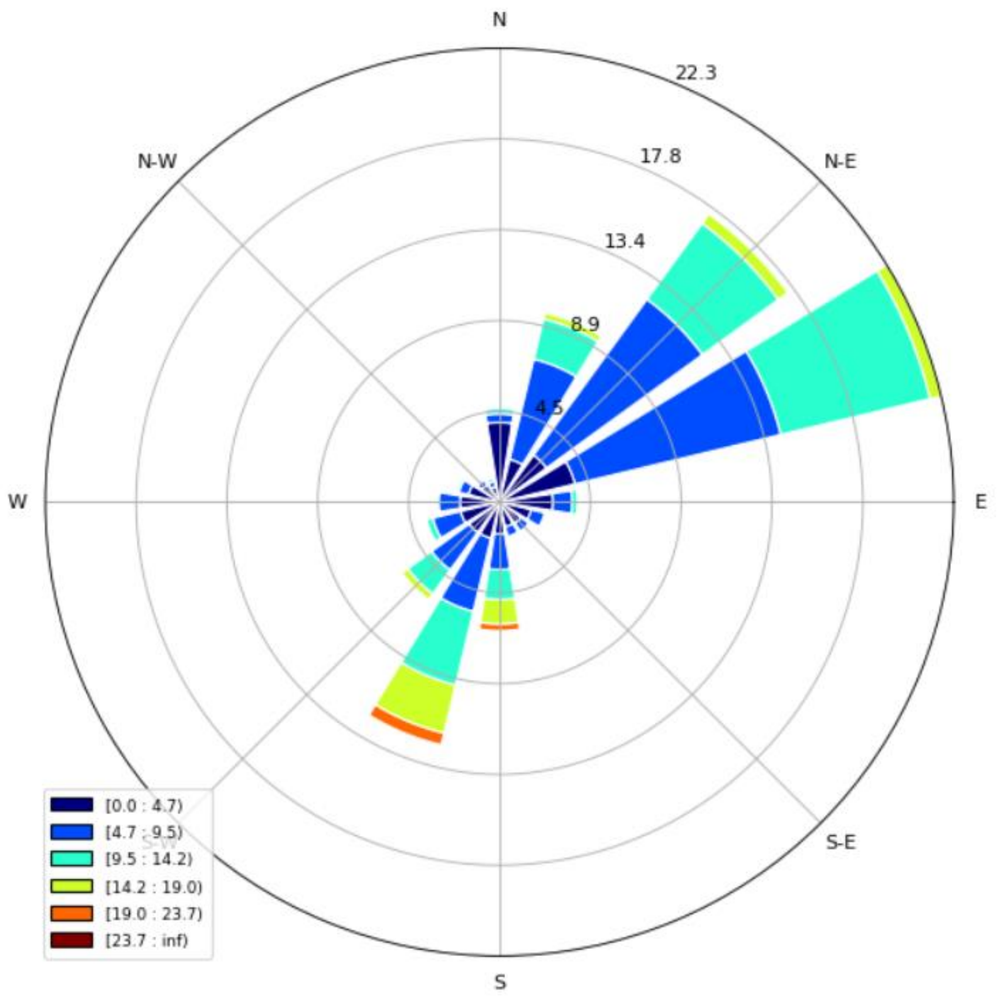
The DateTime column was converted to date and time, the year was eliminated, it only had one value and did not provide us with information. We later converted this column into the index.

The original numerical column values were float64, they were converted to float32 to increase the calculation speed.

```
In [5]: wt_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50530 entries, 0 to 50529
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   DateTime        50530 non-null  object
 1   LVA_Power       50530 non-null  float64
 2   WindSpeed       50530 non-null  float64
 3   TPC_KWh         50530 non-null  float64
 4   Wind_Direction  50530 non-null  float64
dtypes: float64(4), object(1)
memory usage: 1.9+ MB
```

The WindDirection column is expressed in degrees, although we created another one by extrapolating the data to cardinal points to relate it more easily to wind speed.



Implementation

The algorithms used in the predictions were executed through a pipeline with several metrics to be able to contrast the results obtained.

In this way, it is easier to evaluate the models with better results to later refine their hyperparameters and achieve better results.

Regardless of these early algorithms, we also used Arima, although the predictions did not improve.

Similarly, we used a neural network with the same results.

This is because the data provided are scarce in terms of the number of characteristics.

Refinement

To improve training results we decided to use the hyperopt framework, which creates a grid of hyperparameters and selected values among which it finds the best results.

```
In [57]: from functools import partial
         from hyperopt import fmin, hp, tpe, Trials, space_eval
```

```
In [58]: # Define searched space
         hyper_space = {'n_estimators': 1000 + hp.randint('n_estimators', 1000),
                        'max_depth':  hp.choice('max_depth', [3, 5, 8, 12]),
                        'num_leaves': hp.choice('num_leaves', [7, 25, 50, 800]),
                        'subsample': hp.uniform('subsample', 0.6, 1.0)}
```

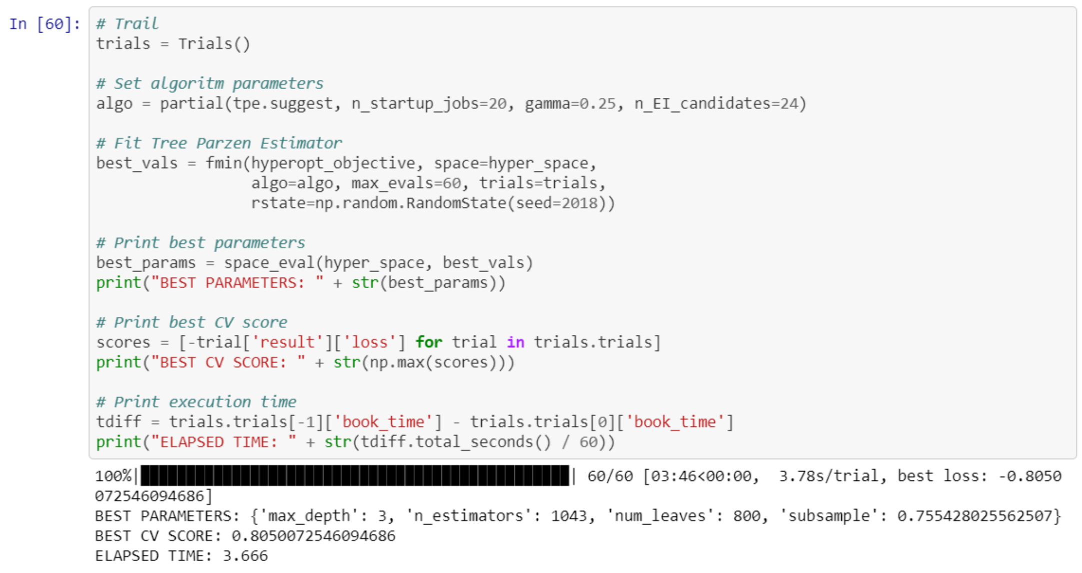
```
In [59]: def evaluate(params, X, y):

             # Initilize instance of estimator
             est = LGBMRegressor(boosting_type='gbdt', n_jobs=-1, random_state=2018)

             # Set params
             est.set_params(**params)

             # Calc CV score
             scores = cross_val_score(estimator=est, X=X, y=y, scoring='r2', cv=4)
             score = np.mean(scores)

             return score

         # Objective minizmied
         hyperopt_objective = lambda params: (-1.0) * evaluate(params, X_train, y_train)
```

On this particular occasion, when using LGBMRegressor, we decided to use the number of estimators, max_depth, num_leaves and subsample.

Then, using the cross validatos technique and with cv=4 so that the calculations do not take too long, we were given the optimal values among those provided.

4.- Results

A final model was generated with tuned parameters. The submission of predictions from final model scored 0.9405.



```
In [60]: # Trail
         trials = Trials()

         # Set algoritm parameters
         algo = partial(tpe.suggest, n_startup_jobs=20, gamma=0.25, n_EI_candidates=24)

         # Fit Tree Parzen Estimator
         best_vals = fmin(hyperopt_objective, space=hyper_space,
                          algo=algo, max_evals=60, trials=trials,
                          rstate=np.random.RandomState(seed=2018))

         # Print best parameters
         best_params = space_eval(hyper_space, best_vals)
         print("BEST PARAMETERS: " + str(best_params))

         # Print best CV score
         scores = [-trial['result']['loss'] for trial in trials.trials]
         print("BEST CV SCORE: " + str(np.max(scores)))

         # Print execution time
         tdiff = trials.trials[-1]['book_time'] - trials.trials[0]['book_time']
         print("ELAPSED TIME: " + str(tdiff.total_seconds() / 60))
```

```
100%|████████████████████████████████████████████████| 60/60 [03:46<00:00,  3.78s/trial, best loss: -0.8050
072546094686]
BEST PARAMETERS: {'max_depth': 3, 'n_estimators': 1043, 'num_leaves': 800, 'subsample': 0.755428025562507}
BEST CV SCORE: 0.8050072546094686
ELAPSED TIME: 3.666
```

R2-score in test set:

- ScaledLR : 0.85492
- ScaledLASSO: 0.85501
- ScaledEN: 0.80315
- ScaledGBMR: 0.78462
- ScaledCART: 0.68208
- ScaledGBO: 0.81525
- ScaledXGBR: 0.79058
- ScaledKNN: 0.78632
- SaledSVM: 0.84633
- LGBMRegressor: 0.9405
- Deep Neural Network: 0.9119
- Arima (Predict 3-hours-ahead): 0.912

The best results are provided using LGMBregressor and Neural Networks.

Justification

This model is robust enough to solve the problem we face, although, given the small amount of data, it will only serve as a basis for larger projects.

5.- Conclussion

This project provides reliable predictions with the data provided, although it well serves as a basis for future projects with more extensive data, given the large number of algorithms used for its training, such as the refinement of hyperparameters.

The results prove that the choice of a deeply model improves the behaviour of the model. The best performance has been done using neural networks.

The future works directions could be:

- Test another architecture like as LSTM or CNN to extract the features of the input signal using sequences.
- Use wavelet transform or variational mode decomposition to filter the input signal into high and low frequencies. Each frequency will be a new input of an independence model.
- Random search of the hyperparameters of the proposed ANN to improve the performance.
- Try the performance use other location to have a robost measure of the error.