ELEN062-1 Introduction to machine learning

Faculty of Applied Science

Wind predicting model

Teachers :
Pierre GEURTS
Louis WEHENKEL

Assistant :
Yann CLAES

Students :
Romain LAMBERMONT, s190931
Arthur LOUIS, s191230

December 16, 2022

Contents

List of Figures

List of Tables

1 Introduction

For this project, we received the data collected on 10 wind farms in Australia with missing power values respecting some given weeks in every wind farm. The task we were asked was to compute an approximation of the real power of the farm given the wind speed projected on the two main axis at heights of 10 and 100 meters.

In this brief report, we'll explore the different techniques used to find the best way to approximate the real wind data from our files by using regression algorithm from the `scikit-learn` package and neural networks from the `pytorch` package.

We started by approximating the data with some linear regression algorithms before trying some decision trees, k-nearest-neighbors, random forests and finally neural networks.

We'll talk about handling the given data, our generic method for our regression algorithms, our method to tune the hyperparameters, the obtained scores for the different tuned algorithms. We'll then go into more details concerning the neural network we used in the last part of the project and also discuss some possible improvements.

2 Data handling

By taking a first look at the data and the statement, we quickly found the missing power values represented by a -1 in the `TARGETVAR` column of all the `Y_i.csv` files. We used this specific -1 value to identify and split our training/test sets. We then wanted to check our files for outliers. We used boxplots to begin investigate the outliers topic :

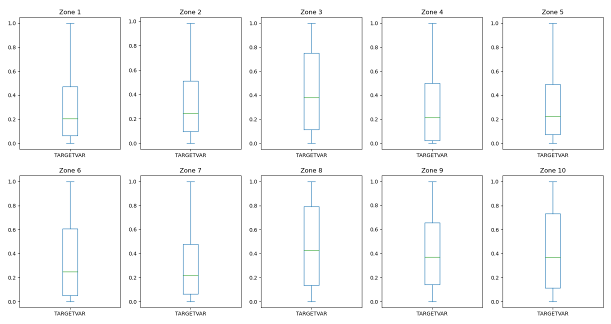

Figure 1: Boxplot of TARGETVAR values in Y_i zones

On one hand, as the data in the `Y_i.csv` zones was normalized, it is as expected that we didn't find outliers in it. On the other hand, by diving into the boxplots for the variables in the `X_i.csv` files, we clearly detect some outliers (using the `boxplot.py` file with the basic data) :

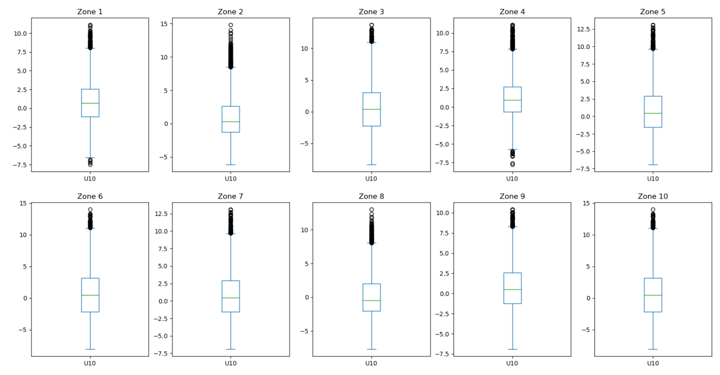

Figure 2: Boxplot of U10 values in X_i zones

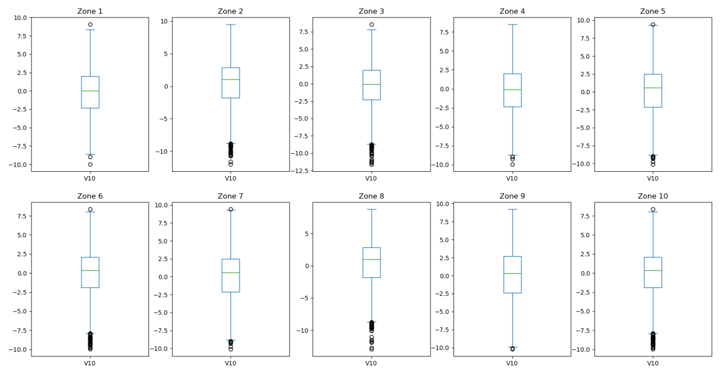

Figure 3: Boxplot of V10 values in X_i zones

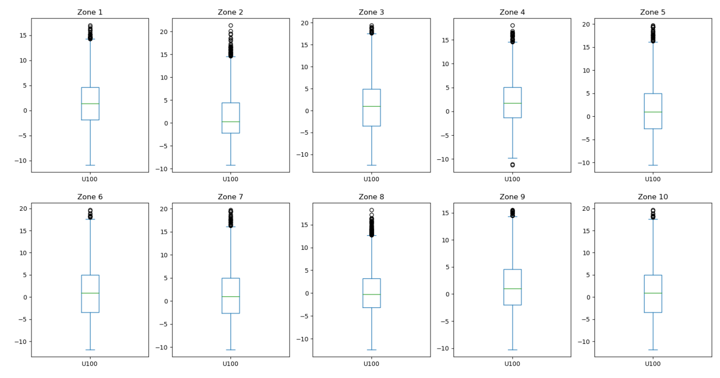

Figure 4: Boxplot of U100 values in X_i zones

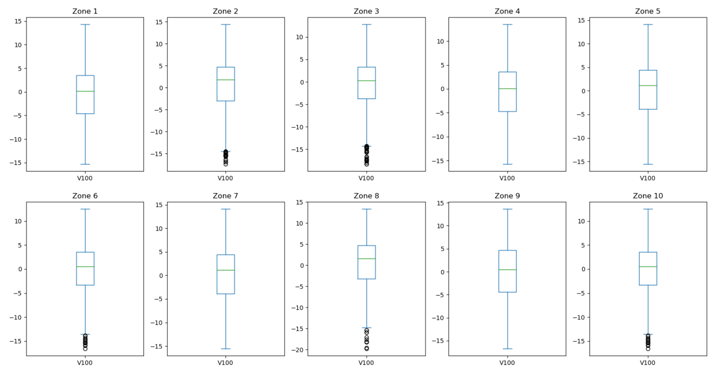

Figure 5: Boxplot of V100 values in X_i zones

We then needed to handle these outliers before fitting our models. This is done in the `data.py` file using the `handle_outliers` function by fixing the value of each outlier to the nearest non-outlier value in the dataframe. Indeed, in regression algorithm, handling the data is the first big step as outliers may induce the algorithm to perform a lot more differently. Here's the resulting boxplot for the `U10` variable :

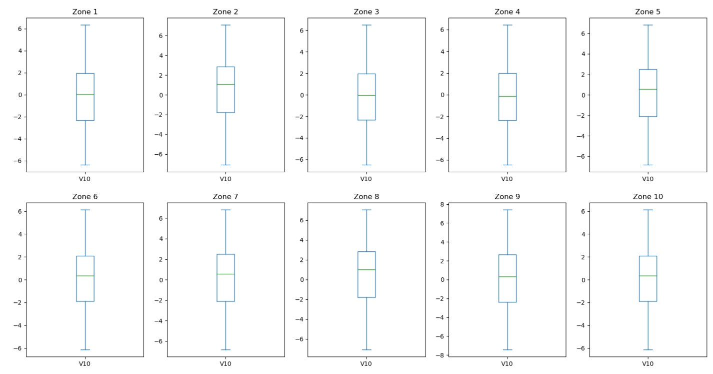

Figure 6: Boxplot of U10 values in X_i zones after outlier handling

3 General method for Regression Algorithms

For all the methods of regression algorithms, we have the same basic method in 5 steps :

1. Process the data (explained below)
2. Create a model by zone
3. Tune hyper-parameters (explained below)
4. Fit the data
5. Predict missing values

Data processing

For each zone, the data is gathered from the `.csv` files which have previsously been handled for outliers. This data is then splited between the training and testing set by locating the `-1` values in the `TARGETVAR` columns from the `Y_i.csv`.

Hyperparameters tuning

Hyperparameter tuning is the process of choosing the optimal values for the hyperparameters of a machine learning model. Hyperparameters are parameters that are not learned from the data but are set by the practitioner. They control the behavior of the learning algorithm and have a significant impact on the performance of the model.

There are several methods for tuning hyperparameters, including manual tuning, grid search, and random search.

Manual tuning involves manually trying out different values for the hyperparameters and evaluating the performance of the model for each combination.

Grid search is a more automated approach that involves specifying a grid of hyperparameter values to search over. The model is trained and evaluated for each combination of hyperparameters, and the combination that gives the best performance is selected.

Random search is another automated approach that involves sampling random combinations of hyperparameters and evaluating the model for each combination.

Hyperparameter tuning is an important step in the machine learning process, as it can significantly improve the performance of the model. It is generally recommended to tune the hyperparameters of a model after it has been trained on the data, as the optimal values of the hyperparameters may depend on the specific characteristics of the training data.

In this project we used grid search for every regression with the function `GridSearchCV` using 5-fold cross-validation on our training set. However grid search can be computationally expensive, as it requires training and evaluating the model for every combination in the grid. This can become infeasible for large grids or complex models. This explain why one use manual tuning for the random forest regression to reduce the computation time.

The values gathered by the grid search are reported in this table (using `hyperparams.py` file to compute paramaters for each zone) :

Zone	Ridge (alpha)	Decision Tree (m_depth , m_s_split, min_s_l)	kNN (n_nbrs, weight, algo)
1	10	10 10 2	32 distance auto
2	10	10 2 2	32 distance brute
3	10	10 2 2	32 uniform auto
4	10	10 10 2	32 distance brute
5	10	10 2 2	16 uniform auto
6	10	10 2 1	32 uniform auto
7	10	10 5 2	16 uniform auto
8	10	10 10 2	32 distance auto
9	10	10 5 2	16 uniform auto
10	10	10 10 2	32 uniform auto

Table 1: Best hyperparameters relative to each zone

For the random forest regression algorithms, we fixed the values of the different hyperparamters (`n_estimators`, `max_depth`, `min_samples_split`, `min_samples_leaf`) respectively at 1000, 1000, 10 and 4.

4 Regression methods

For this project, one used multiple regression method to predict the data. Here are a list of all method used and a brief explanation (fitting and prediction for all our methods in `submission.py` file) :

- **Linear regression:** Linear regression is a simple and widely-used method for modeling the relationship between a dependent variable and one or more independent variables. It assumes that the relationship between the variables is linear, and estimates the coefficients of the model using a least squares approach.

- **Random forest regression:** Random forest is an ensemble method that combines the predictions of multiple decision trees. It creates a set of decision trees using bootstrapped samples of the training data, and each tree makes a prediction for the target variable. The final prediction is the average of the predictions made by the individual trees.

- **Decision tree regressor:** Decision tree is a type of model that makes predictions based on a set of rules learned from the training data. It creates a tree-like structure in which each internal node represents a decision based on one of the input features, and each leaf node represents a predicted value for the target variable.

- **Ridge:** Ridge is a linear regression model that adds a regularization term to the objective function. This term penalizes large coefficients and helps to prevent overfitting. Ridge regression is often used when there are a large number of features and the risk of overfitting is high.

- **K-nearest neighbors regression:** K-nearest neighbors (KNN) is a non-parametric method that makes predictions based on the average of the target values of the K-nearest neighbors in the training dataset. It is sensitive to the scale of the features and may require preprocessing to achieve good performance.

For each method, we computed the `MSE` and `MAE` for all zones by comparing the values computed by regression against the values available in the file `ytrueall` on the assistant's website since the day after the end of the challenge. The method's scores are reported in the table below (using `score.py` file) :

Score		
Decision Tree	MSE	0.075401
	MAE	0.197220
k-NN	MSE	0.071602
	MAE	0.201271
Linear Regression	MSE	0.098086
	MAE	0.263663
Random Forest	MSE	0.066831
	MAE	0.186580
Ridge	MSE	0.098082
	MAE	0.263664

Table 2: Results for regression algorithms

In our case, looking at mean of the MSE and MAE metrics on each zone, one can say that Random Forest is the best regression algorithm for our data.

One of the reasons for the strong performance of random forests is that they are based on the idea of building an ensemble of decision trees, where each tree makes a prediction and the final prediction is the average or majority vote of all the trees. This approach helps to reduce overfitting and improve the generalization of the model to unseen data.

By looking at the values of our errors for the decision trees, this makes the interpretation above more logical as the Decision Tree is the second best algorithm used to implement regression in our project. This can be explained by a kind of feature selection principle. Indeed, during the training part of the regression, the decision tree automatically selects the most important features for making predictions.

In our case, the k-nearest-neighbors regression, also gives us some good results. Indeed, as previously seen in the first project, this algorithm, when the hyperparameters are wisely chosen produces some of the best results possible.

Lastly, as expected, the two linear regression algorithms produced the worst results. Indeed, linear regression and ridge regression comes are the same kind. In our case, they produced the same results which means that even with the regularization term, it doesn't overfit our data. We simply can keep the linear regression in our case.

Overall, it can be said that the results obtained using random forests tend to be highly reliable and consistently outperform those of other machine learning algorithms in our case.

5 Neural Network

For this project, we tried an other type of model: The neural network. For our machine learning project, we decided to use a neural network with 4 input features, a hidden layer with 10 nodes, and 1 output. The 4 inputs used in our implementation are `U10,V10,U100,V100` for each zone independently (implementation in `neural_network.py` file).

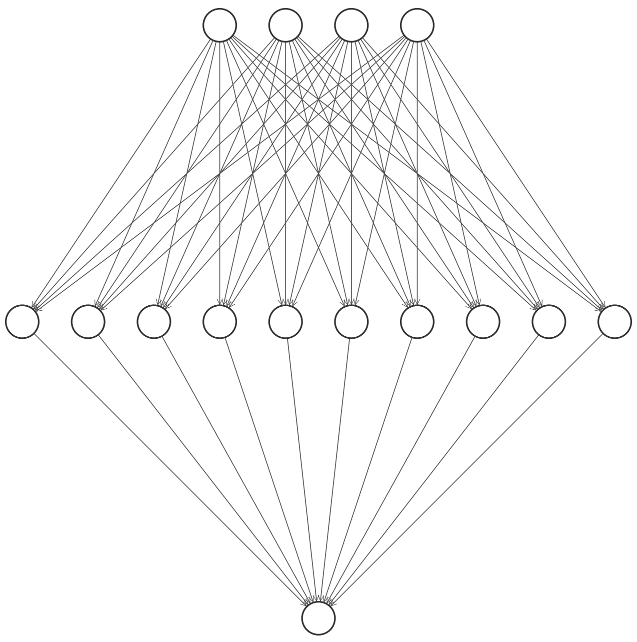

Figure 7: Neural Network schematic

We choose linear layer for our project which is a type of layer that performs a linear operation on its input. A linear operation is an arithmetic operation that preserves the sum and difference of the inputs, meaning that the output of the operation is simply a scaled and shifted version of the input.

For example, suppose we have a linear layer that takes as input a vector x of size n, and applies a linear transformation to it using a weight matrix W of size n x m and a bias vector b of size m. The output of the linear layer is then given by:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Here, W is the weight matrix, which determines how the input is scaled and transformed. The bias vector b determines how the input is shifted. The output y is a transformed version of the input x, and has size m.

The neural network implemented learns using 1000 `epochs` and by passing our loss function backward into the network to update the network's weights. The loss function we have chosen for our implementation is the mean-square error (`MSE`).

At first, we also implemented another neural network taking 8 inputs in input (the first 4 and 4 additional input : hour, day, month and year) and 20 hidden layers (implementation in `neural_network2.py` file). Here are the results obtained by the neural networks :

Neural Network	MSE	MAE
`nn1`	0.066251	0.185222
`nn2`	0.100500	0.270464

Table 3: Results for neural networks

6 Stacking

Stacking is a machine learning technique that involves training a second-level model to make predictions based on the outputs of several base models. It is a form of ensemble learning, which combines the predictions of multiple models to create a more accurate and stable prediction.

In stacking, the base models are trained on the same dataset and then make predictions. These predictions are then combined and used as input to the second-level model, which is trained to make a final prediction based on the output of the base models. This final prediction is typically more accurate than the predictions of the individual base models.

Stacking can be used with a variety of base models, such as decision trees, neural networks, or support vector machines. It is a powerful technique that can improve the performance of machine learning models, especially when the base models have complementary strengths and weaknesses.

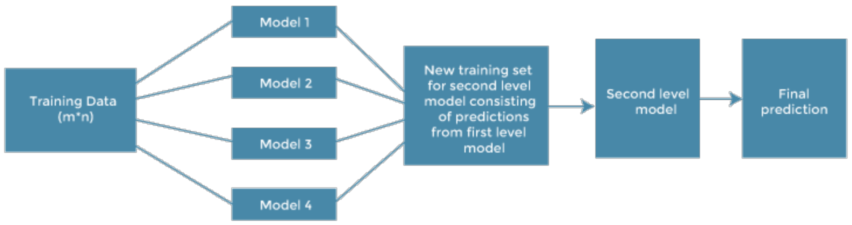
Training Data (m*n)
Model 1
Model 2
Model 3
Model 4
New training set for second level model consisting of predictions from first level model
Second level model
Final prediction

Figure 8: Stacking schematic

In our case, we implemented a stacking regressor in the `stacking.py` file. Our ensemble regressor gets his base approximations from all the approximations discussed here above (except random forests and neural networks). The second level model is based on a decision tree. All first level models are trained with the tuned hyperparameters for each zone. Here are the results we obtained :

	MSE	MAE
Stacking	0.070009	0.194127

Table 4: Results for stacking regressor

As can be seen in the table here above, the results we obtained using stacking were an improvement on the individual regression algorithms but were still unable to beat the random forests and neural networks.

7 Conclusion and possible improvements

In conclusion, after exploring all kinds of different techniques to fit the wind power data correctly, we find that the one that produces the best results is the neural network. Indeed, it gave us the best scores while being a lot faster than stacking and random forests.

We could improve our implementations and methods by fine-tuning the hyperparameters even more, preprocess the data in a better way.
In the case of neural networks, finding the good selection of inputs to produce the best results could be a better approach (more inputs, outputs from previous computations of the power, ...).
In the case of stacking, we could choose our regressors to cancel out the advantages and weaknesses of each regressor to produce a cleaner prediction.