LIÈGE université
Sciences Appliquées

Université de Liège
Faculté des Sciences Appliquées

Project 3: Competition

ELEN0062-1 : Introduction to machine learning

Andrei Galan (s191903)
Tom Moës(s191408)
Jérôme Pierre (s190979)

December 16, 2022

Introduction

In this project, we are asked to predict the power generated by 10 different wind turbine farms at given points in time. More precisely, we are given 10 hourly sampled datasets containing the components of the speed of the wind in N-S direction and in E-W direction at an altitude of 10 meters and 100 meters through time.

In order to do this regression task, different methods will be taken into account after performing an analysis of the data.

Preliminary data analysis

Before doing anything, it is a great idea to have a look at our data or see if we can't see underlying structures.

New features

Let's start with some physical sense. We are trying to find the values of wind power P for 10 wind farms. However, it is known that a simplified expression of the power produced by wind farms is proportional to the cube of the speed of the wind.
The power produced by a wind turbine is basically the derivative of the kinetic energy of the wind that sets in motion the blades of the turbine with respect to time.

$$P = \frac{dE}{dt}$$

One knows that the kinetic energy is given, for all cases, by:

$$E_k = \frac{1}{2} \cdot m \cdot v^2 \tag{1}$$

where v is the velocity, m the mass and E in $[J]$, or $[\frac{kg \cdot m^2}{s^2}]$. The mass set in motion by the wind is :

$$m = \rho \cdot A \cdot x$$

where ρ is the density of the air $[\frac{kg}{m^3}]$, A the cross section area of the wind in $[m^2]$ and x the distance through which the wind travels.
By deriving 1 with respect to time, one obtains:

$$P = \frac{dE}{dt} = \frac{d}{dt}(\frac{1}{2} \cdot \rho \cdot A \cdot x \cdot v^2) = \frac{1}{2} \cdot \rho \cdot A \cdot v^2 \cdot \frac{dx}{dt}$$

One knows that the derivative of the position with respect to time is equal to the velocity v. The final result is given by the following formula:

$$P = \frac{1}{2} \cdot \rho \cdot A \cdot v^3 \tag{2}$$

where P is expressed in [W], or $[\frac{kg \cdot m^2}{s^3}]$, ρ the mass density of the wind $[\frac{kg}{m^3}]$, A the cross section area, $[m^2]$, and v the velocity of the wind, $[\frac{m}{s}]$.

However, even though it gives a great first idea of the kind of relations one can expect, it is never that simple in real life. Indeed, all sorts of efficiency coefficients, such

as the gearbox or generator efficiency, were not taken into account. Other phenomena which might interfere with the output power such as shading phenomena weren't taken into account either.

Yet, physically speaking, it makes sense to compute the norm of the vector (the previous formula also supports that it is a good idea). Since we introduce the norm of a vector, it might be a great idea to also add the corresponding angle. That is why we'll add those features to our dataset.

Let's now really have a look at our data with a pair plot of the 2000 first instances of the first wind farm.

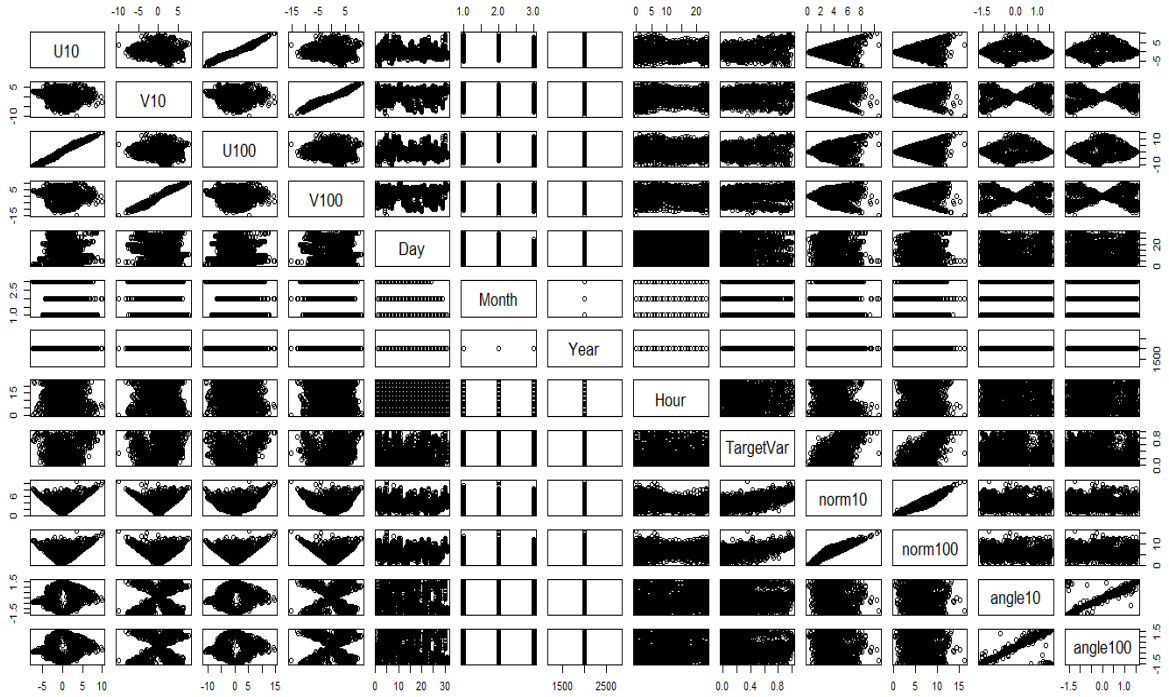


Figure 1: Pairs plot of the dataset

There seems to be no clear outlier in our dataset which is great. If we wanted to be sure of that, we could have used robust estimations with some ddplot. Another nice property of our dataset is that it contains no NaNs at all.

The first thing that can be observed in Figure 1 is that the different values, norm and angle at heights 10 m and 100m, are well correlated. That would mean that if we were in a dimension-reduction context, we could decrease the dimension of our dataset by using a simple linear correlation.

Another interesting thing is that the target variable and the norm of the speed are pretty highly correlated especially with the one at 100m. To be exact, the correlation between the targeted variable and the norm 100 was 0.733 and the one associated with the norm at 10 meter high was 0.70.

Those results can be seen in the correlation matrix that can be seen in Figure 2

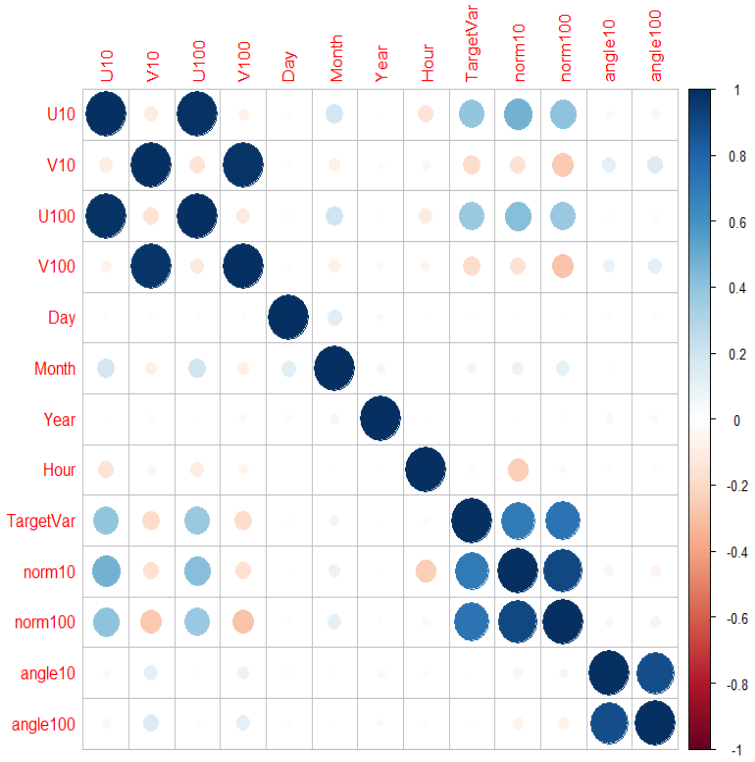


Figure 2: Correlation matrix

We obviously also tried to add the cube of the norm of the speed at both heights since theory says it is a great approximation. However, we got a worse correlation, probably due to the reasons we talked about earlier.

Something interesting that was found is that it might be interesting to combine other sites' data. Indeed, it was quite surprising to realize that the targeted value was sometimes better correlated with the norm of the speed in other sites than in the one it was obtained from. For example, it can be seen in Figure 3 that the norm of the speed of the wind at 100 m from zone 4 and 6 seem to be better predictors of the targeted variable from zone 0 than the one from the same zone.

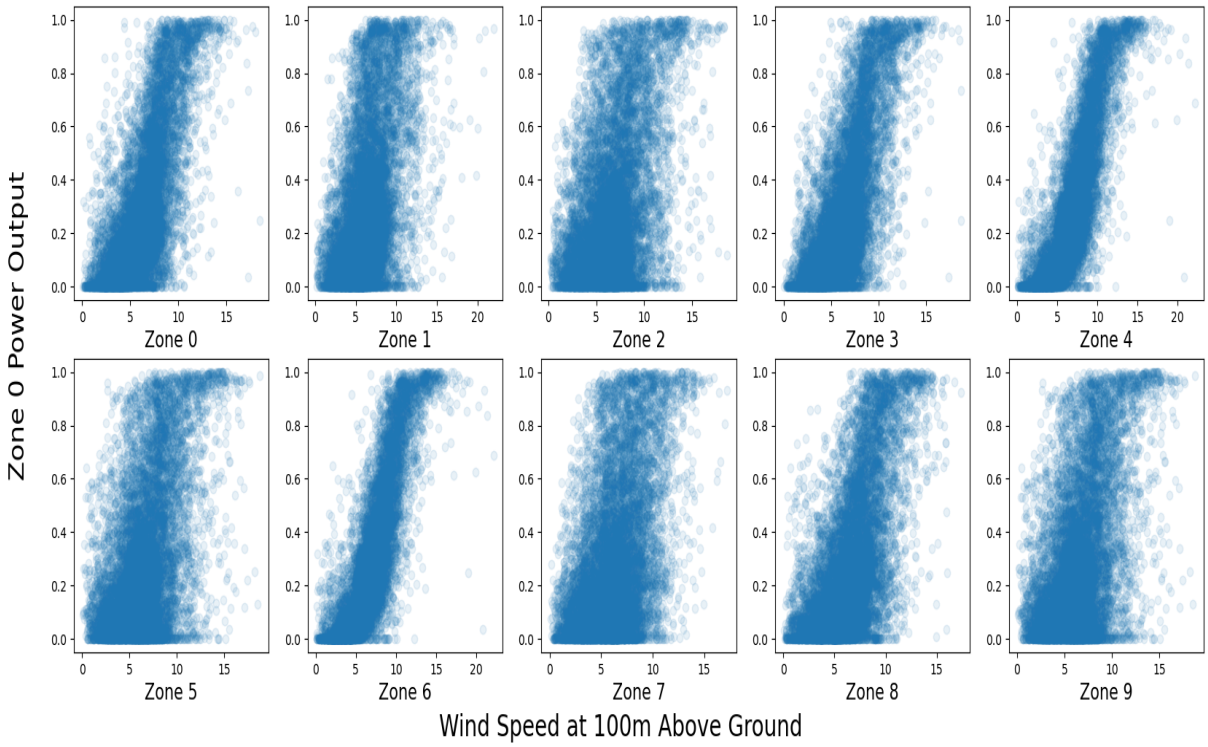


Figure 3: Comparison of the norm of the speed

We know that all the wind farms are located in Australia which may explain this correlation. However, since Australia is quite a large country, it seemed quite unlikely at first that the different sites could be so linked together.

That means that a possibility to increase the performance of our models is to add the different norms of the speed from other zones as new columns in our dataset.

Other similar techniques could have been used in order to get better results. An example of that is to consider the deterministic nature of the macroscopic world and then add features from previous time steps as columns. The same can also be done with future time steps. The issue with this method is that it induces NaN values for the few first or last columns of our dataset. We decided to take care of that by replacing NaN with the value of the current featured associated.

The final idea of this kind was to add measures of acceleration to our dataset. Obviously, one cannot directly approximate derivatives at a given timestep because our data has been sampled hourly. The wind's speed can vary quite a lot in such a time. That's why we will only have an approximation.

In practice, we tried to get results by introducing those values:

$$\mathrm{acc}_{\mathrm{prev}}(\mathrm{i},\mathrm{t}) = [\mathrm{norm}_{100}(\mathrm{t}) - \mathrm{norm}_{100}(\mathrm{t}-\mathrm{i})]/\mathrm{i}$$
$$\mathrm{acc}_{\mathrm{next}}(\mathrm{i},\mathrm{t}) = [\mathrm{norm}_{100}(\mathrm{t}) - \mathrm{norm}_{100}(\mathrm{t}+\mathrm{i})]/\mathrm{i}$$

Since adding those values will also make NaNs appear, we decided to force the first or the last instances of those values to copy the first one that was different from NaN.

Main approach

In order to have the best results possible, we followed the following strategy.

Firstly, we choose the different features that we would keep in our dataset. Then, we have to try different machine learning models for each zone, select the best parameters and then have an estimation of the model to compare with others. This method requires us to start by defining which proportions are given to our learning set, validation set and test set. We chose to use 80% for the learning set and 10% for both validation set and test set. Since we had quite a large dataset for each region (more than 10K instances), we thought 10% was ok for the two last sets.

Finding the best parameters was made by using K-fold cross-validation. That means that we decomposed our learning set into smaller sets of given size. In our case, we considered a size of 1500 instances due to the large size of the dataset. For every possible value of those parameters, we train the model on the different subsets and try to predict the validation set. It allows getting the estimation of the expected error, bias and variance through the same process that was discussed in project 2. The corresponding curve is plotted for each different estimator. This method is useful since it allows us to easily compute the MAE[1] on top of it and also allows us to get a measure of the variability of the error. This can be seen in different graphs (such as Figure 4a or Figure 4b). In those graphs, the darker red or blue zone corresponds to the standard deviation around the mean values of the MSE[2] or MAE. The lighter region describes the set of values between the maximum and the minimum error found. Finally, once we select the parameters that minimize the MSE and/or the MAE, we can get an estimation of the error made by the model with this optimal set of parameters by training our model on the learning set and then trying to predict the test set. Indeed, we can't just consider the value of the error given when trying to fix the parameters because it is a way too optimistic estimation. Even here, the proportion of the test might be too small to get a great estimation. Fortunately, we had the submission platform that could also be used as a test set.

A legit remark here is to point out that we should put more thought into doing that because if the subset of the learning set is not carefully selected, we might destroy causality. Indeed, a common approach in such a context consists in decomposing the learning set in k subsets and making sure that causality is preserved in each of them, and then aggregating the subsets corresponding to the first timesteps in order to try to predict the next subset. In our case, we decided not to take into account the date, the month, the year, or the hour in our prediction. The idea is that since there is such a great causality between the different features different from time and the targeted variable, time causality can be overlooked. That is why, we dropped the year, the month, the day and the hour columns from our dataset.

Since we found quite a lot of different possible interesting features to add to our dataset, we didn't try every possibility. It would have been quite time-consuming for not such an important increase otherwise. We tried something way more similar to a greedy approach. We started by limiting the number of possible accelerations or previous (resp. following) features that we were interested in. Then, we tried the different possibilities while fixing the current best. For example, if we had found it interesting to add the norms

[1] mean absolute error

[2] mean square error

of the other sites in a previous test, we would have kept it to see if it is interesting to add 4 accelerations. If we had then found that adding accelerations was useless, we wouldn't have tried them anymore afterward. Such a method sure is quite "fast" to perform but cannot make that we don't miss an interesting decrease in error. Indeed, it is possible that adding a feature will only turn useful if another one is also added. This method is likely to fail to find such combinations.

Let's now review the different machine learning algorithms we have tried.

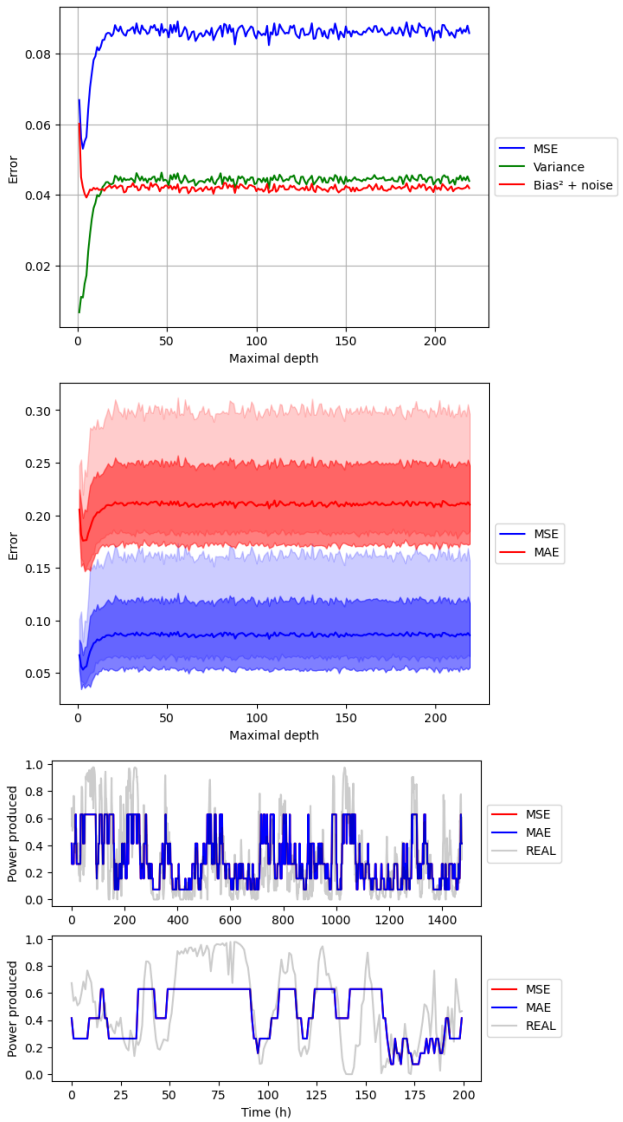
Decision trees

The first idea was to use a decision tree regressor. This method consists in building a tree by splitting our data into subsets by observing the input features in order to make predictions on the output feature. Each split is made in such a way that a measure of error is minimized.

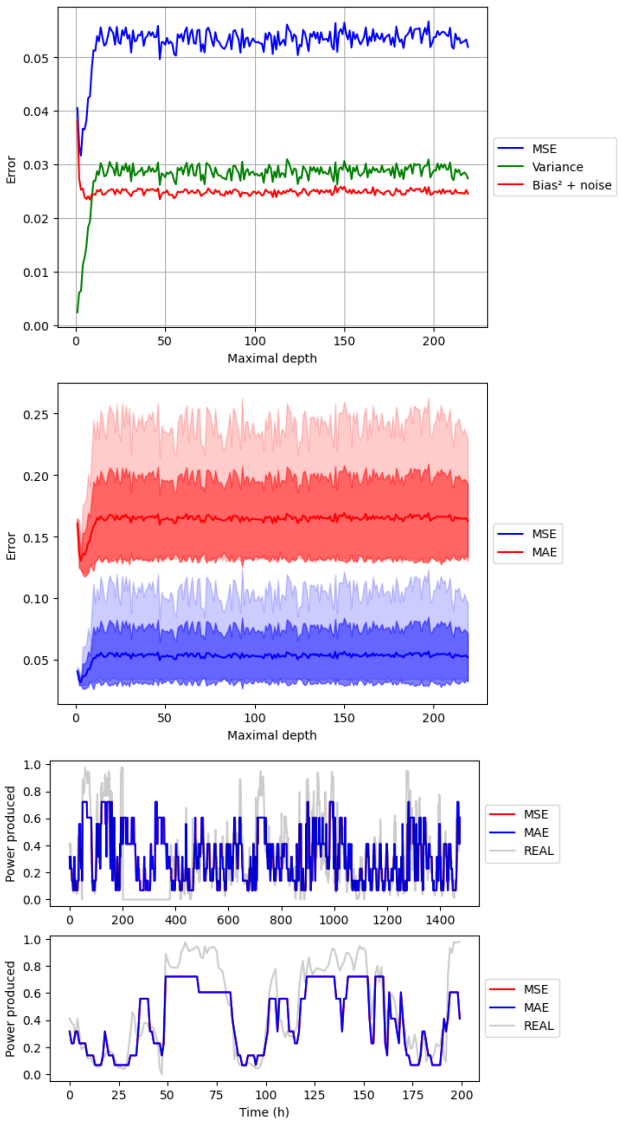
The maximal depth determines the maximum number of splits a tree can have. keeping it low will allow preventing overfitting. The procedure described above is used in order to fix it. Minimizing the MSE in the k-fold cross-validation gave us the following parameters for each zone:

$$\text{Maximal depths} : [3, 3, 4, 3, 3, 4, 2, 2, 4, 3]$$

The following graphs were made using them. (here with the first, fourth, and seventh zones), depicting the error with respect to the maximal depth, and the estimate of the given data with respect to time.



(a) 1^{st} zone



(b) 2^{nd} zone

Let's first observe that the trends in the bias and variance curve are the theoretical ones. Indeed, increasing the maximal depth will result in an increase in the complexity of the model. The decision tree then tries to approximate as well as possible the data from the learning sample by building deep trees which results in an increase in variance and a decrease in bias (possible overfitting). If the maximal depth is too small, the model gets too simple and then results in an increase in bias and a decrease in variance (possible underfitting).

What's more, we can observe that the bias is always at least as important as the variance which induces it might be a bit annoying for making great predictions.

From the figures above, it can also be observed that the shape of the graphs remains the same whatever the zone. The only difference is in the variance of the errors. Indeed, we observe a larger variance for the errors in zone 1 (Figure 4a) than in zone 2 (Figure 4b).

Finally, it can be seen in the plots of the predicted variable through time that the predictions gives curves that seem a bit too simple for this scenario. This is why we considered this method as suboptimal.

Random forest

We can use random forest to try to get better results. The random forest algorithm is an ensemble method used for regression trees. It randomly creates a tree based on the learning set such that the estimator is an average of the predictions. It gives better results than the simple regression tree which is why it might be interesting.

There are different parameters that can be tuned in the implementation of random forest in scikit learn. The only one we deemed as interesting was the maximal depth of the trees[3]. Applying our methodology in it yields the following graphs:

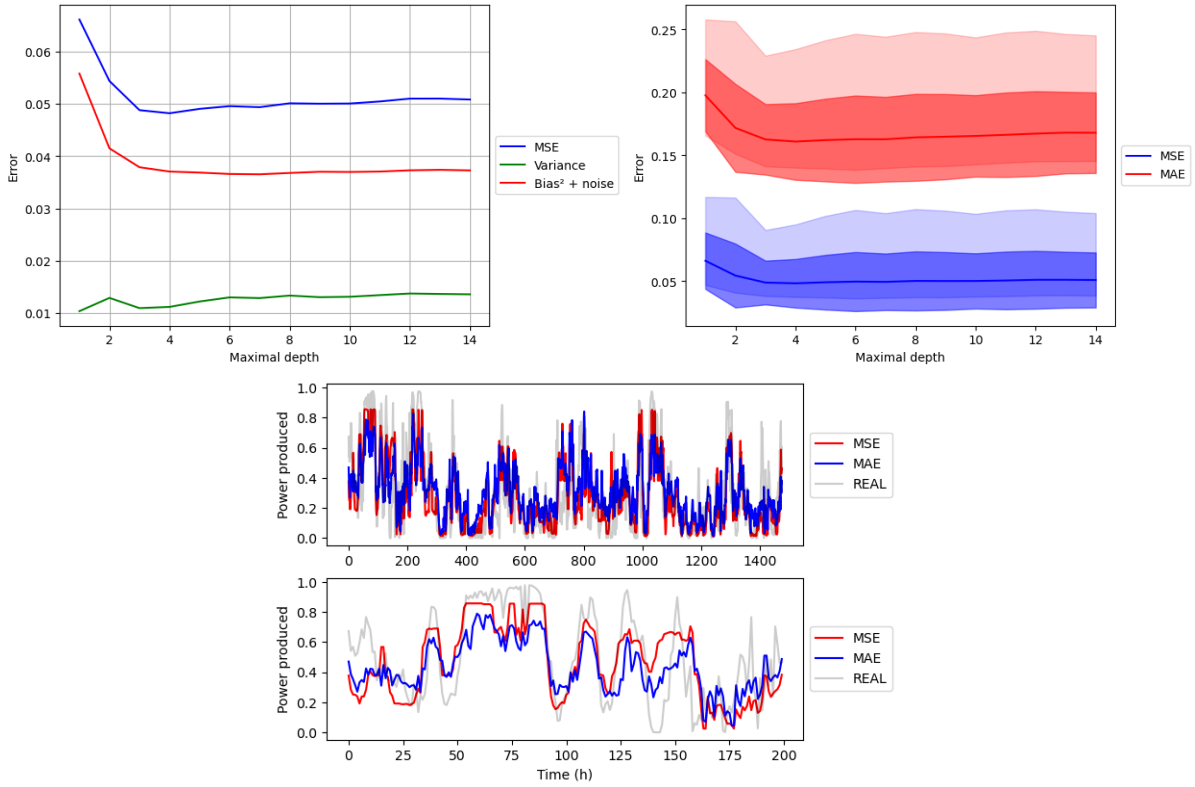


Figure 5: 1^{st} zone

The same kind of graphs is obtained for all the zones. It can be observed that increasing the maximal depth still decreases the bias but the variance doesn't increase as much as in the regression trees. It means that bias is the main contributor to the error.

Once again, the variance of the errors obtained during cross-validation is quite high.

Regarding the aspect of the predicted values, we can observe that this method gives curves that seem to be way better estimators than regression trees.

[3]We decided that we could simply set the number of estimators to the initial value even though 100 might make computations a bit time-consuming.

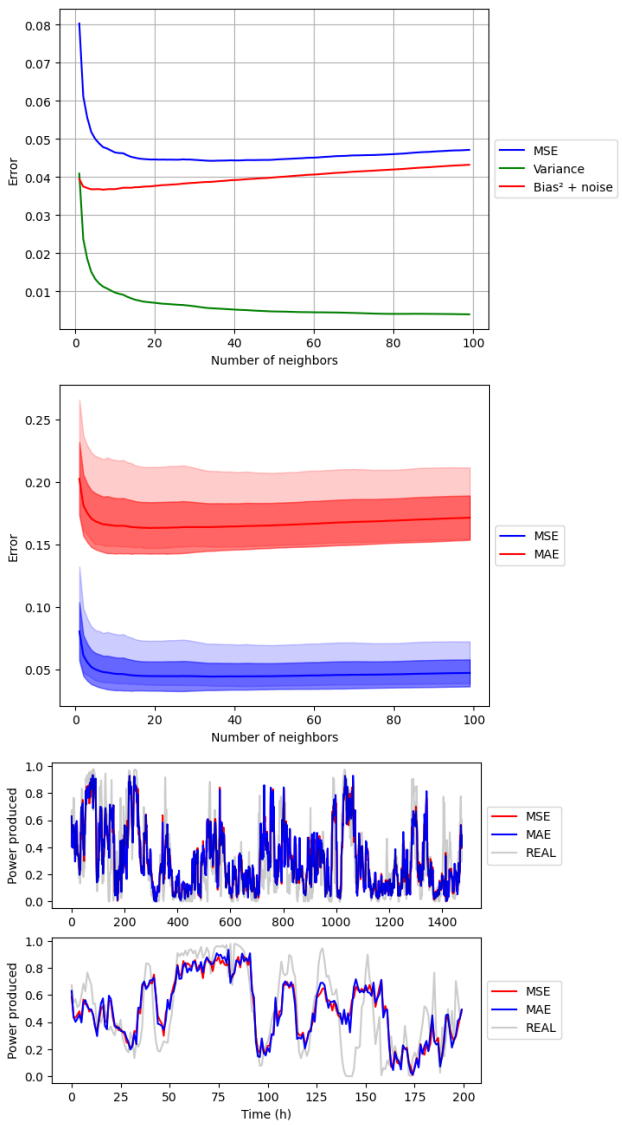
KNN

Another technique we tried is KNN. This technique consists in computing the mean of the k closest points from where we want to make the prediction.

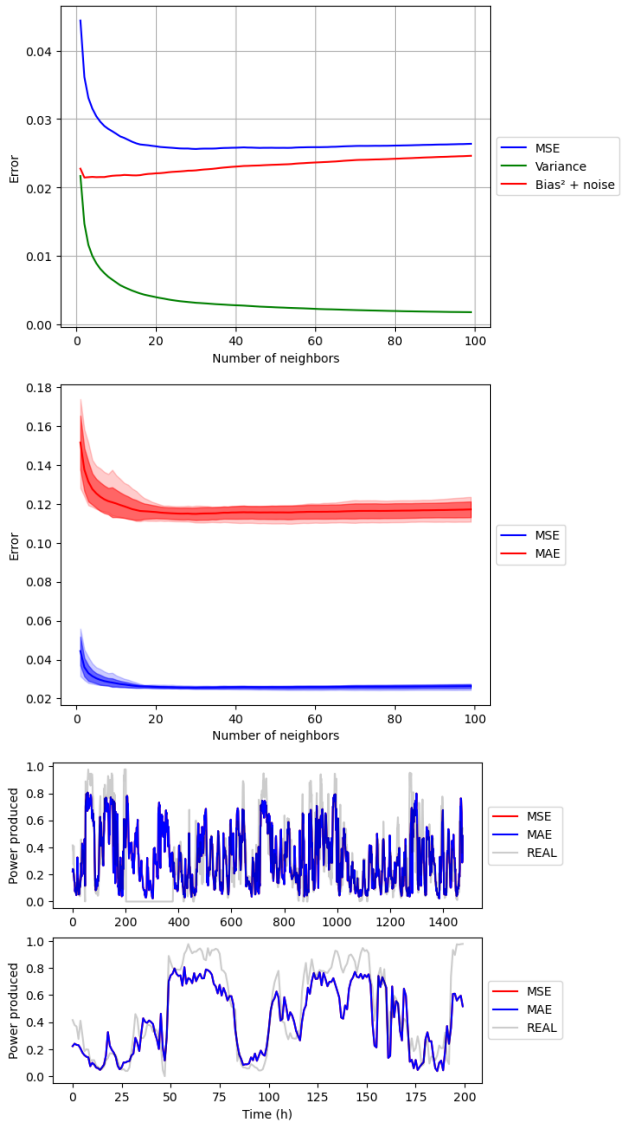
Cross-validation gave us that the best values for the k were for each zone:

$$\text{Number of neighbors} : [34, 28, 57, 34, 32, 59, 61, 92, 41, 69]$$

Examples of graphs that allowed us to get such results can be seen below:



(a) 1^{st} zone



(b) 2^{nd} zone

Since k is a measure of the inverse of the complexity, the variance decreases when it increases while the bias tends to increase. It results in an MSE decreasing at first, then slowly increasing.

The graphs clearly show that for small values of k, the estimators overfit the data (large variance and "low" bias). When k has large values, we can't detect smaller groups that can hold information (large bias, small variance).

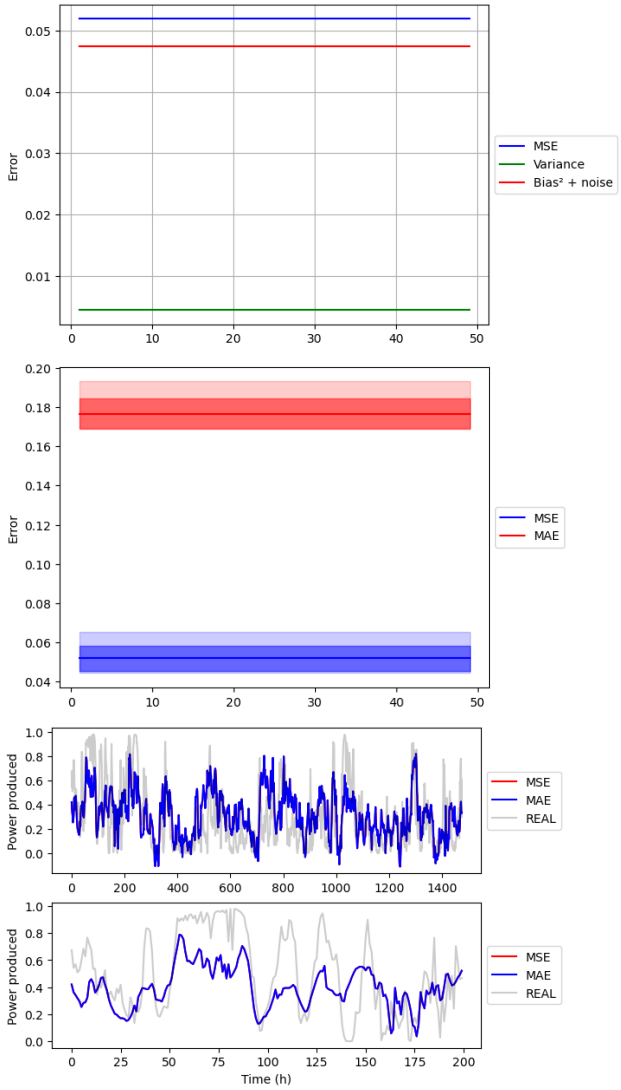
In this case, the variance in the error can change quite significantly according to the current zone.

As can be observed in Figure 6a and Figure 6b, the method has a good estimation of the initial data, and it could be interesting to use it to predict later values.

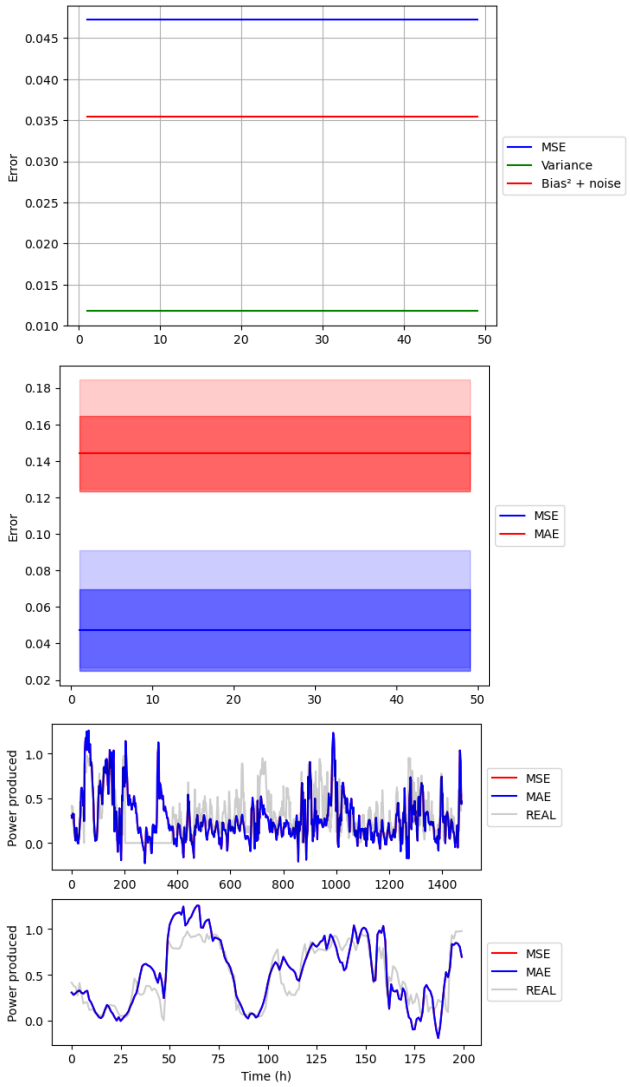
Linear regression

We also tried to use linear regression techniques.

The simplest one (without λ) simply consists in fitting a line by minimizing the residual sum of squares (RSS). This gives the following graphs for the first two zones.



(a) 1^{st} zone



(b) 2^{nd} zone

From the graphs, we can deduce that depending on the zone, we either have high or low variance in the error as in KNN. However, we only see horizontal lines in Figure7a and 7b because this method doesn't have any complexity parameter (which is why there is no name for the x-axis).

In general, the estimation can follow the trend of the data but doesn't give great approximations of the targeted values.

Ridge regression

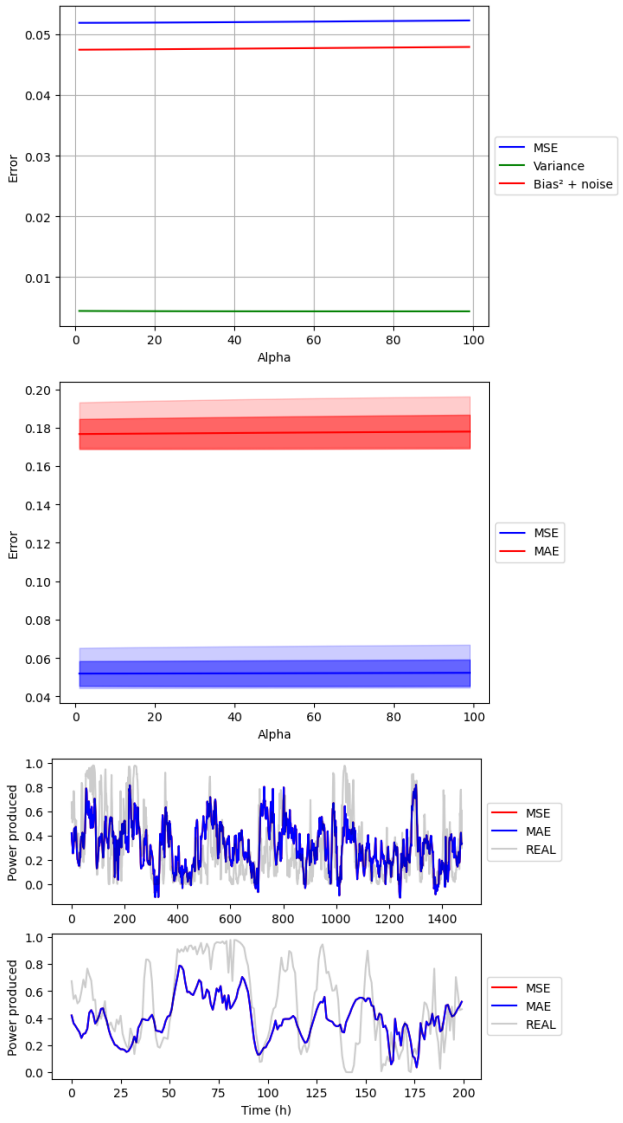
The method of Ridge regression is a modified version of the linear regression. It is still linear, but it uses the following system:

$$\min_{\beta}(\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda(\beta^T\beta - c) \qquad ; \lambda \in [0; \infty[$$

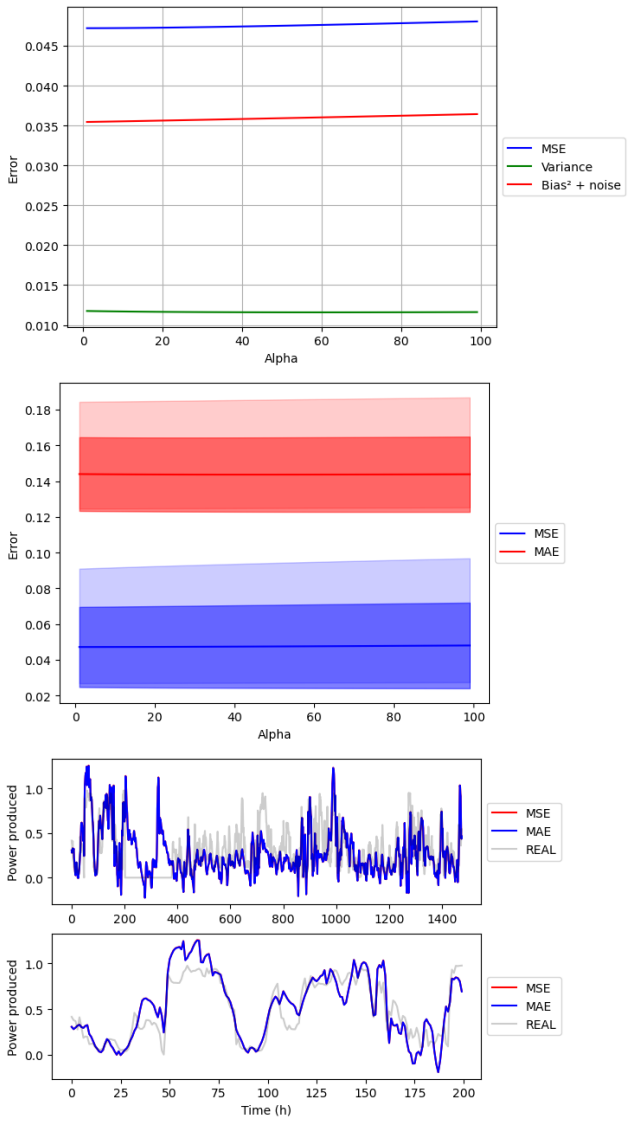
Ridge regression tries to minimize the previous expression.

The main idea behind ridge regression is to penalize large coefficients. It can even make them tend to 0 by increasing λ.

Applying the same methodology in order to determine λ gave us the following results:



(a) 1^{st} zone



(b) 2^{nd} zone

Ridge regression presents a small increase in the bias with its parameter λ but keeps a constant variance. We think that the variance remains almost the same because it is already quite small and faces some difficulties to decrease as would be expected. Indeed theoretically, we expect to have an increase in bias and a decrease in variance as λ increases.

Indeed, we can see that the larger λ gets, the smaller the sum of the square of the coefficient must be in order to be selected to minimize the loss function. They (the β's) have then less flexibility in the values they can take. This induces less variability and a higher bias. On the other hand, if λ gets really small, the β's have more possible values to possibly take which directly leads to higher variance and lower bias.

As for the linear regression, the method also follows the different trends of the data but don't get really close to the actual values.

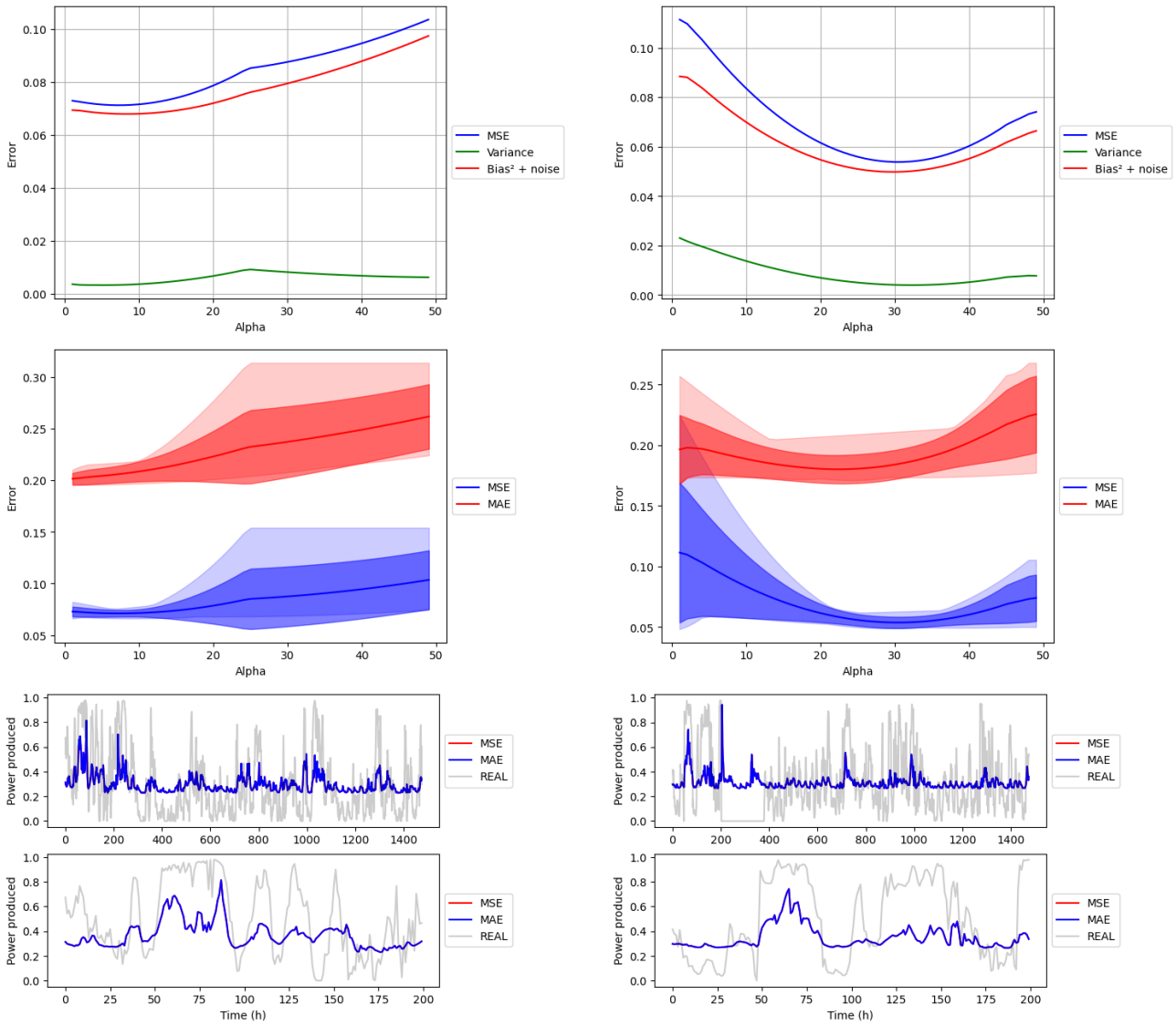
From the graphs, it can be seen that increasing the parameter λ slightly increases the MSE for each zone and the variance stays relatively constant, but it also has difficulties following the original data.

Lasso regression

Another possibility is to use Lasso. The idea behind Lasso is the same as the one from Ridge regression. However, we'll now try to use the L2 norm instead of the L1 one. This allows Lasso to exactly set some coefficients to 0. The lasso is described by the following optimization problem:

$$\min_{\beta_0,\beta} \left(\sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2 \right)$$
$$s.t. \sum_{j=1}^{p} |\beta_j| \leq \lambda$$

This time, our methodology for determining λ gave us the following results:



(a) 1^{st} zone

(b) 2^{nd} zone

Lasso regression has quite interesting graphs. Indeed, the behavior isn't what we are used to encountering. We can still observe some increase in bias when its parameter λ increases, and a decrease in variance (especially for large values). The reason behind this is the same as in ridge regression. We think that the different discontinuities correspond to values of λ at which some coefficients are fixed to 0.

The error also shows some intervals where it has a large variance, and others with a lower one. It seems quite hard to give a complete general rule for interpreting the evolution in the variance of the errors. We can only state that when the variance is minimal corresponds, we can be quite sure of how well the model will perform.

Nevertheless, this estimator can't catch the trend of the data nor give a good approximation of the actual targeted variable

Overall, we can conclude that linear methods don't fit this dataset, and won't be used for the final answer.

Another idea would be to use elastic-net methods in order to associate both Ridge and Lasso methods. This method requires us to fit 2 parameters instead of one, but because of the bad results from Lasso, we don't expect to have good results using this algorithm.

SVR

We then decided to use support vector regression(SVR) techniques. As in any regression method, SVR consists in finding a hyperplane that will fit correctly our data. In order to do so, SVR can reach a higher dimensional space by using a kernel. The final objective is to find the hyperplane for which a predefined margin[4] of size ϵ can contain as many elements as possible. If we have a learning set of inputs $\{x_i\}$ and corresponding outputs $\{y\}$, SVR will try to minimize the L2 norm of the coefficient vector $\mathbf{w}$ in the following constrained optimization problem:

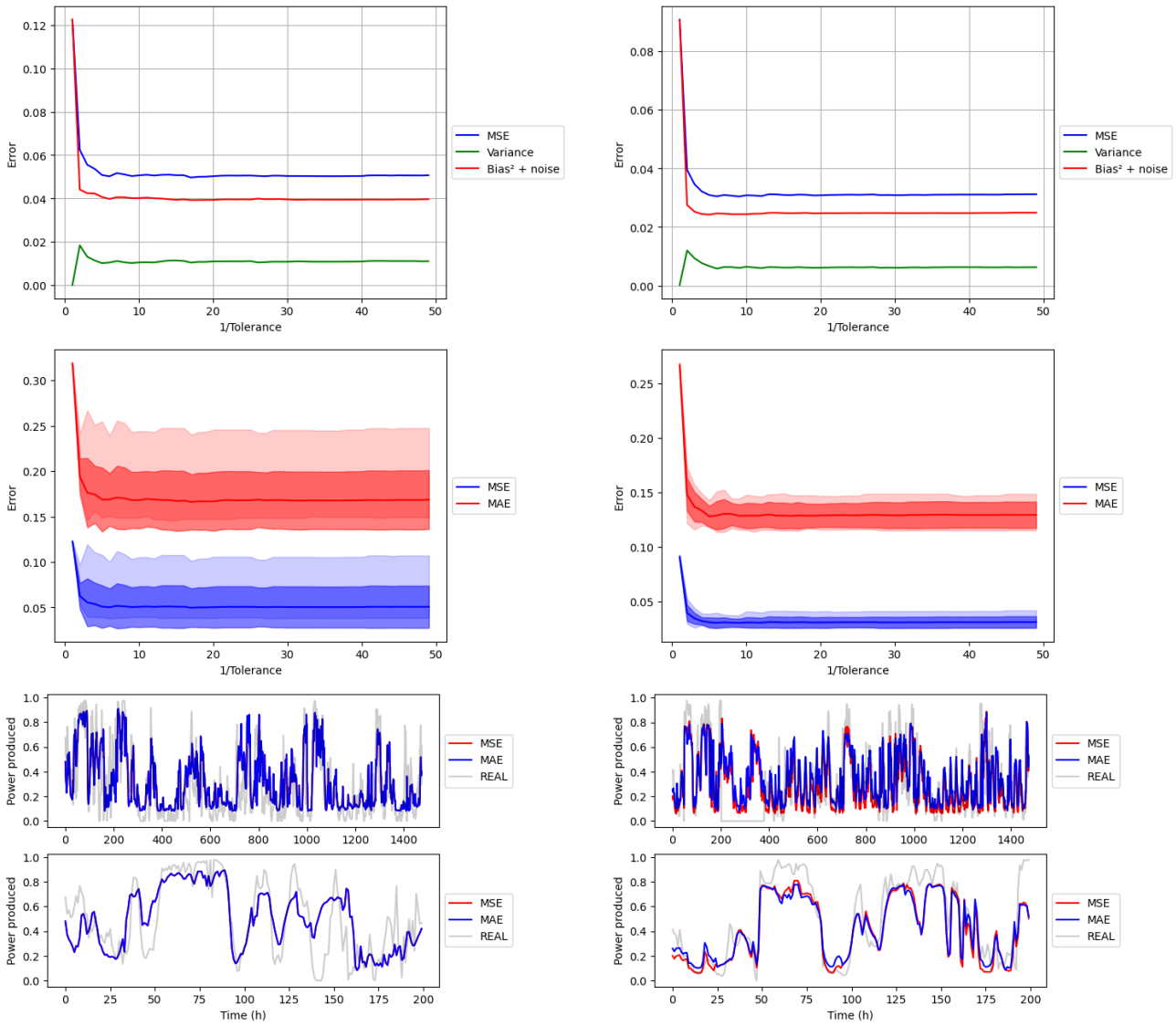
$$min(\frac{1}{2}||\mathbf{w}||_2)$$
$$s.t : |y_i - w_i x_i| \leq \epsilon$$

Since we can either change the kernel or the epsilon, we had to repeat the procedure multiple times.

We decided to use the kernel called in sklearn 'rgb' after having compared the different possibilities. Then we applied the same methodology as in the other algorithms which gave the following tolerances and figures.

$$\text{Tolerance : } [0.059, 0.111, 0.007, 0.010, 0.043, 0.111, 0.014, 0.050, 0.077, 0.071]$$

[4]It is basically an error



(a) 1^{st} zone

(b) 2^{nd} zone

The interesting parts of the graphs are at the beginning, where the variance starts to increase when the inverse of the tolerance gets really small which makes sense because it just means we accept being less precise. Most of the time, the variance and the bias are constant but at some point, further increasing the error makes the variance drop and the bias decrease. Once again, considering that increasing the tolerance means accepting less optimal hyperplanes, it totally makes sense. The obtained hyperplanes are more dependent on the data when they have to respect a small tolerance.

Regarding the error, it starts at some high value and then settles with a relatively constant variance. Once again, some differences can be observed between regions. Indeed the variance around the resulting MAE or the MSE is way larger in the first than in the second region.

Figures 10a and 10b show that the predictions can follow the trend of the data, with a good estimation of the original values.

Results

Finally, with all of those methods, we regroup the absolute error in the following table, for each zone, and then the average of those, giving an approximation of the score we can have on the submission platform.

Zone	1	2	3	4	5	6	7	8	9	10	Averages
Linear regression	0.143	0.125	0.242	0.225	0.259	0.164	0.111	0.299	0.184	0.198	0.195
Ridge	0.143	0.125	0.242	0.225	0.259	0.164	0.111	0.299	0.184	0.198	0.195
Lasso	0.165	0.184	0.238	0.233	0.269	0.229	0.167	0.275	0.205	0.252	0.222
KNN	0.124	0.107	0.239	0.217	0.255	0.146	0.093	0.291	0.181	0.183	0.184
Decision trees	0.140	0.117	0.236	0.221	0.254	0.149	0.109	0.285	0.179	0.189	0.188
Random forest	0.133	0.109	0.241	0.219	0.258	0.152	0.099	0.292	0.184	0.191	0.188
SVR	0.136	0.115	0.235	0.218	0.252	0.156	0.101	0.293	0.176	0.184	0.186

With that, we can notably say that the KNN method is appropriate to use with this dataset, since it has the lowest score among the methods used, but the SVR can also be used since their scores are close. We can also see that the Lasso regression was the least useful method, because of its high score.

Note that the linear regression and ridge regression have close scores (because of rounding errors, they have the same values), meaning that adding complexity in linear regression can be neglected for this dataset.

We then select KNN to further our approach. Indeed, we have highlighted all along the behavior of the bias and the variance. We came up to the conclusion that the main bother for having great predictions was coming from the bias. However, we can see in different figures that this bias increases very slowly for different zones. What's more sometimes, the variance of the error around the MAE or the MSE was quite small. It leads us to consider increasing k quite a lot for regions 2, 7 and 8. Indeed doing so will make our model simpler. We just need to make sure not to make it so simple that we encounter underfitting. That was one great idea because adding this with the different features we introduced before lead us to the 3^{rd} position in the competition.

Our final k's are [60, 90, 60, 120, 60, 50, 80, 3660, 400, 500].

In the end, we got a global MAE of 0.1807325

Further work

Until now, we haven't really taken into account the time dependency aspect of our data. Indeed, we have taken it a bit into account when making new features and we think it could be used a bit more. We even think it could be a great idea in order to decrease the bias that is so annoying. However, since we already had great results and we have other time restrictions to respect, we didn't have time to further investigate what was laid after. We leave it as possible further work.

First idea

The first thing that spotted our attention is the "high sampling frequency" of our data. Since we are analyzing a macroscopic physical process and then a physically deterministic process, it might be possible to take advantage of causality. The simple way to do so is to compute the difference of our targeted variable between two consecutive time steps. Then, we considered it would be a great idea to have a look at the distribution of the differences between the succeeding targeted values. This lead to a histogram from Figure11.

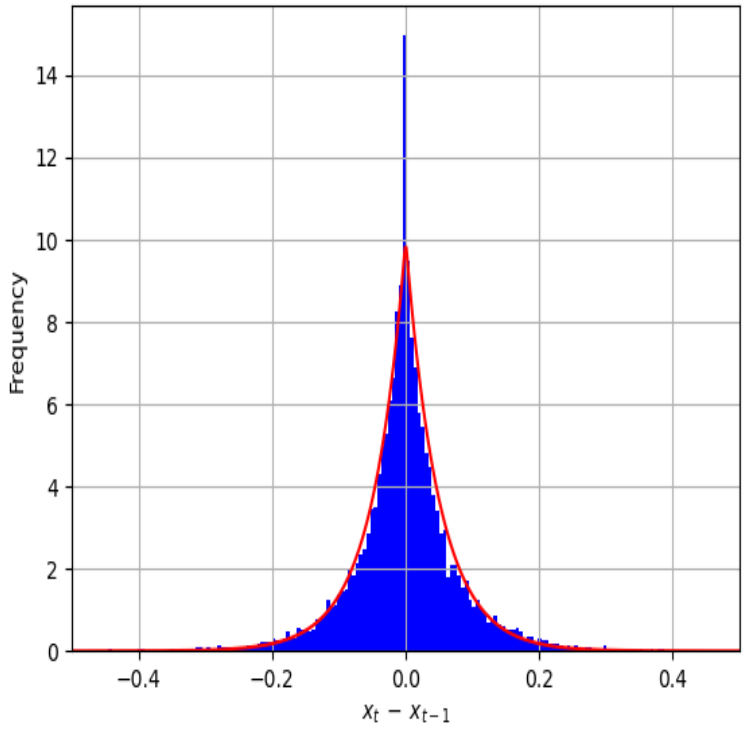


Figure 11: Histogram of the different following values

We can see that this distribution is extremely similar to the Laplace(0, 0.05) drawn in red in the figure above. If we then consider a first-order Markov process as a great representation of our situation, we could use that to make a prediction. Indeed, since we consider Markov chains, we have the target value at time t x_t is only dependent on the previous value x_{t-1}:

$$P(x_t|x_{0:t-1}) = P(x_t|x_{t-1}) \qquad \text{and} \qquad \mathrm{x_t} - \mathrm{x_{t-1}} \sim \mathrm{Laplace}(0, 0.05)$$

This allows us to compute the log-likelihood of a series of data of size T:

$$\mathcal{L} = \prod_{i=1}^{T+1} f(x_t|x_{t-1})$$
$$l = \sum_{i=1}^{T+1} log(f(x_t|x_{t-1}))$$

where $f(x_t|x_{t-1})$ is the PDF of the Laplace distribution centered at the origin for $x_t - x_{t-1}$

Note that we also considered the point directly before and directly after the data that has to be predicted.

A simple technique would be to maximize the log-likelihood to estimate the points. The issue is that we have long series of unknown data. This would make uncertainty spread in the whole predictions and then lead to poor results[5]. However, this technique would be a great idea if we had a very small series of unknown data.

Even though we won't use this technique alone, it might be a great idea to further increase the precision of other models.

Indeed, let's consider one of our models has found the series $\mathcal{S} = \{x_t | t \in 1, T\}$. We can create a loss function that will try to adjust as well as possible by adjusting the variables by taking into account the previous time series and the MLE.

Let a and b be two coefficients and $\mathcal{M}$ be another time series corresponding to the same instants than $\mathcal{S}$, we can build the following loss function in the following way:

$$
\begin{aligned}
d &= MSE(\mathcal{M}, \mathcal{S}) \\
l &= \sum_{i=1}^{T+1} log(f(x_t|x_{t-1})) \\
Loss &= -a\, l + b\, d
\end{aligned}
$$

Using an optimization function such as "optimize" could then give us better results.

Another simple idea would be to combine different methods together.

Other approaches

We could still use other algorithms. We think that using common time series forecasting algorithms such as SARIMA, transformer model or maybe even filtering high frequencies from an FFT might be useful in order to predict the values. From what we have done, we realized that the time interval in which we had to make predictions might be a bit too long for that. Indeed, the further we go into unknown data, the less precise we get. It might still be useful for the first few values to find though.

[5] Most of the time the predictions would be close to the previous value.