LELEC2870 - Project in Machine learning

Prediction of air-quality in Beijing

Group AM: Martin Braquet (06641500), Amaury Gouverneur (69331500)

Friday 20th December, 2019

I. Introduction

This report aims in predicting the concentration of PM2.5 in the air of Beijing. This is done using regression models on a dataset of 7684 records of meteorological and weather data from March 1st 2013 to February 28th 2017. The 15 recorded input features are the following:

- time [year, month, day, hour],
- SO2, NO2, CO, O3, concentrations [$\mu g/m^3$],
- temperature and dew point temperature [C°],
- pressure [hPa],
- rain precipitation [mm],
- wind direction [cardinal] and speed [m/s],
- id of the air-quality. monitoring site.

The recorded output variable is the corresponding PM2.5 concentration [$\mu g/m^3$].

This paper is organized as follows: features processing, selection and extraction will be discussed in sections II, III and IV, the error estimation and the models implementations are presented in section V and VI before concluding in section VII.

II. Features processing

The time feature recorded in the year, month, day, hour format is converted in seconds in the variable `time` with $\texttt{time} = 0$ corresponding to the first record. This format is more usable but however does not express the cyclic relations in time, namely the rotation of the earth around the sun and the rotation of the earth around itself (see Figure 1). To do so 4 extra variables are created: `syear` and `cyear`, encoding the progress of the earth rotation around the sun, and `sday` and `cday`, encoding the progress of the earth rotation around itself. The values are computed as follows:

$$\begin{aligned}
\texttt{syear} &= \sin\left(2\pi\frac{\texttt{time}}{365\cdot 24\cdot 60\cdot 60}\right),\\
\texttt{cyear} &= \cos\left(2\pi\frac{\texttt{time}}{365\cdot 24\cdot 60\cdot 60}\right),\\
\texttt{sday} &= \sin\left(2\pi\frac{\texttt{time}}{24\cdot 60\cdot 60}\right),\\
\texttt{sday} &= \cos\left(2\pi\frac{\texttt{time}}{24\cdot 60\cdot 60}\right).
\end{aligned}$$

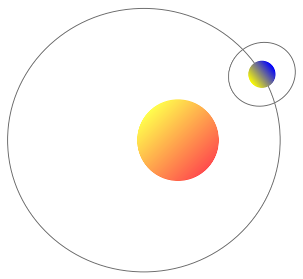

Fig. 1. Earth revolution around the sun

The wind direction recorded as cardinals directions are also translated in a format expressing the cyclic relation, `swd` and `cwd`, respectively the sine and cosine of the angle of the cardinal direction on a wind rose (see Figure 2).

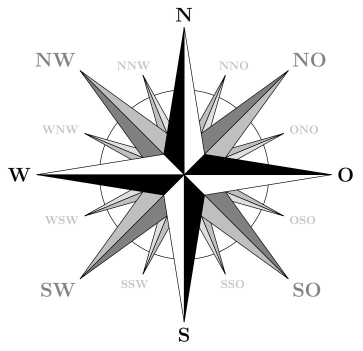

Fig. 2. Wind rose

The complete set of inputs is thus composed of 17 features. The set of inputs is then normalized using a standard normalization method.

III. Features selection

Features selection is a technique to drop some less useful inputs. The intrinsic principle aims to maximize the relevance (relation between input and output) and minimize the redundancy (relation between input and input).

It is important to consider the newly created features (cos/sin) in the previous paragraph as a pair of inputs resulting from one unique feature, it is thus not advised to remove one of them (the sine or cosine) even if they present a high dependency.

A. Correlation

Figure 3 presents the correlation between the inputs and the output (absolute value). Since the correlation only detects linear relations between variables, a zero correlation between an input and the output is not sufficient to drop this input.

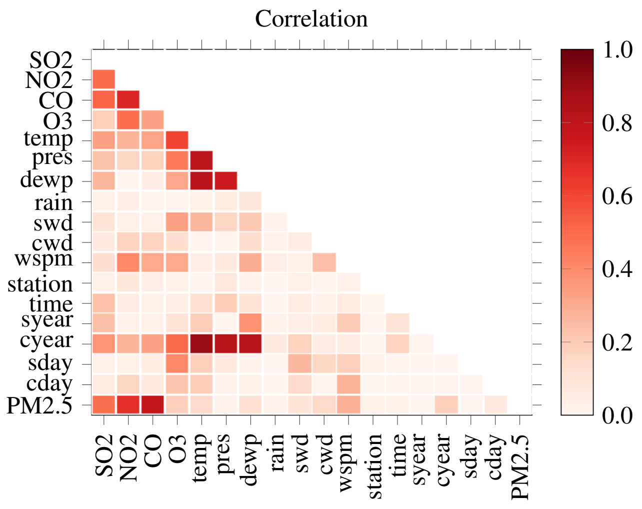

Fig. 3. Correlation matrix between the inputs and the output

B. Mutual information

Figure 4 shows the mutual information between the inputs and the output (absolute value). This method is able to detect any dependency between variables and is thus particularly purposeful to select the right features.

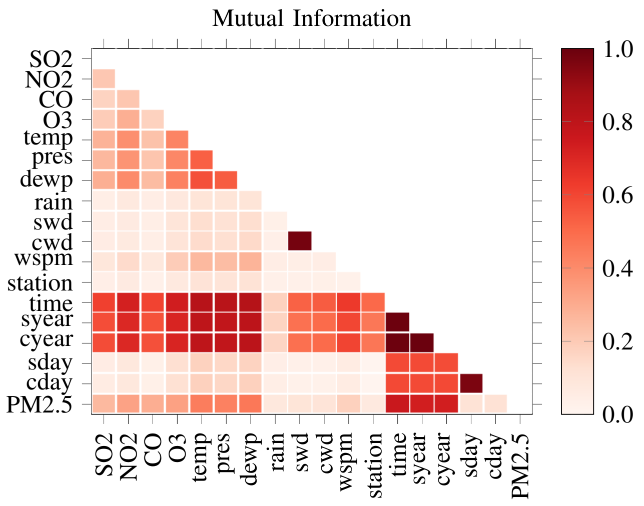

Fig. 4. Mutual Information matrix between the inputs and the output

C. Redundancy and relevance

Based on the correlation and MI obtained above, one can drop the features in Table I because they have few relevance with the output.

Feature	MI with output
station	0.074
rain	0.079
swd	0.098
cwd	0.1
sday	0.11
cday	0.11
wspm	0.18

TABLE I
Non relevant features

Additionally, it is possible to remove the temperature (see Table II), the pressure (see Table III) and the time (see Table IV) since they are redundant with other inputs.

Feature	MI	Correlation
dewp	0.57	0.81
cyear	0.79	0.9

TABLE II
Relations between some features and a redundant feature: Temperature

Feature	MI	Correlation
dewp	0.54	0.74
cyear	0.79	0.82

TABLE III
Relations between some features and a redundant feature: Pressure

It is worth noting that the dependencies between the time and the other features is non linear, such that a small correlation is computed.

Feature	MI	Correlation
`syear`	0.99	0.1
`cyear`	0.99	0.17

TABLE IV
RELATIONS BETWEEN SOME FEATURES AND A REDUNDANT FEATURE: TIME

Finally, the set resulting of features selection contains 7 features.

IV. FEATURES EXTRACTION

Features extraction consists to transform the input into other features such that this new smaller set of features almost fully describes the content of the inputs.

A. *Principal Component Analysis*

The Principal Component Analysis method uses an orthogonal transformation to convert a set of inputs into a set of linearly uncorrelated variables called principal components [1].

Figure 5 depicts the error for several numbers of principal components, ranging from 1 to 17 (the full number of features). One can conclude that the most important correlation of the data are combined in 3 principal components, lowering the error to $55.5\,\mu\mathrm{g/m^3}$. Adding more components does not lead to better results considering that having only 3 input features brings very efficient computations.

However, it is important to note that this method is linear and thus drops important variables if they are non-linearly dependent. This problem depends upon the linear characteristics of the input features. It is consequent for the data analysed here and hence leads to bad results as discussed thereafter.

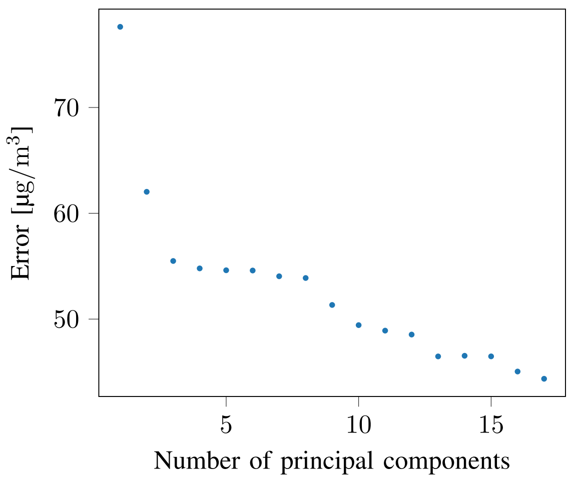

Fig. 5. Bootstrap 632 error for different numbers of principal components

V. ERROR ESTIMATION

The errors of the following models are estimated thanks to the Bootstrap 632 method gently provided in the MLxtend package [2]. This highly efficient method is characterized by a low bias and a low variance, which makes it particularly purposeful for model validation (compared to the simpler K-fold validation for instance). If not mentioned in the following sections, the number of splits in the Bootstrap method is set to 10 due to the limited available computation power in classical computers. This small number only influences the randomness of the bootstrap error but not the mean value (most important part).

Figure 6 shows the error distribution of the linear regression model with the selected features, for 1 and 10 splits. The error is very narrow for 10 splits ($\sigma = 0.255\,\mu\mathrm{g/m^3}$) compared to 1 split ($\sigma = 0.762\,\mu\mathrm{g/m^3}$), computing the error with 10 splits is thus cogent.

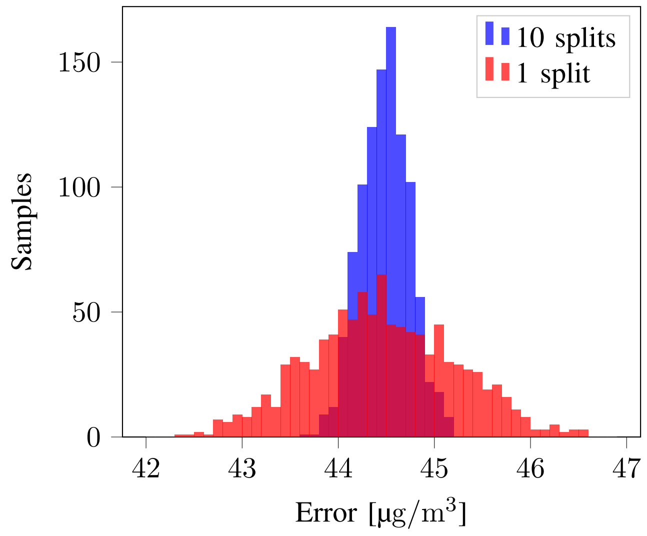

Fig. 6. Bootstrap 632 error distribution for different numbers of splits

VI. MODELS IMPLEMENTATIONS

Seven models have been trained and compared in order to bring the most powerful one for the estimation of the secret dataset. These models are tested with the bootstrap 632 error on the three previously detailed input sets: the full features, the selected features and the PCA features.

Let the dataset be $D = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ where x_i is the input value and y_i is the target value, the goal of a regression model is to estimate the function $y = f(x)$.

A. Linear regression

The linear regression estimates the function $y = f(x)$ by $\hat{f}(x) = w^T x$. The weight vector w is computed by minimizing the mean squared error on the training data:

$$\min_{\mathbf{w}} ||\mathbf{y}_{\text{train}} - \mathbf{w}^T \mathbf{X}_{\text{train}}||_2^2.$$

The error for the three implemented sets is given in Table V. One can see that the 7 selected features give almost the same error as for the full input set, which validates the features selection (at least for linear models). However, the PCA method, although very fast, is not convincing because it has few features.

Features	Error [μg/m^3]
Full	44.35
Selected	44.50
PCA	54.10

TABLE V
BOOTSTRAP 632 ERROR FOR THE LINEAR REGRESSION MODEL

B. Ridge regression

The ridge regression is similar to the linear regression with a shrinkage penalty on $\mathbf{w}$ (L_2 regularization):

$$\min_{\mathbf{w}} ||\mathbf{y}_{\text{train}} - \mathbf{w}^T \mathbf{X}_{\text{train}}||_2^2 + \lambda||\mathbf{w}||_2^2.$$

The parameter λ controls the relative impact of the two terms. Increasing λ decreases the variance while increasing the bias.

As shown in Figure 7, the ridge regression performs better with the *selected features* and *full features* than with the *PCA features*. The performance is almost constant for $\lambda \in [10^{-2}, 10^2]$ and decreases after when the model starts to suffer from underfitting. The best result is achieved with the *full features* and $\lambda = 0.61$, the error is $44.15\,\mu\text{g/m}^3$.

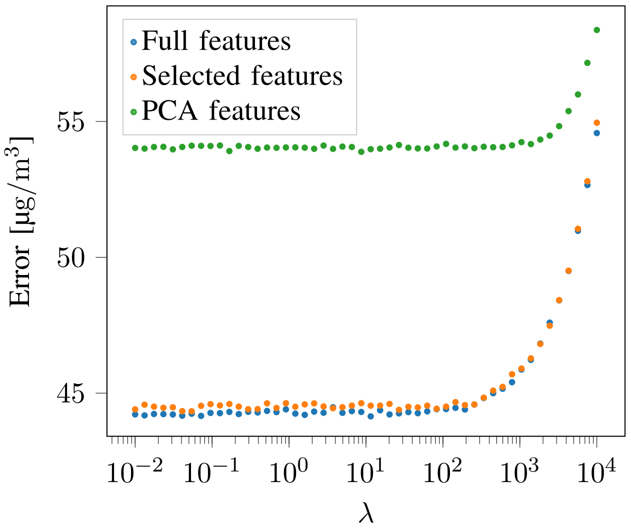

Fig. 7. Bootstrap 632 error for the Ridge regression model

C. Lasso

The Lasso regression is similar to the ridge regression with a shrinkage penalty on $\mathbf{w}$ (L_1 regularization):

$$\min_{\mathbf{w}} ||\mathbf{y}_{\text{train}} - \mathbf{w}^T \mathbf{X}_{\text{train}}||_2^2 + \lambda \sum_{i=1}^{N} |\mathbf{w_i}|.$$

As shown in Figure 8, the lasso regression performances are almost constant until $\lambda = 1$ and start to decrease after because of underfitting. Similarly to the ridge regression the Lasso performs better with the *full* and *selected features*. The best result is achieved with the *full features* and $\lambda = 0.01$, the error is $44.06\,\mu\text{g/m}^3$.

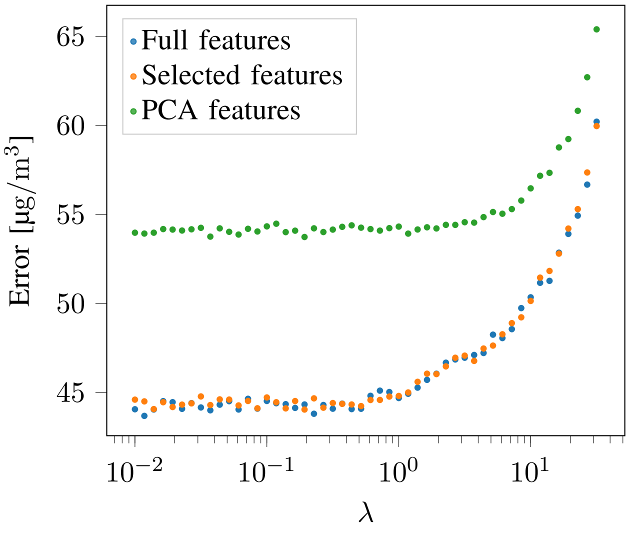

Fig. 8. Bootstrap 632 error for the Lasso model

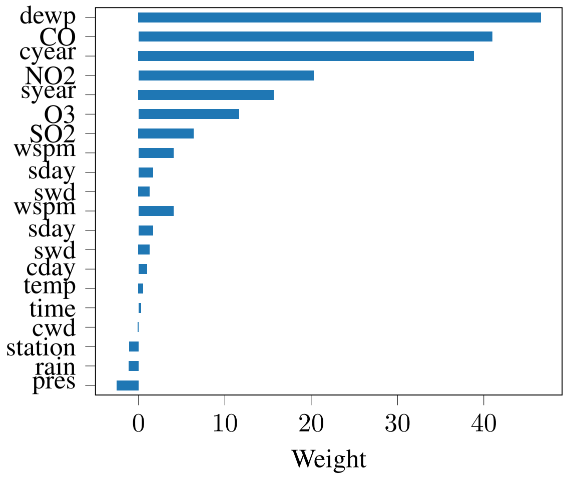

Fig. 9. Weights of the Lasso model ($\lambda = 0.1$)

The weights for $\lambda = 0.1$ are given in Figure 9. As found in the features selection, the output is very sensitive to three features, and more or less 7 features completely describe the input space. Some coefficients end up being set to almost zero, making the model easier to interpret.

As both the Ridge regression and Lasso models are better with a small regularization coefficient, they are redundant with the linear regression. This can be explained by the high number of samples used for the training, which are sufficient to prevent overfitting in the linear regression model. These two additional models cannot exploit their advantage of reducing the linear regression overfitting.

D. *K-Nearest Neighbour*

Figure 10 presents the error for the KNN model with respect to the number of neighbours K (hyperparameter). For this model, using the full input set should dramatically increase the error since in a high dimensional space, the K nearest neighbours (with the Euclidian norm) are greatly influenced by useless dimensions and thus lead to select wrong neighbours. As expected, the model with features selection outperforms the others, reaching an error of $38.51\,\mu\mathrm{g/m^3}$ for $K = 4$.

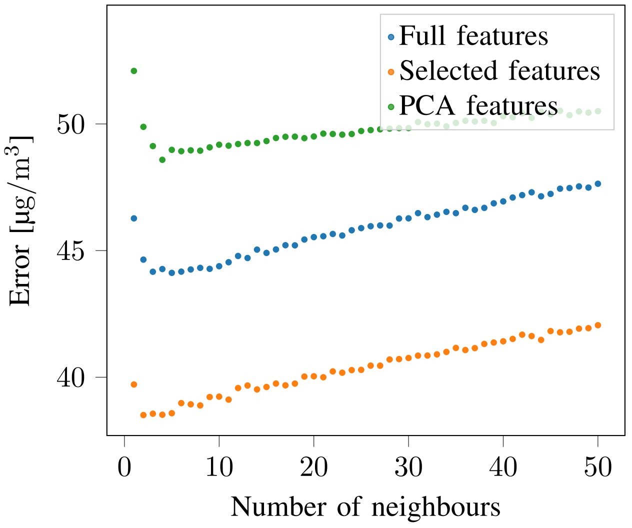

Fig. 10. Bootstrap 632 error for the KNN model

E. *Regression tree*

A regression tree partitions the input space in smaller regions recursively until reaching a leaf where a simple model is fitted to. Each node represents a binary questions about one or several feature(s) that will divide the space in two. For a classic regression tree, the model at each leaf is a constant estimate: the average of the target value of the training data that belongs to this leaf. The binary questions on the nodes are chosen such that the information gain is maximized. Mechanisms such as pruning are necessary to avoid overfitting.

As shown in Figure 11, the performance obtained by the *full* and *selected features* are very close and outperform the ones obtained with the *PCA features*. The performances increase as the maximal depth of the tree increases until reaching a maximum at 7 for the *PCA features*, then it starts to overfit since there are too few features in PCA. The error obtained with the *full* and *selected features* stops decreasing and starts to level out at a depth of around 10. It has to be noted than the trees do not grow further than a depth of 33, explaining the absence of overfitting as the allowed maximal depth continues to increase. The best result is achieved with the *selected features* and a depth of 11, the error is $40.23\,\mu\mathrm{g/m^3}$.

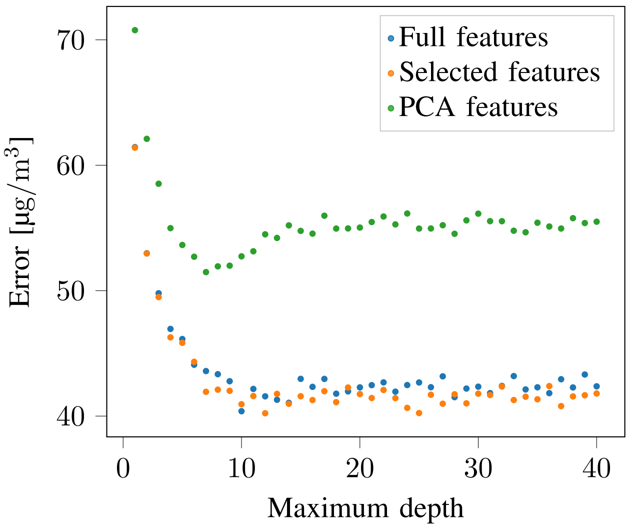

Fig. 11. Bootstrap 632 error for the regression tree model

F. *Bootstrap aggregating trees*

The core idea of an ensemble method is to train weak regressors (or classifiers) and combine them to construct a regressor (or classifier) more powerful than any of the individual ones.

A simple ensemble method is the so-called Bootstrap aggregating or Bagging. Let $D = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ be the training data, the principle of the Bagging method is detailed below.

Iteratively, for $b = 1, \ldots, B$, do the following:

- sample training examples, with replacement, N' times from D to create $D_b (N' \leq N)$,

- use this bootstrap sample D_b to estimate the regression (or classification) function f_b.

The bagging estimate is

$$f_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x).$$

As shown in Figure 12, the *full* and *selected features* are very close and outperform the ones obtained with the *PCA features*. The error decreases as the maximum depth increases until starting to plateau at a depth of 12 with no overfitting. As for the regression tree, this is due to the fact that the trees do not grow deeper than 33. The best result is achieved with the *selected features* and a depth of 19, the error obtained in this case is $31.42\,\mu\text{g/m}^3$.

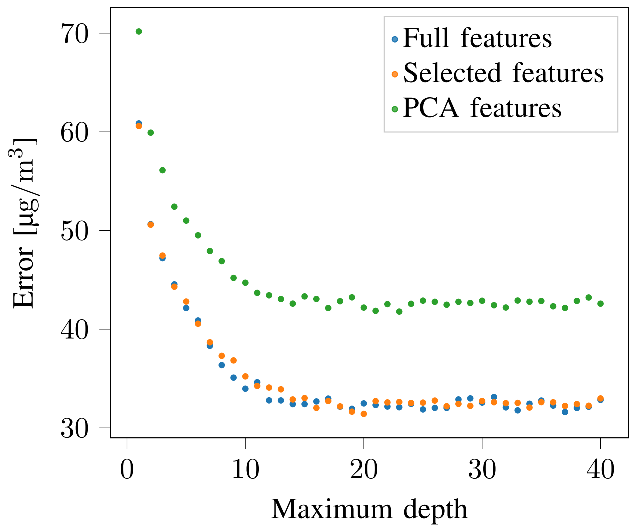

Fig. 12. Bootstrap 632 error for the Bagging trees model

G. Multilayer Perceptron

A multilayer perceptron takes as inputs the features (17 for the full set and 7 for the selected/PCA set) and propagates these information though a deep neural network to output the final PM2.5 estimation. At each layer, the data are subject to a batch normalisation followed by a ReLU activation function.

Considering the high number of parameters which are subject to optimization (number of hidden layers, neurons per hidden layer, epochs, learning rate, batch size), it might seem interesting to use an optimization algorithm such as a greedy search or a genetic search. However, it appears after simulation that finding this optimum is difficult since the performances are better when the network is very large, in such a way that overfitting is only noticed for big neural networks which are impossible to train in a limited time. The following paragraphs will thus be dedicated to analysing the error for several numbers of neurons per layer, epochs per training period, and hidden layers.

The results presented in Figure 13 depict the error variation for different numbers of neurons (hyperparameter) in each of the 8 hidden layers, ranging from 1 to 50. The learning step is done by feeding the network with the inputs/output pairs, more precisely with 50 epochs and a batch size of 128. It can be seen that the neural network performs the best with the full features as inputs since the weights of the network are precisely adapted to select the information in each features, even the smallest one. On the other hand, deep neural networks do not aim at dropping some features, which is proven by the poor results from features selection and features extraction.

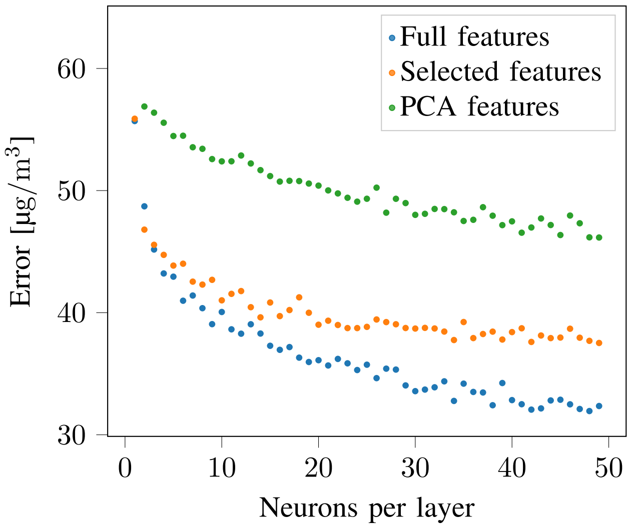

Fig. 13. Bootstrap 632 error for the MLP model with different numbers of neurons per hidden layer

One can also analyse the error for different training periods, characterised by a varying number of epochs. Figure 14 shows this error with a number of epochs ranging from 1 to 80. The error is stabilized after around 50 epochs for all inputs subsets since all the neural networks weights do not vary anymore. The error does not increase for additional epochs, leading to the conclusion that the network configuration is too small to bring overfitting due to long training periods[1].

[1]In case of possible overfitting, the early stop approach consists to stop the training when the validation error reaches a minimum.

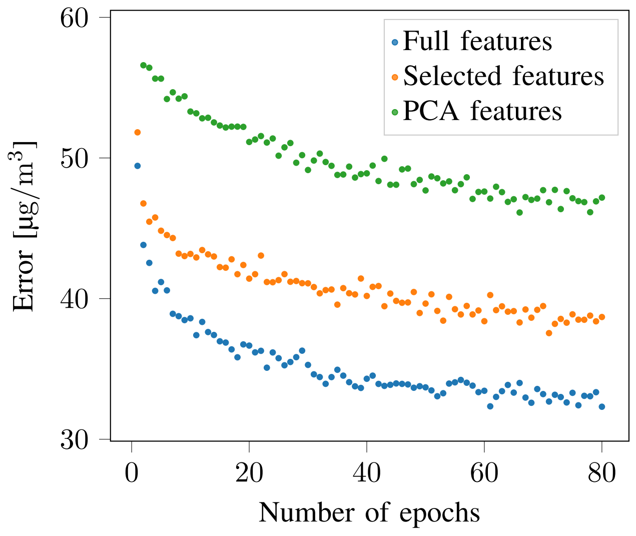

Fig. 14. Bootstrap 632 error for the MLP model with different numbers of epochs per training period

The next step thus consists to increase the number of hidden layers and reduce the error as far as possible. Figure 15 presents this error for the full set, 50 neurons per hidden layer and a number of epochs proportional to the number of layers since larger networks require a longer training period. One can see a minimum error for 18 hidden layers but no overfitting appears as it should be expected for such large networks. Hence, overfitting should arise for larger networks.

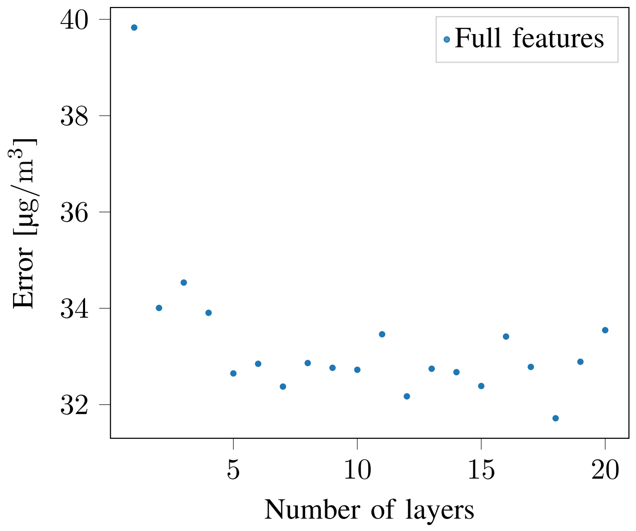

Fig. 15. Bootstrap 632 error for the MLP model with different numbers of hidden layers

Finally, a huge neural network of 20 hidden layers and 100 neurons has been trained with 300 epochs. It gives a bootstrap 632 error of 34.12 $\mu g/m^3$, suggesting the start of a small overfitting.

It can be noted that the PyTorch library has been used, leading the authors to implement an estimator in order to match the specifications of the bootstrap 632 method [3].

VII. Conclusion

To conclude, the lowest error for the analysed models are summarized in Table VI (the errors are expressed in $\mu g/m^3$).

Model	Error (Full)	Error (Selected)	Error (PCA)
Linear regression	44.35	44.50	54.10
Ridge regression	44.15	44.38	53.91
Lasso	44.06	44.07	53.92
KNN	44.12	38.51	48.58
Regression tree	40.39	40.23	51.48
Boot. agg. trees	31.60	31.42	41.78
MLP	31.94	37.52	46.16

TABLE VI
Summarized error for all the models

A. *Final model selection*

The bootstrap aggregating trees method with the features selection is chosen as the final method since it has the lowest error. This model is used to predict the output of the secret set, `Y2`. The bootstrap 632 method aims to estimate the error on a set belonging to the entire possible space. Hence, the expected RMS error on `Y2` is the bootstrap 632 error on `Y1`: 31.42 $\mu g/m^3$.

Finally, combining the previous model predictions by means of ensemble methods or a voting classifier are good perspectives for a future work in the prediction improvements and the quest for the lowest error.

References

[1] Wikipedia. *Principal component analysis*. 2019. URL: https://en.wikipedia.org/wiki/Principal_component_analysis.

[2] Sebastian Raschka. “MLxtend: Providing machine learning and data science utilities and extensions to Python’s scientific computing stack”. In: *The Journal of Open Source Software* 3.24 (Apr. 2018). DOI: 10.21105/joss.00638. URL: http://joss.theoj.org/papers/10.21105/joss.00638.

[3] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.