Predicting Air Quality in Paris

Kaggle Challenge Report

Group Members: Elise Barattini, Benjamin Cerf, Matthieu Delsart, Malo Renaudin, Augustin de Saint-Affrique, Eva Toledano - **Github Repository:** https://github.com/matthieudelsart/time_series_project

Introduction

This Kaggle challenge consisted of a multivariate time series problem. It focused on forecasting hourly concentrations of five air pollutants in Paris by leveraging historical and external data sources.

Data Preprocessing

Our preprocessing phase occurred in several phases:

1. Data cleaning
2. Exploratory Data Analysis (EDA)
3. Addition of external data
4. Feature engineering
5. Missing data imputation

1. Data cleaning. The first step in cleaning the data consisted in converting the *id* column into Pandas's datetime format. We then checked for outliers in the target distribution by using box plots and violin plots. While we started by removing the most extreme values by using the Interquartile Range method [1], we finally decided to use all the data as there was no obviously wrong value.

2. EDA. Our EDA uncovered 3 main insights:

- A visible cyclicity of targets, with obvious daily and yearly patterns. This was reinforced by the autocorrelation plot, which suggested 1h, 2h, 12h, 24h lags for example
- A strong interdependence of targets, with correlation reaching up to 0.9, meaning we would have to model them together
- Numerous missing values in the targets

3. Addition of external data. As air pollution is likely to be linked with external factors, we decided to complete the data with 3 main sources:

○ *Holidays data*. As we thought that air pollution was likely to be connected to human activity, we decided to add binary features corresponding to school vacation and public holidays. These we retrieved thanks to [2]

○ *Weather data.* Air pollution is often linked to weather conditions, which can influence the concentration of spreading of pollutants. Here, the main challenge was to find a free, easy-to-use, reliable external data provider. We finally opted for Meteo France data records [3]. We then filtered on the Montsouris observatory, which is the main observatory for Paris, and we selected the features we deemed likely to be the most important: *temperature*, *humidity*, *wind speed*, *precipitation*, *atmospheric pressure,* and *global solar radiation* (as a proxy for the UV intensity). Some of these features showed very strong correlations with the targets, which was encouraging.

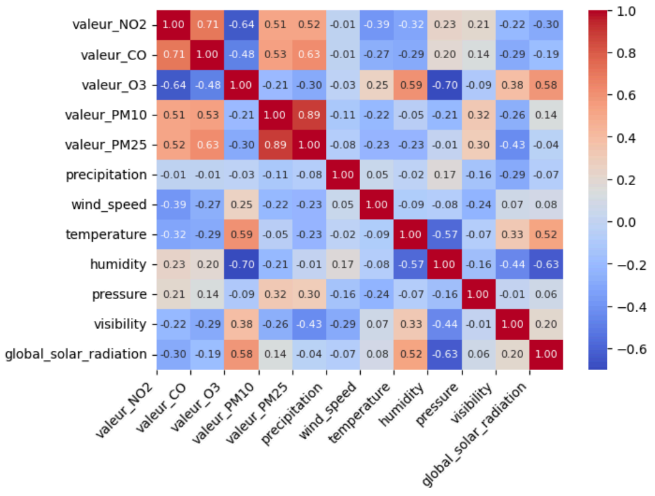


Fig. 1. Correlation matrix between the targets and weather features

○ *Traffic data*. Traffic emissions are one of the primary sources of pollution in Paris, hence the addition of an hourly average traffic metric for Paris over the period. For this purpose, we used the Comptage routier dataset from Open Data Paris [4]. For better data consistency over the period, we averaged the hourly traffic on active counters that were operational throughout most of the period and had minimal downtime.

4. Feature Engineering. Coherently with our EDA, our main feature engineering focus aimed to better represent temporality. Even though we knew that some advanced models could deal without these afterwards, we created 3 main types of features: (i)

Extracted raw temporal features from the date (year, month, day of the week, etc.) (ii) Trigonometric-encoded temporal features to represent cyclicity, especially for the hour, the day of the week and the day of the year (iii) Lag features, both on the target variables (to represent temporal continuity and cyclicity) and the features (as we thought some features could have a delayed effect, for instance between UV intensity and pollutant concentration). We did not keep all these lag features when training but they overall proved very helpful to improve prediction.

5. Imputation of missing values. As seen in the EDA, an essential point of preprocessing, especially on a time-series problem, was to fill in the missing values of our target variables. We had noticed that all targets had missing values, especially carbon monoxide (CO), for which a very high number of values were missing. We used the Qolmat library [5], which uses scikit-learn to compare several data imputation techniques. Thanks to their comparison function, we tested different imputation techniques:

- *Median:* fills missing values with the median of the series
- *Interpolation:* estimates missing values by interpolating neighboring values
- *Residuals:* predicts missing data based on residuals
- *TSOU (Time Series Optimal Unbiased):* time-series specific optimization method
- *MICE (Multiple Imputation by Chained Equations)*: estimates missing values iteratively based on dependencies among features

We looked at 4 metrics: MAE (mean absolute error), WMAPE (weighted mean absolute percentage error), Kullback-Leibler divergence, and Wasserstein distance between the distributions. Among these, the TSOU method appeared to perform best. We then tested two different methods for TSOU: sample (random sampling for imputation) and MLE (Maximum Likelihood Estimation), which performed best, so we selected it for the final imputation of values.

We first used this method only on the target variables to fill the majority of missing values. Having in mind that the last values of the training set were the most sensitive ones for predictions, we then added another round of imputation for the last 3 month values only, that was based not only on the target variables but also on the external variables we had indeed during preprocessing. This second round enabled us to get better results for prediction.

Model Architecture and Training

As the problem we addressed was a multivariate time series problem with external data, many "classical" pure time-series models were not applicable to our case. Given the tight time constraint, we focused on 3 main types of models: time-series AutoML, classical ML models, and deep learning models for time series. At every stage, we assessed the models' performances through time-series cross-validation and time-series compatible train-test splits.

AutoML model: AutoGluon-TimeSeries. [6]
AutoGluon is an open-source AutoML framework designed by Amazon that relies on the fact that it is often more effective and less costly to try several models and ensemble them rather than relying on hyperparameters optimization of one model. In our use case, AutoGluon-TimeSeries was attractive because it enabled to quickly test different types of models: baselines (naive seasonal model), time-series statistical models leaving aside external features (AutoETS, AutoARIMA), tabular models (simple and recursive), deep learning models (DeepAR, Temporal Fusion Transformer), and a pre-trained model with zero-shot forecasting (Chronos). The automatic cross-validation and ability to retrieve the results from each model were also appreciable.
However, the results were somewhat disappointing, with a final MAE of 4.7. This may be due to not optimal implementation on our side or to the inherent limits of trying to fit so many models in less than 30mn - which is the limit we chose.

Classical ML models: XGBoost, Catboost, Random Forest regressors
We tested four types of classical Machine Learning regressors with some fine tuning for time series and multi-output predictions: XGBoost, Catboost, stacked XGBoost and Catboost, and Random Forest. Among these, our best model was an XGBoost with lags on external features.

XGBoost is one of the most famous Gradient Boosting algorithm models, well-known for its speed and accuracy, often well-performing in Kaggle challenges. We analyzed the MSE, the MAE and the R^2 score when looking at our validation set's results, even though we chose the MAE loss function so as to align it with the final scoring metric used. It turned out that the model which delivered the best results was a classical XGBoost Regressor, completed with a lot of external data for which we had implemented 3 lags (1, 2, and 3 hours before the prediction). The chosen features thus included all preprocessed and external features, as well as their lags, but not the target lags. We performed a grid search to optimize our model hyperparameters:

- *Number of estimators*: 500
- *Learning rate*: 0.1
- *Maximum depth*: 10

This implementation gave us the best results , with a MAE of 3.27 in the public leaderboard. We would have liked to perform an in-depth feature selection, and to implement a more advanced recursive model from this regressor, but we did not have time for it.

Deep Learning models: LSTM & TFT

Deep learning models were immediately attractive for our problem due to their ability to handle multivariate targets, to incorporate external data, and to learn complex temporal patterns without the need to explicitly encode them. However, their implementation was much more challenging. We tried 2 main models: a LSTM (Long Short-Term Memory model) and a TFT (Temporal Fusion Transformer), both implemented with the PyTorch library. Contrary to the "tabular ML" approach, we did not encode lag features for these models as they are supposed to understand them implicitly.

1. LSTM

We designed a simple LSTM neural network [7], designed to handle time series data. The model architecture is described below:

○ *Input Layer*
  Input: external features, combined with trigonometric ones. Numeric features are scaled.
  Sequence length: 24h (one day), meaning the model uses the previous 24 time steps to predict at the current step
  Input size: combined length of all features which are concatenated as input

○ *LSTM Layer:* a single LSTM layer with a hidden state size of 10 units

○ *Fully Connected (FC) Layer*: the last hidden state is passed through a fully connected (FC) layer to output predictions

Hyperparameters:

- *Sequence Length*: 24
- *Hidden Size* : 10
- *Batch Size*: 64
- *Learning Rate*: 0.001
- *Weight Decay*: One try with 0, another one with 1e-5

Training:

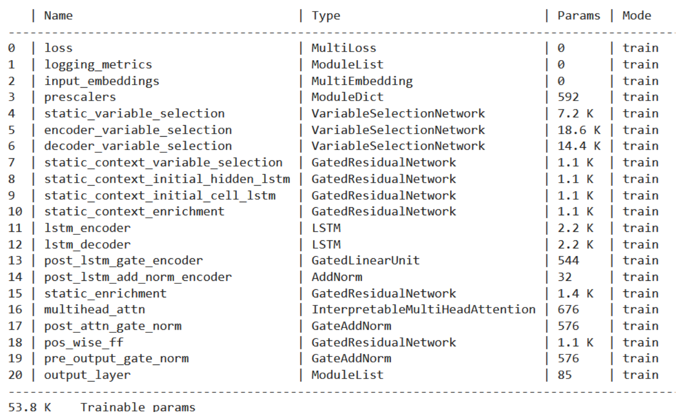
- *Loss Function*: MSE
- *Optimizer*: Adam
- *Regularization*: Weight decay
- *Time*: 3mn (20 epochs)

2. TFT

Our second deep-learning model is a Temporal Fusion Transformer [8], which we thought was promising due to its ability to learn very complex temporal patterns. Due to its complexity, we did not build the model from scratch but rather reused the pre-built TemporalFusionTransformer class from pytorch.forecasting. Due to build and train times, we were only able to test one implementation, for which we chose a relatively compact implementation and simple hyperparameters:

- *Learning rate*: 0.03 - relatively high to assess the model quickly
- *Hidden size*: 16 - low to start simple
- *Attention head size*: 4 - standard to balance expressiveness and complexity
- *Dropout rate*: 0.1 - relatively low since the model is relatively small
- *Hidden continuous size*: 8 - low to start relatively simply again
- *Output size*: [1, 1, 1, 1, 1] - for a multi-output problem
- *Batch size*: 64
- *Max encoder length*: 168 = 7 x 24 - start simple with a week
- *Max prediction length:* 504 - corresponding to the data to predict

This TFT allowed a built-in distinction between variables known in advance (weather, time features) and unknown variables that were predicted on the go, such as the car traffic.



```
   | Name                               | Type                            | Params | Mode
--------------------------------------------------------------------------------------------
0  | loss                               | MultiLoss                       | 0      | train
1  | logging_metrics                    | ModuleList                      | 0      | train
2  | input_embeddings                   | MultiEmbedding                  | 0      | train
3  | prescalers                         | ModuleDict                      | 592    | train
4  | static_variable_selection          | VariableSelectionNetwork        | 7.2 K  | train
5  | encoder_variable_selection         | VariableSelectionNetwork        | 18.6 K | train
6  | decoder_variable_selection         | VariableSelectionNetwork        | 14.4 K | train
7  | static_context_variable_selection  | GatedResidualNetwork            | 1.1 K  | train
8  | static_context_initial_hidden_lstm | GatedResidualNetwork            | 1.1 K  | train
9  | static_context_initial_cell_lstm   | GatedResidualNetwork            | 1.1 K  | train
10 | static_context_enrichment          | GatedResidualNetwork            | 1.1 K  | train
11 | lstm_encoder                       | LSTM                            | 2.2 K  | train
12 | lstm_decoder                       | LSTM                            | 2.2 K  | train
13 | post_lstm_gate_encoder             | GatedLinearUnit                 | 544    | train
14 | post_lstm_add_norm_encoder         | AddNorm                         | 32     | train
15 | static_enrichment                  | GatedResidualNetwork            | 1.4 K  | train
16 | multihead_attn                     | InterpretableMultiHeadAttention | 676    | train
17 | post_attn_gate_norm                | GateAddNorm                     | 576    | train
18 | pos_wise_ff                        | GatedResidualNetwork            | 1.1 K  | train
19 | pre_output_gate_norm               | GateAddNorm                     | 576    | train
20 | output_layer                       | ModuleList                      | 85     | train
--------------------------------------------------------------------------------------------
53.8 K    Trainable params
```

Fig. 2. List of trainable parameters of the TFT model

Training:

- *Loss used:* MAE - aligned with the evaluation metric
- *Optimizer*: Adam - default choice
- *Process*: in 2 parts - (i) with a train and validation set, to make sure the model was working (ii) with the full dataset
- *Time*: 8 epochs in total, ~1h with a A100 GPU available on Google Colab

This model scored 3.8 MAE, which is very encouraging for a first trial and we would have liked to have more time to train longer and improve our implementation and its parameters.

Results and Comparison

The following table summarizes our comparison of the different models, both in terms of performance and usage.

	MAE	Training Time	Strengths	Weaknesses
Autogluon	4.76	20mn	Automatic, ability to test several models at once	Lack of tweakability and readibility, data preformatting is not straightforward
Random Forest	4.05	~40s	Good performance with short training time, familiar framework (scikit-learn)	Requires feature engineering and tuning to achieve optimal results
XGBoost with lags	3.27	1mn	Familiar framework, short training time allows to test a lot of different things, best performing model on this challenge	Low interpretability, difficult to integrate lags on the target (on the prediction in the test set)
LSTM	4.16	3mn	Ability to capture long temporal patterns while keeping the training lighter than more complex models such as TFT	Introduces several new hyperparameters whose individual impact on the final performance is difficult to define, not interpretable by nature
TFT	3.81	1h	Potentially strongest model, ability to capture deep temporal patterns without the need of explicit encoding	Complexity to implement, need to ramp up with pytorch.forecasting framework, extensive training time

Our results indicate that tabular ensemble models performed best, which is likely to be at least partly due to our familiarity with them, which enabled us to fully leverage their capabilities. With more time however, we believe that exploring more complex architectures like the TFT could have yielded very interesting results.

Discussion

We learned a lot at every step of the project, trying to overcome every new challenge we faced.

Our first challenge was gathering external data, cleansing it and making it usable in our models. We particularly struggled with traffic data, dealing with millions of observations and thousands of counters that were not always simultaneously operational. Ultimately, we succeeded in processing this data effectively, resulting in a coherent traffic dataset with minimal missing values across the entire period, that we later imputed.

While we had already dealt with temporal data, we gained a lot of insights as to how to deal with them specifically. This includes how to create lags when using traditional ML models, or on the contrary why it was not necessarily useful to include them when applying deep learning to time series, for the LSTM and TFT models. We were especially interested to learn that the attention mechanism of the transformer could naturally handle these temporal dependencies.

Finally, we aimed to build a TFT model to improve our MAE, as a TFT appeared to be one of the most suitable frameworks for our use case. However, during training, we realized that it required significantly more time and computational power than anticipated, with each epoch taking 8 minutes to complete. Had we had more resources, we would have liked to dive deeper into this model and improve it and its parameters.

Conclusion

In conclusion, rigorous data preprocessing, the integration of external factors such as weather and traffic, and the exploration of various ML frameworks – including AutoML, classical models like XGBoost, and deep learning techniques such as LSTM and TFT – enabled us to precisely model the targets and to secure 1st place in this challenge.

With additional time and computational power, we would have liked to further refine our deep learning models to achieve even better results.

Annexes

References

[1] See for example on Medium.com here

[2] Retrieved on Data Gouv

[3] Meteo France hourly records, retrieved here

[4] Comptage routier - hourly observations:

- 2020-2023 data retrieved here
- 2024 data retrieved here

[5] Qolmat library: main page

[6] Shchur, O., Turkmen, C., Erickson, N., Shen, H., Shirkov, A., Hu, T., & Wang, Y. (2023). AutoGluon-TimeSeries: AutoML for probabilistic time series forecasting. In *Proceedings of the International Conference on Automated Machine Learning.* Library link

[7] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.

[8] Lim, B., Arík, S. Ö., Loeff, N., & Pfister, T. (2021). Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting, 37*(4), 1748-1764.