Aircraft Numerical "Twin": A Time Series Regression Competition

Adrien Pavao, Isabelle Guyon, Nachar Stéphane, Fabrice Lebeau, Martin Ghienne, Ludovic Platon, Tristan Barbagelata, Pierre Escamilla, Sana Mzali, Meng Liao, et al.

Aircraft Numerical “Twin”: A Time Series Regression Competition

Adrien Pavão*, Isabelle Guyon*, Nachar Stéphane†, Fabrice Lebeau†, Martin Ghienne‡, Ludovic Platon‡, Tristan Barbagelata‡, Pierre Escamilla‡, Sana Mzali‡, Meng Liao‡, Sylvain Lassonde‡, Antonin Braun‡, Slim Ben Amor§, Liliana Cucu-Grosjean§, Marwan Wehaiba§, Avner Bar-Hen§, Adriana Gogonel§, Alaeddine Ben Cheikh¶, Marc Duda¶, Julien Laugel¶, Mathieu Marauri¶, Mhamed Souissi¶, Théo Lecerf‖, Mehdi Elion‖, Sonia Tabti‖, Julien Budynek‖, Pauline Le Bouteiller‖, Antonin Penon**, Raphaël-David Lasseri**, Julien Ripoche**, Thomas Epalle**

**Université Paris-Saclay, †Dassault-Aviation, ‡Aquila/Supméca, §Statinf, ¶MFGLabs, ‖Fieldbox AI, **Magic LEMP*

***Abstract*—This paper presents the design and analysis of a data science competition on a problem of time series regression from aeronautics data. For the purpose of performing predictive maintenance, aviation companies seek to create aircraft “numerical twins”, which are programs capable of accurately predicting strains at strategic positions in various body parts of the aircraft. Given a number of input parameters (sensor data) recorded in sequence during the flight, the competition participants had to predict output values (gauges), also recorded sequentially during test flights, but not recorded during regular flights. The competition data included hundreds of complete flights. It was a code submission competition with complete blind testing of algorithms. The results indicate that such a problem can be effectively solved with gradient boosted trees, after preprocessing and feature engineering. Deep learning methods did not prove as efficient.**

***Index Terms*—aeronautics, time series regression, machine learning, deep learning, gradient boosted trees**

I. Introduction

The Paris Region AI Challenge 2020, organized by the Ile-de-France region, Dassault Aviation, and Université Paris-Saclay, was an industry contest between 10 startup companies, taking place from February 1 to April 9, 2021. The teams had to tackle a multivariate time-series regression task, involving aeronautics data from test flights of Dassault Aviation aircraft. Given a number of input parameters (sensor data) recorded in sequence during the flight, the participants had to predict output values (strain gauges), also recorded sequentially on test aircraft, but not recorded on service aircraft. In the application domain considered, collecting strain gauge data on service aircraft is costly or infeasible. The objective is to create a predictive model of the strain gauges measurements from the data of test aircraft, to obtain “virtual” strain gauges for service aircraft, hence the concept of “Aircraft Numerical Twin”. The main goals are to optimize scheduled maintenance and to improve the design of future aircraft.

Besides delivering solutions to this problem, the challenge addressed questions of interest in machine learning:

1) **Hand-crafted feature engineering.** Is the design and extraction of features a key factor of success?
2) **Explainability**. Do participants’ solutions provide explainable decisions or interpretable models?
3) **Asynchronous time series.** Do the participants come up with effective or original means of handling different sampling rates and missing values?
4) **Model design.** Does the challenge reveal that one type of machine learning model is superior to another, for the task of the challenge?
5) **Generic workflow.** Does a high level modular organization of solutions emerge?
6) **Joint model and HP selection.** Are models and their hyper-parameters selected automatically?
7) **Transfer/Meta learning.** Are solutions provided generic and applicable to new domains?
8) **Hardware constraints.** Are computational/memory limitations observed?
9) **Diversity and creativity.** Do participants’ solutions include novel complementary ideas, offering opportunities of synergistic combinations?
10) **Intrinsic difficulty and modeling difficulty.** Were the tasks of the challenge of a difficulty adapted to push the state-of-the-art in the domain considered?

This paper answers those questions, presents the statistical analysis of the best submission of each candidate, and summarizes the highlights of winning solutions.

II. Problem setting

Time series regression addresses the problem of **predicting certain unavailable *output* time series from other given *input* time series** (Figure 1). Non sequential meta-data may also be available. This task differs from time series forecasting in two aspects: (1) past, present, and *future* values from the input time series are available to predict the output time series; (2) the prediction error

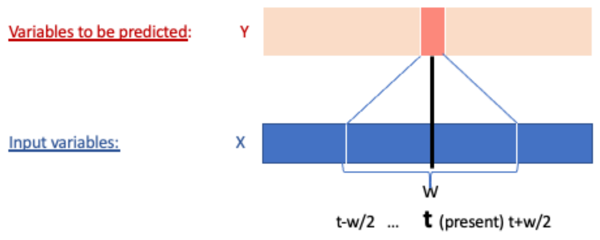


Fig. 1: **Setting of Paris Region AI Challenge 2020: Time series regression.** Input variables are obtained from "sensors" and output variables are obtained from "gauges". A typical method is to regress a window of inputs centered at time t to predict $\mathbf{y}_t$.

is computed globally for the entire output time series, no on-line feedback is provided on past errors.

In our application domain, input time series (data routinely recorded) and output time series (strain gauge data recorded on test aircraft only) are multi-variate time series, **irregularly sampled** or sampled at different frequencies, but with given time stamps. We refer to this problem as "asynchronous time series" regression. The problem is cast into a supervised learning problem. For training purposes, target data are recorded during test flights (but absent during regular flights).

Formally, both input $X = [x_{t,j}], t = 0 \cdots t_{max}, j = 1 \cdots n_{sensors}$ and output $Y = [y_{t,j}], t = 0 \cdots t_{max}, j = 1 \cdots n_{gauges}$ are multivariate time series. The index t indicates time and the index j the various sensors (called sensors for the inputs and gauges for the outputs). At training time, both X and Y are available. At test time, only X is available and Y must be predicted. A typical way to proceed is to learn to regress a window of input data centered at time t to the output at time t.

In the native data format, a dataset $\mathcal{D} = \{(t_i, z_i)_{j,k}\}$ contains time series $(t_i, z_i)_{j,k} \in \mathcal{T}$, with $j \in [\![1, n_{gauges} + n_{sensors}]\!]$, indicating a measured parameter, and $k \in [\![1, n_{flights}]\!]$, indicating a flight. Here t_i indicates a time stamp and z_i a measurement.

Furthermore, the series can be divided into two groups, $\mathcal{T}^{n_{sensors}}$, and $\mathcal{T}^{n_{gauges}}$, representing respectively the inputs and the outputs. The goal is to predict missing output values in the test time series, given the time stamps (recall that time series are asynchronous).

III. Challenge design

A. Protocol of evaluation and metric

Ten participants (startup companies, SME and academics) were pre-selected upon submitting an application file. The challenge was organized with **code submission** in two phases (feedback phase and final test phase) on the Codalab platform. Both training and testing of the models was done on the platform in both phases, hence the participant had **no direct access to the data**, except for some sample data (not part of the challenge) provided with the starting kit, for illustrative purposes. **Pre-training off-platform was not allowed**, except for hyper-parameter adjustment. This was enforced by checking the submission sizes and later the code itself.

The objective of the **feedback phase** was to let participants develop their model. To that end, the participants could make up to 100 submissions over the 2-month duration of the feedback phase. Each submission was run on the Codalab platform, using a **dedicated compute worker**, with a generous execution time limit of *five days*. One compute worker included 1 GPU NVIDIA RTX 2080Ti, 4 vCPUs and 16 GB DDR4 RAM. The dataset was stored on an SSD shared by all 10 compute workers. The code submitted by the participants was executed inside a docker container, with pre-installed machine learning libraries including Tensorflow, Pytorch, and scikit-learn. The code of the participants had **no internet access**. The evaluation was carried out in two steps, in two separate containers: (1) Training the predictive model and testing it on unlabeled test sequences; (2) Scoring the predictions using the ground truth of the labels. Hence, **the code of the participants was never exposed to the ground truth**. Upon termination of the execution of their code, the participants received feedback on their performance on a leader board on the Codalab platform. The participants had access to minimal information via logs to debug their code; they could not download the predictions made by their code.

The objective of the **final test phase** was to evaluate the code submitted by each participant on fresh data, not used to develop their solution. Only **one final submission per participant** was allowed. The tests were also carried out on the Codalab platform, using the same hardware as in the feedback phase.

The predictions were evaluated using the Mean Absolute Error (MAE), defined as: $MAE = \frac{1}{n}\Sigma_{i=1}^{n}|y_i - \hat{y}_i|$ where $\mathbf{y}$ is a ground truth output variable and $\hat{\mathbf{y}}$ a prediction. Performances were averaged over all data points in all test sequences and over all gauge outputs.

B. Data

The input time series included between 124 and 130 input parameters (a.k.a. features or sensor data), depending on the fights (some inputs were missing in some flights). The output time series included between 62 and 67 output parameters (strain gauges) to be predicted.

During the **feedback phase**, the code of a participant was trained on 117 complete flight records and tested on 4805 flight sequences from 186 flight rectords. During the **final test**, it was trained on the same data, but tested on 7398 *other* test flight sequences from the same flights used for testing in the feedback phase.

Altogether, there were $\simeq 70GB$ of data. The volume of data presented a major difficulty to the code of the

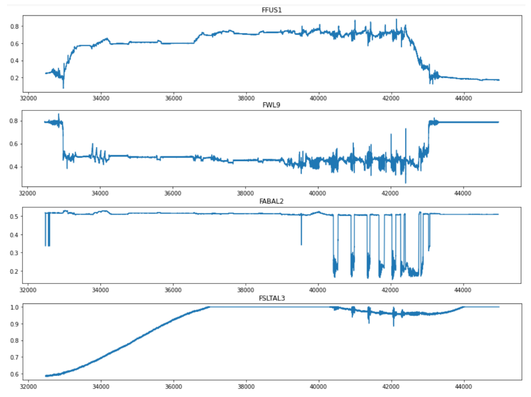


Fig. 2: **Sample data:** Four illustrative input time series.

participants. It was not possible to load all the data at once in live memory (only 16 GB RAM available).

TABLE I: **Dataset statistics.** "#" means "number of".

# train flights	# test flights	# feedback sequences	# final sequences
117	186	4805	7398
# input	**# output**	**mean variable length**	**mean sequence length**
124 - 130	62 - 67	856450	49

C. Starting kit

Since the participants' code was blind tested on the Codalab platform, the organizers provided them with a well documented interface to prepare their submission. In addition, they obtained a "starting kit", including sample code, sample data, and the evaluation code.

The code interface imposed that the participants would use a **Python object "wrapper"**[1] including two methods: $\texttt{fit}(\mathcal{D}_{train})$ and $\texttt{predict}(X)$. The evaluation code enforced constraints imposed by the rules of the challenge, such as the impossibility to train on test set sequences. A so-called **ingestion program** was in charge of receiving the submissions of the participants, calling $\texttt{fit}(\mathcal{D}_{train})$ to train the model, and then repeatedly calling $\texttt{predict}(X)$ on all test sequences, randomly ordered. The ingestion program had no access to the ground truth Y of the test sequences. The predictions thus made were then scored by a **scoring program**, run in a separate container, having access to the ground truth. Both the ingestion program and the scoring program were provided to the participants, so they could run them on their local computers.

One additional difficulty of the challenge is that the series are asynchronous and irregularly sampled. To facilitate the task of the participants, the starting kit included a **data reader** converting asynchronous data to regularly sampled series. However, the participants were expected to turn in predictions at *given time stamps* in the output series, hence some interpolation needed to be done as post-processing. Alternatively, the participants could use the native format and address directly the problem of asynchronous time series.

[1]The python object could call libraries not written in Python.

The starting kit included also sample of data, called **public data** featuring 3 complete flights. Each flight represented $\simeq$ 1 Gb of data.

IV. Algorithms of Time Series Regression

We briefly describe methods that were essential building blocks of the participants' solutions.

A. Linear regression

In its simplest formulation, a linear model for time series regression predicts the outputs at time t from the inputs at time t:

$$\hat{y}_t^i = \sum_j w_{i,j} x_t^j \ , \tag{1}$$

or, in matrix-vector notation: $\mathbf{\hat{y}_t} = W\mathbf{x_t}$, where $\mathbf{x_t}$ is a vector of dimension $n_{sensors}$, $\mathbf{\hat{y}_t}$ is a vector of dimension n_{gauges}, and W is a weight matrix of dimension $n_{gauges} \times n_{sensors}$. A bias value can be included by adding a constant input equal to 1. The model can be trained using ordinary least squares [1].

A variant of this model uses as input a feature vector, which may include, in addition to present values of x_t^j, "lag" values x_{t-k}^j and "lead" values x_{t+k}^j. Such models are **non causal filters**, since they use both the past and future to predict the present. They perform a **convolution**. Possibly, other more complex features may be extracted from the input sequence x_t^j, $j = 1 \cdots n_{sensors}$, $t = 0 \cdots t_{max}$, *e.g.* products of original features.

Such linear models bear resemblance with ARIMAX models [2] frequently used in time series forecasting. However, ARIMAX models use past values of x_k^j, **and past values** of y_k^i, $k = 0 \cdots t-1$, to predict y_t^l. In contrast, the time series regression linear models **do not use past values** of y_k^i, **but may use future values** of x_k^j.

B. Ensemble of decision trees

Ensembles of decision trees are successful for classification and regression from feature-based data (*aka* "tabular data") [3]. Recently, they have also proven efficient for time series regression [4], with adequate **feature engineering** (*e.g.* including lag and lead features).

Ensemble methods combine the predictions of several "weak learners" in order to produce more accurate predictions [5]. Ensemble of decision trees use decision trees as weak learners [6]. A decision tree recursively partitions the input space along the axes (features). During training, starting from a root node including all the training examples, the data are split according to a "decision rule" into two subsets, by setting a threshold on one of the features, thus creating two new nodes. The process is iterated until a halting condition is reached or each node contains a single example. The terminal nodes are called leaves. At test time, an example finds its way through the tree by applying the decision rules at each node of the tree. When a leaf is reached, the predictions are made according to the average of training example output values, in the case of regression, or the most frequent label in the case of classification. The choice of features and decision rules vary depending on the algorithms. Typically, one minimizes the sample heterogeneity within nodes.

Random Forests (RFs) [7] create ensembles of decision trees based on the principle of bagging (**b**ootstrap **agg**regat**ing**). Bagging consists in resampling the training set (with replacement) to create many variants of the training set, of the same size [5]. Thus some examples will be drawn multiple times. One new weak learner (here, decision tree) is then trained on each variant of the training set. Predictions are made by averaging the results of weak learners. As an additional refinement, the RF algorithm also sub-samples the features when training a weak learner, to handle large feature spaces.

Gradient Boosted Trees (GBT) [8] create *sequential* ensembles of decision trees (as opposed to *parallel* ensembles like RF). Briefly, let $\mathcal{H}$ be the set of the weak learners (decision trees), and $\{h_i\}_{i\in[\![0,B]\!]}$ a subset of $\mathcal{H}$. The GBT predictor is a linear combination of weak learners ($\gamma_b \in \mathbb{R}$):

$$f_B(\mathbf{x}) = \sum_{b=0}^{B} \gamma_b h_b(\mathbf{x}) \tag{2}$$

Given partial sums $(f_b)_{b\in[\![0,B]\!]}$, a loss function $\ell(y,\ \hat{y})$, and training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, f_B is built sequentially:

$$\begin{aligned} f_0(\mathbf{x}) &= \arg\min_{\gamma\in\mathbb{R}} \sum_{i=1}^{n} \ell(y_i,\ \gamma) \text{ (constant model)} \\ f_b(\mathbf{x}) &= f_{b-1}(\mathbf{x}) + \gamma_b h_b(\mathbf{x}) \\ \text{with } \gamma_b &= \arg\min_{\gamma\in\mathbb{R}} \sum_{i=1}^{n} \ell(y_i,\ f_{b-1}(\mathbf{x}_i) + \gamma h_b(\mathbf{x}_i)) \end{aligned} \tag{3}$$

and $h_b(\mathbf{x})$ is a weak learner fitting the pseudo-residuals of f_{b-1}, trained with $\{(\mathbf{x}_i,\ \nabla_{f_{b-1}}\ell(y_i,\ f_{b-1}(\mathbf{x}_i))\}_{i=1}^{n}$.

C. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are deep learning methods, successful notably in image recognition [9] and speech processing [10]. "Neurons" are linear models, followed by an "activation function" performing a "soft" decision. CNNs are several stacked layers of neurons with local connections and shared weights, which act as "convolution kernels" or feature extractors. CNNs are usually trained with stochastic gradient descent [11].

Figure 3 shows a recent version of a CNN for time-dependent inputs, called Time Convolutional Network (TCN) [12], applicable to time series regression. TCNs

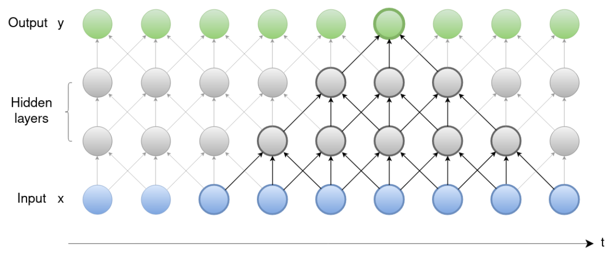


Fig. 3: **Diagram of a basic TCN**. Each circle represents a neuron and edges represent time lags. A kernel is a neuron having local connections and being repeated over the entire time series, to compute the next layer. Inputs may be multivariate and each layer may include multiple kernels. The outputs of "causal" TCNs receive only information from "lag" (past) inputs, while those of "non causal" TCNs can also get information from present and "lead" (future) inputs. Circled units belong to the field of view of the circled output neuron.

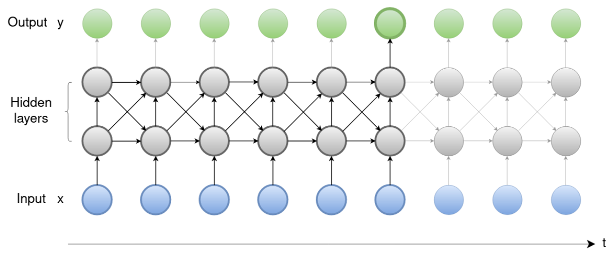


Fig. 4: **Diagram of a RNN**. Each cell uses the previous output as a new input. A cell is a differentiable module *e.g.* an LSTM cell [13]. Circled units belong to the field of view of the circled output neuron. Compared to (non-causal) TCNs, RNNs do not see future inputs.

are 1D convolutional layers stacks, preserving series length by zero-padding. TCNs for time series forecasting use "causal convolutions" *i.e.* the output at time *t* receives only *past* information. In contrast, TCNs for time series regression allow the output to receive past and *future* information. In basic CNN designs, the output gets a large field of view only if the network is very deep or the kernels very large. TCNs alleviate this problem with "dilated convolutions", "inflating" the kernel by inserting holes between elements. One last refinement consists in using "residual" connections, or "skip" connections, which permits training deep nets without overfitting.

D. RNNs

Recurrent Neural Networks (RNNs) differ from feed-forward neural networks by their use of feedback connections. This is represented in Figure 4 by horizontal connections between hidden units: a unit gets, at the next time step, inputs from itself. RNNs have been used for time series regression or classification problems with long-term time dependencies [14]. However, in their vanilla version, they do not incorporate lead features, which could be important in time series regression.

RNNs are typically trained with stochastic gradient descent [11], using "backpropagation through time",

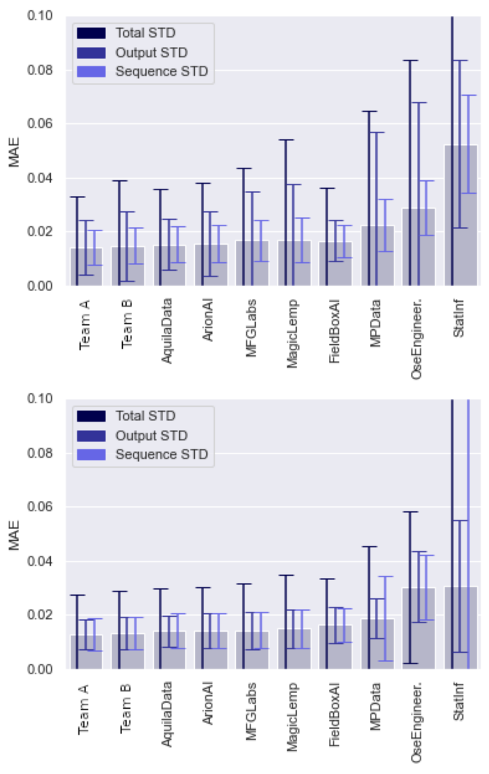


Fig. 5: **Mean of MAE** on all sequences and all output variables. Top: feedback phase. Bottom: final phase. Standard deviations shown are: **Total STD** = std computed on all scores (all pairs {output, sequence}); **Output STD** = average scores grouped by output variables; **Sequence STD** = average scores grouped by flight sequences. As observed in Figure 6, it seems that the tasks of the final phase were slightly easier.

which boils down to unfolding the network in time, up to a certain depth. To enhance the long-term memory, an RNN can be built from cells incorporating time delays or feedback loops, *e.g.* Long Short-Term Memory cells (LSTM) [13] and Gated Recurrent Units (GRU) [15]. Such architectures alleviate the problem of vanishing gradients encountered when training traditional RNNs. They can handle lags of unknown duration and are relatively insensitive to gap length, an advantage over other RNNs, hidden Markov models, and TCNs.

V. Challenge results

A. Synopsis of the results

The results are presented in Table III, in order of performance in the final phase, best comes first. The participants tested three types of methods:

- **Linear models** inspired by ARIMAX models [2].
- **Deep-learning** feed-forward CNNs, regular MLPs, and RNNs (typically LSTM).
- **Gradient Boosted Tree** (GBT) regressors (CatBoost, XGBoost, LightGBM), incorporating the time component via feature engineering.

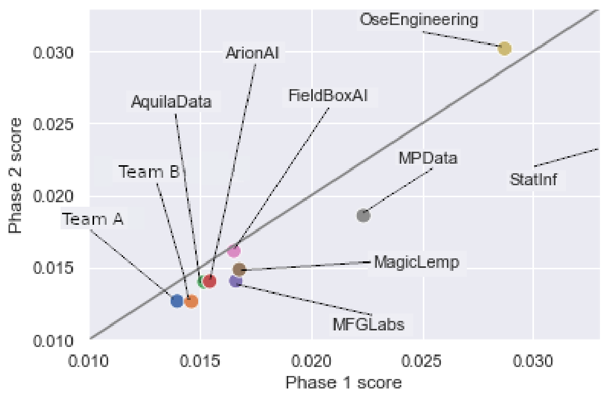


Fig. 6: **Comparison of the scores in feedback phase (phase 1) and final phase (phase 2).** Final phase scores are better, indicating that the outputs were more predictable and that participants did not overfit.

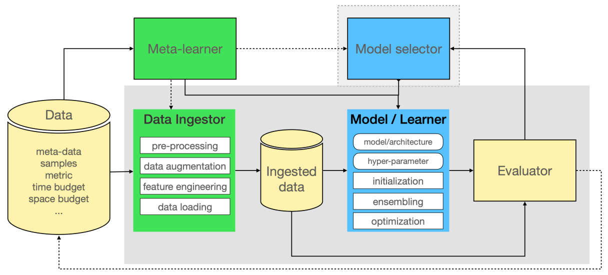


Fig. 7: **Generic workflow shared by most participating teams (inspired by [16]).**

We indicate in boldface in Table III the preferred method of the participants. The top 6 teams all used GBT methods in their final solution. If we assumed that teams were selecting at random GBT *vs.* other methods, there would be $1/2^6 = 1.5\%$ chance to get all 6 first teams to select GBT methods. Therefore this result is significant.

As it can be seen in Table III and Figure 5, the first seven teams got a similar performance, impossible to significantly distinguish. The final winner was selected by a jury, taking into account scientific methodology and future plans described by each team in a report. The report highlights are presented in supplemental material.

Figure 6 compares the performances of the participants in the feedback phase and the final phase. The data of the final phase seems to have been slightly easier. There is no sign of overfitting the feedback phase.

Table II answers the 10 challenge questions. Further details are provided below.

B. Generic workflow

The participants all used a similar generic workflow, as described in Figure 7. We briefly summarize the most salient features of the solutions of the participants. More details are provided in supplemental material.

TABLE II: **Answers to the 10 challenge questions.** Green checkmarks indicate positive answers, orange checkmarks partial answers, and red crosses negative answer.

Question	Answer	Comment
Q1 Good hand-crafted features?	✔	Top ranking methods based on gradient boosted trees relied heavily on feature engineering. Several participants used domain knowledge to engineer features.
Q2 Good explainability?	✔	Limited to feature selection and interpretation of human hand-crafted features.
Q3 Asynchronous time series dealt with?	✔	Limited to resampling to synchronize, then interpolating.
Q4 Conclusive results?	✔	All six top ranking participants used gradient boosted tree methods. Such methods seem therefore best suited to time series regression, as confirmed in [4].
Q5 Generic workflow?	✔	A generic workflow similar to the starting kit was adopted by all participants.
Q6 HP selection	✔	Most participants performed random search or grid search, using cross-validation ignoring sample time ordering or using hold-out strategies at the flight sequence level. Only two participants performed Bayesian optimization.
Q7 Transfer learning or meta-learning?	✔	The solutions provided are fairly generic and should be applicable to new time-series regression problem, except some aspects of feature engineering.
Q8 Hardware constraints observed?	✔	All solutions ran in less than 1 day (way below the time budget of 5 days). The submission of only one participant crashed, possibly due to memory limitations.
Q9 Creativity?	✔	The participants contributed novel solutions that are complementary, with a variety of types of features and models (*e.g.* based on gradient boosted trees or neural networks). This offers opportunities for creating ensembles. Little "physics"or knowledge about the task went into the model design.
Q10 Difficulty?	✘	The task difficulty was within reach of the participants, but the performances of the participants were very close, thus the methods could not be well differentiated.

TABLE III: **MAE performances in both phases.** The order is given by final phase scores, best is first. The score differences between participants being not statistically significant, the ranking is based upon digits of lesser significance, indicated as subscripts. We indicate also the approaches used by each team and the duration of execution of their final entry, which is the duration of training and predictions, and is therefore similar between the two phases. All runs are under 1 day of calculations, way under the time budget limitation of 5 days.

Team	Score phase 1	Score phase 2	Main methods	Duration (seconds)	Duration (hours)
Team A	0.01_{399}	0.01_{269}	**CatBoost**, multiple regression	83045	23.07
Team B	0.01_{462}	0.01_{274}	**XGBoost**	70031	19.45
Aquila-Supméca	0.01_{519}	0.01_{402}	**LightGBM**, RF, Ridge, MLP	9551	2.65
Arion AI	0.01_{545}	0.01_{406}	**XGBoost**, regular regression	30243	8.40
MFG Labs	0.01_{662}	0.01_{408}	**CatBoost**, Tabnet (attention), TCN	58699	16.30
Magic LEMP	0.01_{678}	0.01_{488}	Cascade **XGBoost**, LSTM, CNN	64392	17.89
FieldBox.ai	0.01_{653}	0.01_{616}	**MLP** CNN	12335	3.43
MP Data	0.02_{236}	0.01_{861}	CNN, GNN, attention, LSTM	59312	16.47
Ose Engineering	0.02_{871}	0.03_{019}	CNN, LSTM	21250	5.90
Statinf	0.05_{243}	0.03_{053}	**MLP**, LSTM, RF	38119	10.59

The bulk of the work of the top ranking participants went into data loading and feature engineering. The data loader was critical since the entire training set could not fit in RAM. Several approaches were taken including: subsampling at a lower rate (the default was 10 Hz, typically it was reduced to 3 Hz); reducing the resolution of coefficients (from 64bit floating point to 32bit integers); or compressing data via a wavelet transform.

Three types of features were considered: ad hoc features incorporating human knowledge about the task, generic combinations of original features, and signal processing features (derivative, FFT, wavelets). Lag and sometimes lead features were also considered, but did not seem to be of much help, according to systematic experiments conducted and reported by the participants. All in all, it seems to be possible to treat this problem as a regular regression problem, largely ignoring the time component.

Concerning the regressor, as previously mentioned, the most successful participants used gradient boosted trees (GBT). Some participants compared linear classifiers, GBT, and neural networks, and concluded in favor of GBT. The fact that lag and lead features did not help much may explain why this problem could be solved by classical regressors such as GBT. Several implementations of GBT were tried and compared for speed and accuracy, and seem to perform relatively similarly. See Table IV for a summary of the main features of methods. It is worth noticing that the teams using GBT spent quite some effort on feature engineering while those using neural networks relied on their model to learn the features.

The time budget may have been a factor influencing the choice of algorithm. Although the total time budget per submission was 5 days, the duration of the feedback phase was only 2 months. Some participants may have preferred obtaining frequent feedback rather than letting models run for several days. However, the participants could train their models on their own computers to perform model selection and in the end no submission made on the platform exceeded one day. Hence, we do not see computational time as a strong limiting factor.

TABLE IV: **Summary of approaches taken.** sr=sampling rate (default=10 Hz); strat = stratified sampling (by phase or maneuver); uni = uniform sampling; interpol = interpolate bet. samples; md = replacement of missing data (md-const = repl. by a constant value; md-clust = repl. by median of same cluster or nearest in cluster, md-fill = back-fill, front-fill, or nearest-fill; md-m = repl. by mean of that sensor); lag or lead = use lag or lead features; tenc = target encoding; norm = feature normalization, standardization, or drift removal; clean = data cleaning; fc = feature construction (fc-adhoc = ad hoc feature construction; fc-combi = multiply, add or square features; fc-fft = add frequency features; fc-der = derivative computed; fc-ma = moving average; fc-trigo = apply trigonometric transforms; fc-stat = flight-level summary statistics; fc-clust: apply clustering or unsupervised learning); fs = feature selection or reduction of the number of sensors; fn = feature number; recast = reduce numerical resolution (e.g. 32 bits); clip = clipping or spike removal; val = use a validation set or cross-validation, eliminating time ordering of samples; ho = hold-out: train on some flights, test on others; lb = leaderboard feedback; grid = grid-search; bo = Bayesian opt.; early = early stopping; omo = one model per output; me = multiple experts; ens = ensembling; build multiple models with data fitting in memory, then average.

Team	Data loading and sampling	Preprocessing	Feature engineering; feature selection	HPO and Ensembling
Team A	strat	md-const, lag+lead=520, tenc	fc-ma=520, fc-adhoc=50, fc-stat=231	ho, early, omo, me=5
Team B	uni	md-m, md-clust, norm, lag=2	fc-adhoc, fc-der, fs, fn=221	val, lb, early, omo
Aquila-Supméca	uni, sr = 0.5 Hz	md-fill (back-fill), clean, fc-clust	fs (merged redundant sensors)	boosting, stacking, me
Arion AI	sr = 3 Hz	clean, recast	fs(110)	val, bo, lb, omo, ens
MFG Labs	sr = 10 Hz	wavelets, md-fill, norm, lag, clip	fs, fc-adhoc+combi+der+trigo+stat	lb, me=9
Magic LEMP	sr = 3 Hz	md-clust	fc-adhoc(200), fc-fft	bo, omo, boosting
FieldBox.ai	sr = 10 Hz	clean, wavelets	fc-combi, fc-trigo, fs	NA
MP Data	strat	md-clust, md-fill	NA	NA
Ose Engineering	interpol, sr = 10 Hz, strat	md-fill+const, clip, clean, norm	fs	grid
Statinf	interpol, sr = 10 Hz	md-fill+const	fs, fc-der, fc-ma, fn=159	grid, omo (for RF)

C. Statistical analyses

1) Significance of differences: We computed paired differences between MAE values for all 471033 $\{$output, sequence$\}$ pairs in phase 2, for a pair of participants. Namely, for a pair (i, j) of participants, and a given $\{$output, sequence$\}$ pair k, we calculated $MAE(j, k) - MAE(i, k)$, then averaged over k. In the results table, i is the line index and j the column index. This quantity should be positive, on average, if i ranks better than j in the challenge. We then computed a quantity analogous to a p-value, that is the fraction of such differences that are negative. Small p-values shed doubt on the null hypothesis that i and j have identical performances (the alternative is that i is better than j). If we estimate that generally what is considered "significant" corresponds to a p-value less than 0.1, none of the differences observed were found significant (not even between top ranking and bottom ranking participants).

2) Relative differences: Finally we computed, for a pair (i, j) of participants, the average *relative performance difference* $2(MAE(j, k) - MAE(i, k))/(MAE(j, k) + MAE(i, k))$, averaged over $\{$output, sequence$\}$ pair k. We show the result table in (Figure 8), where i is the line index and j the column index.

We distinguish three clusters and a singleton (Ose Engineering). We notice a pair of very similar methods Team A and Team B performing best and a nearby a cluster of 4 other gradient boosted trees. We have another cluster of 3 neural networks performing worse than all gradient boosted trees, and in last position, Ose Engineering.

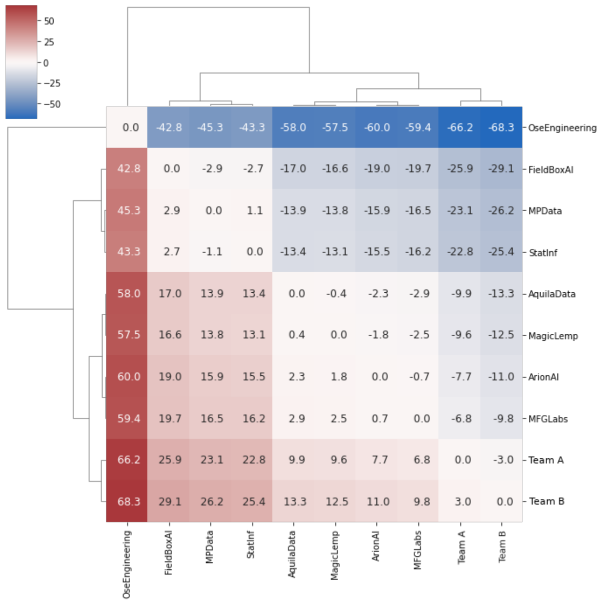


Fig. 8: **Relative differences** in MAE over all 471033 $\{$output, sequence$\}$ pairs in phase 2 (**in percent**). Positive values mean that method i is better than method j on average (i is the line index and j the column index). Lines and columns have been grouped according to a hierarchical clustering algorithm.

VI. Post challenge analyses

A. Alternative metrics

The MAE metric used for the challenge was compared with two other metrics: R^2 and the Pearson correlation coefficient. While the ranking of participants according to MAE and R^2 was similar, we observed noticeable differences between MAE and Pearson correlation coefficient, as illustrated in Figure 9. In particular, the

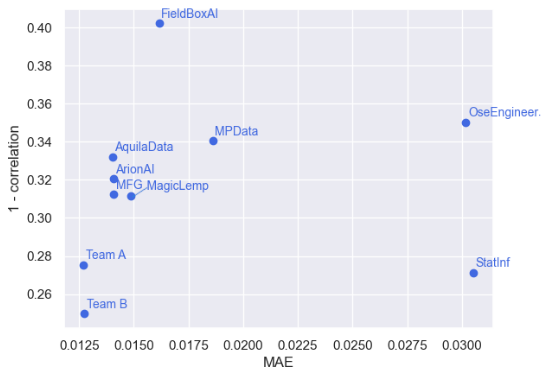


Fig. 9: MAE vs (1 - Pearson's ρ).

last team in the final leaderboard have the second best Pearson's correlation. The Pearson correlation factors out one specific prediction difficulty of the challenge: the time series offsets between flight sequences. This may be due to the fact that the aircraft is heavier when fully loaded with fuel.

B. *Ensembling*

We investigated whether the methods proposed were complementary or not, encouraged by the correlation matrix (see supplemental material). We created voting ensembles of the top 5 participants and all participants, using either the median or the mean of the predictions made. The best attempt, yielding a final test set MAE performance of 0.01_{224}, is obtained when taking the median of all methods' predictions. It is slightly better than the MAE of the best participant 0.01_{269}, but not significantly.

C. *Code open-sourcing*

We are fortunate enough that the code of two teams (Aquila-Supméca and Statinf) were open-sourced[2]. Since the data of the challenge remain confidential, we evaluated their code on a standard benchmark from the "Time Series Extrinsic Regression Archive" [17] (TSER). The results are presented in supplemental material. Without tuning any hyper-parameter on the TSER benchmark, the solutions of the challenge participants fare well, compared with the best performing algorithm of the TSER benchmark [18] and a popular AutoML package (AutoGluon [19]).

VII. Conclusion

The results of the challenge indicate that it is possible to model virtual strain gauges, using as input sensor data from an aircraft. Such model could helps us improve aircraft maintenance and future design. Techniques based on feature engineering and gradient boosted trees performed significantly better than both "classical" linear methods and deep learning methods. The performances of the participants ended up very close to one another by the end of the challenge. The various statistical analyses show that the average relative performance difference separates two teams (Team A and Team B) from the rest. But, the difference between the two first ranking participants is not significant according to any metric, and the rank can even be swapped depending on the metric. Two teams open-sourced their code, making available a range of solutions based on gradient boosted trees and deep-learning. Tested on a standard time-series regression benchmark, these solutions fare well, without any hyper-parameter tuning.

[2]TeamAquila-Supméca: https://tinyurl.com/wetz7sst; TeamStatinf: https://github.com/Anonymous-teams/Challenge-TeamJ.

References

[1] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*, ser. Springer Series in Statistics, 2009.
[2] R. J. Hyndman and G. Athanasopoulos, Eds., *Forecasting: principles and practice*. OTexts, 2021.
[3] V. Harasymiv, "Lessons from 2 million machine learning models on kaggle," *KDnuggets*, 2015.
[4] Z. Xu, W. Tu, and I. Guyon, "Automl meets time series regression design and analysis of the autoseries challenge," *CoRR*, vol. abs/2107.13186, 2021.
[5] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems, LBCS-1857*. Springer, 2000.
[6] J. R. Quinlan, "Induction of decision trees," *MACH. LEARN*, vol. 1, pp. 81–106, 1986.
[7] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002.
[8] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367 – 378, 2002, nonlinear Methods and Data Mining.
[9] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.
[10] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *IEEE international conference on acoustics, speech and signal processing*, 2013, pp. 8599–8603.
[11] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.
[12] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018.
[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.
[14] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015.
[15] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.
[16] Z. Liu *et al.*, "Winning solutions and post-challenge analyses of the ChaLearn AutoDL challenge 2019," *TPAMI, in Press*, 2021.
[17] C. W. Tan, C. Bergmeir, F. Petitjean, and G. I. Webb, "Monash university, uea, UCR time series regression archive," *CoRR*, vol. abs/2006.10996, 2020.
[18] A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels," *Data Min. Knowl. Discov.*, vol. 34, 2020.
[19] N. Erickson *et al.*, "Autogluon-tabular: Robust and accurate automl for structured data," 2020.
[20] C. W. Tan, C. Bergmeir, F. Petitjean, and G. I. Webb, "Time series extrinsic regression," 2020.

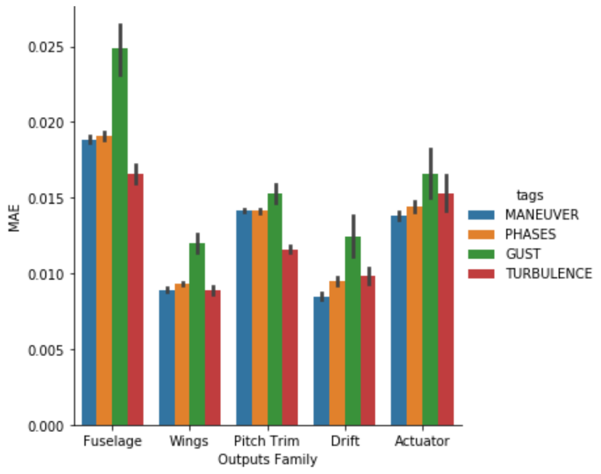


Fig. A.1: Team A: Performance by family of outputs and flight sequence classes.

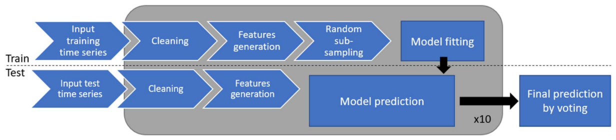


Fig. A.2: Team B: System overview.

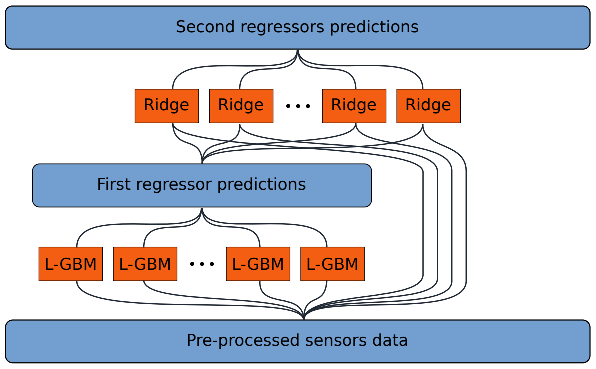

Second regressors predictions
Ridge
Ridge
Ridge
Ridge
First regressor predictions
L-GBM
L-GBM
L-GBM
L-GBM
Pre-processed sensors data

Fig. A.3: Aquila-Supméca: System architecture.

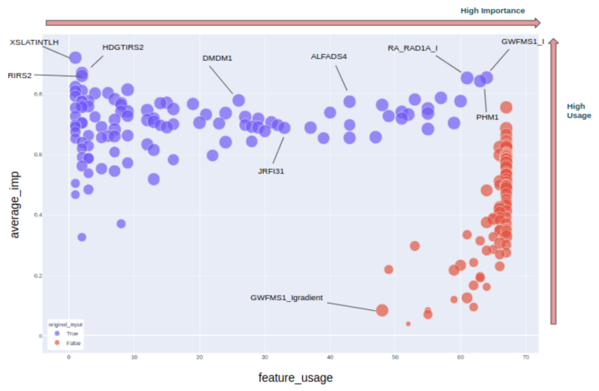


Fig. A.4: Arion AI: Feature importance *vs.* usage assessed by XGBoost. The color indicates an original parameter (blue) or a newly created parameter (red).

Appendix

Supplemental Material

Brief Description of Methods

Team Team A uses CatBoost (one model per output) with a feature representation of 354 features combining ad hoc features and various feature combinations, including lag and lead features. The original aspects include:

- First building one model per type of aircraft.
- Recuperating IRYS data (informations on the various phases of the flight) by writing their own data reader.
- Performing an elaborate stratified data sampling.
- Performing an elaborate feature engineering, with lots of features. Some features constructed by averaging over constant stretches in triplets (Maneuver, Gust, Turbulence).
- Performing feature selection (e.g. lag and lead features seemed to degrade performances.

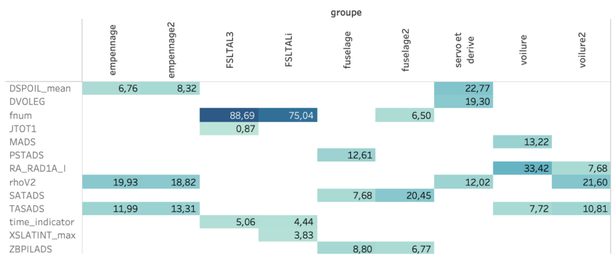
They provide a break up of performances of their system by family of outputs and flight sequence classes (Figure A.1).

Team Team B has a classical pipeline (Figure A.2). The regressor is based on XGBoost (one model per output) with a feature representation of 221 features combining *ad hoc* features, derivatives, various feature combinations, including lag features. The original aspects include:

- Constructing of ad-hoc features w. physical meaning.
- Build models by sub-sampling data from all flights.
- Optimizing number of trees and depth.
- Using multiple experts.
- Using Tukey trimean for ensembling.

Team Aquila-Supméca provided a mathematical formulation of the problem and a physical interpretation of their results. After many explorations of methods that include gradient-boosting, Random Forest, linear regressors, and Multi-layer Perceptrons, they decided to use LightGBM models combined with "stacking" using Ridge Regression (Figure A.3). Their preprocessing included subsampling data to 0.5 Hz, clustering, and data cleaning. The training frugality of the model and its interpretability are the main advantages of the proposed approach. Their team included 4 data scientists, 1 business manager, and 1 academic researcher.

Team Arion AI made an effort to incorporate some aerospace domain knowledge. They studied feature importance for each gauge. Their approach is a simple use of regression with XGBoost, based on intensive feature engineering. They average 3 models for each gauge. The made a nice analysis of feature importance *vs.* feature usage (Figure A.4). Feature importance is evaluated by the average importance of features across all models of the XGBoost ensemble. It is often the case when training gradient boosted tree models not all of the features are used. Feature usage indicates the number of models that used a particular feature.



	groupe								
	empennage	empennage2	FSLTAL3	FSLTALi	fuselage	fuselage2	servo et derive	voilure	voilure2
DSPOIL_mean	6,76	8,32					22,77		
DVOLEG							19,30		
fnum			88,69	75,04		6,50			
JTOT1			0,87						
MADS								13,22	
PSTADS					12,61				
RA_RAD1A_I								33,42	7,68
rhoV2	19,93	18,82					12,02		21,60
SATADS					7,68	20,45			
TASADS	11,99	13,31						7,72	10,81
time_indicator			5,06	4,44					
XSLATINT_max				3,83					
ZBPILADS					8,80	6,77			

Fig. A.5: MFG Labs: Three most important input variables (sensor) for each group of output (gauge).

Team MFG Labs performed many explorations. They put their efforts into building a modular pipeline in terms of data processing and modelization (monovariate or multivariate). Their final solution relies on CatBoost, though among other things they tried Tabnet (attention), and TCN. Their contribution includes :

- Training the ML model on wavelet parameters in lieu of subsampled data and being able to use all the flights while keeping everything in memory.
- Engineering additional features with a physical meaning.
- Building models on groups of targets allowing to predict similar time series together and reduce the inference memory costs.
- Exploring other metrics than RMSE.

They analyzed which input feature predict best output variables (Figure A.6). They conclude that using wavelet coefficients as input to regressors is an effective method to reduce dimensionality and attain a finer grain of analysis on different vibration regimes.

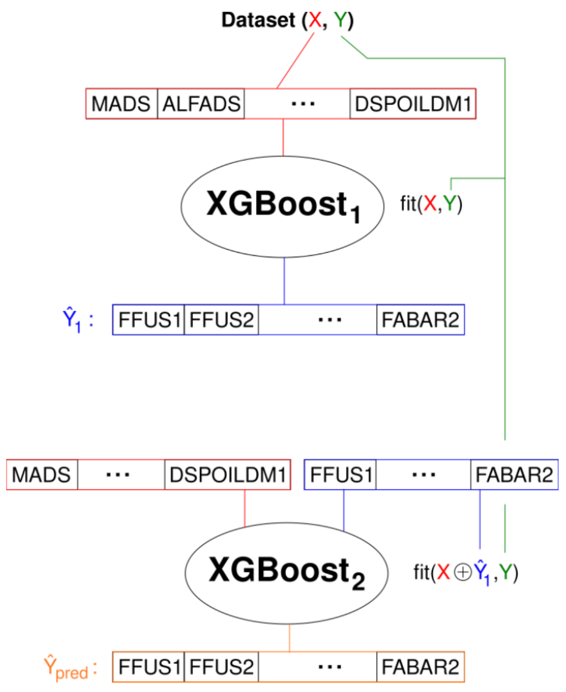


Fig. A.6: Magic LEMP: Cascade XGBoost.

Team Magic LEMP provided a clear and documented report of many explorations they made, demonstrating their strong academic expertise, ranging from gradient boosted trees to sophisticated deep-learning methods. Their final solution is based on a method their proposed called cascade XGBoost (Figure A.6). They are the only ones to provide an error quantification in their analysis. Their feature construction includes an FFT and they end up with 200 features.

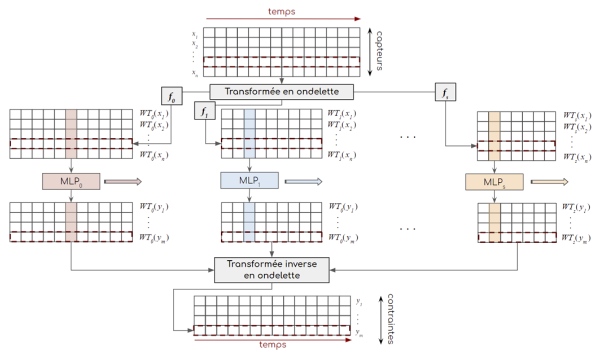


Fig. A.7: TeamG: Frequency-domain model.

Team FieldBox.ai performed a detailed analysis of the YRIS labels indicating the various phases of the flight and an analysis of various data anomalies, leading to an informed preprocessing. After reviewing the machine learning and statistics literature on time series processing, they developed a convolutional neural network (time domain model). After that, they also developed a frequency-domain model (Figure A.7) using a discrete wavelet transform, followed by 6 regular Multi-Layer Perceptrons (MLP), one for each scale. One original aspect is that they propose to also evaluate the confidence in predictions using a technique called MC dropout. Finally, they performed a feature importance study along with a sensitivity-based explainability analysis. Their team consisted of 2 engineers. They conclude that mixed DWT + MLP model offers top-level performances, model training frugality as well as native explainability and uncertainty features, along with refinement possibilities.

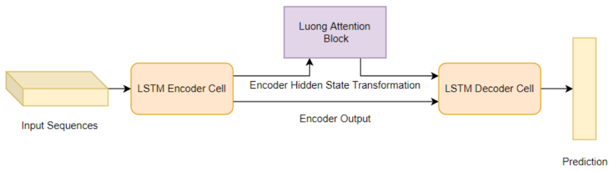


Fig. A.8: MP Data: Attention-based model.

Model	*MAE Score (Training Phase)*
Naïve Fully Connected	0.0378
Attention Based Net	0.0328
CNN with phase as input	0.0294
CNN with 10s samples (Centered Window)	**0.0223**
CNN with 10s samples (Left Window)	0.0244

Fig. A.9: MP Data: Comparison of methods.

Team MP Data conducted an extensive comparison of neural-network models, including Fully Connected MLP, Attention Based Net LSTM (Figure A.8), CNN with phase as input, CNN with 10s samples (Centered Window). The CNN with centered window gave best performances (Figure A.9) . Using 10 second samples improved over training on the whole phase. However, the performances of the methods varied according to phase, maneuver, and gauge. Other methods tried include Gaussian processes, but they proved impractical due to RAM limitations.

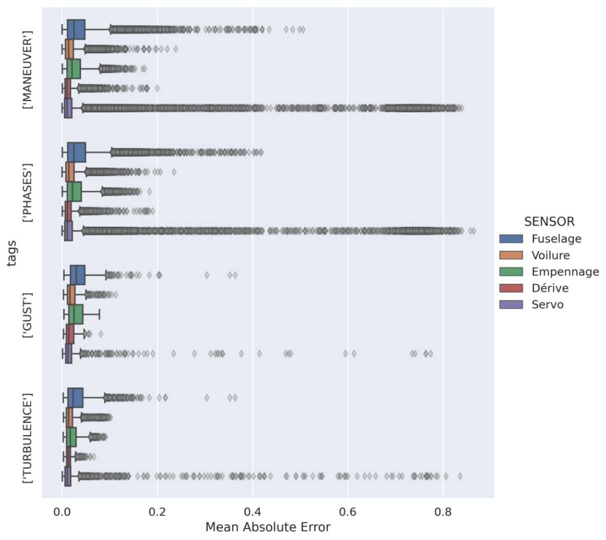


Fig. A.10: Ose Engineering: Results by flight sequence and sensor family.

Team Ose Engineering performed many preprocessing and data cleaning steps, then experimented with several neural network architectures, including CNNs and LSTMs. The model submitted was an LSTM whose hyper-paramaters and learning rate were automatically optimised by grid search. The authors report analyses showing that the model had similar performances on the different flight classes (manoeuvres, turbulence, gust, phases), see Figure A.10, though performance distributions have a long tail. The main failure of the model was observed on the "servo" family which could be due to regular drift and saturation on those family sensors in the training flights.

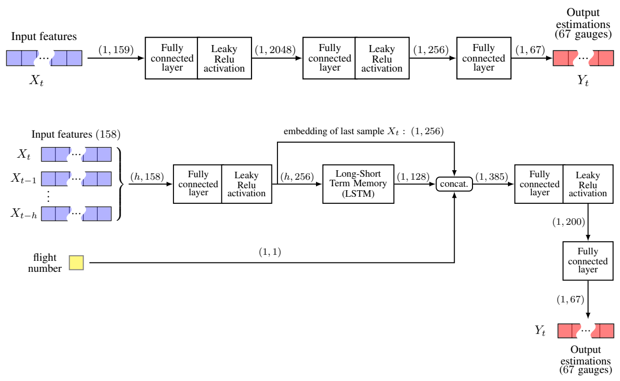


Fig. A.11: MLP and LSTM based models of Statinf

Team Statinf performed rather standard preprocessing steps (filling missing values, resampling, constructing simple features like derivatives, and removing redundant sensors). They then performed a comparison between

MLP, LSTM, RF, and a baseline method making constant output predictions (mean value). The simple MLP model with derivative features provides better performances than the LSTM model used for modeling long and short term history of times series. Hence, using simple model with adapted features could perform better than complicated models. The architectures of the MLP and LSTM based models are shown in Figure A.11. They sucessfully applied their models to the TSER benchmark [17], see results in Table V. Their team includes 3 data scientists and 2 academic researchers

Complementary figures and tables

A. *Number of submissions*

Figure A.12 shows the number of submissions that succeeded of failed for each team and "staff" (the organizers). The top ranking participant (Team Team A) put in significantly more effort. Amazingly enough, the second ranking team (Team Team B) was one of the teams making the smallest number of submissions!

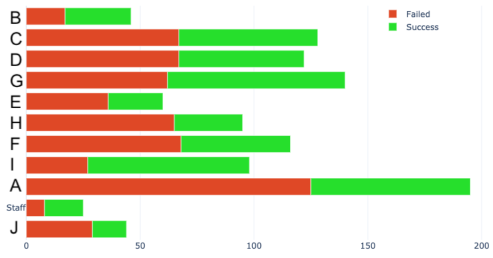


Fig. A.12: **Number of submissions** successful or failed.

B. *Score progression*

Figure A.13 shows how scores have progressed throughout the feedback phase of the challenge. One can see that some teams performed very well early on (Teams A and D, for instance), which might have stimulated others to catch up. Within the last week of the challenge, the scores of top ranking teams became very close to one another. This shows to power of competitions to foster progress.

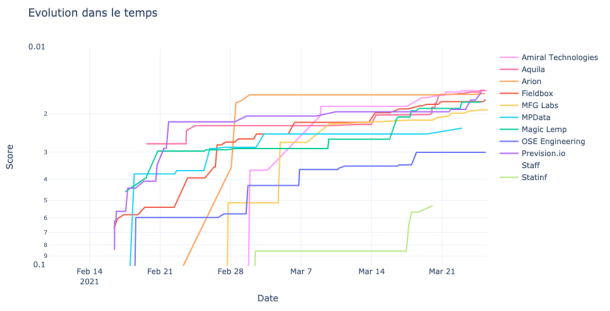
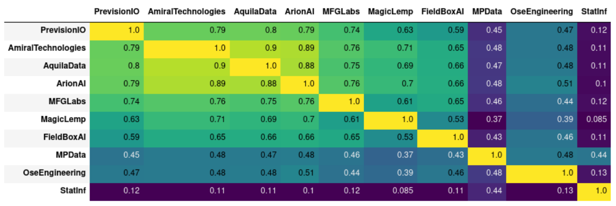


Fig. A.13: **Score progression** during the development phase.

C. *Correlation matrix*

We computed the correlation matrix between MAE values of all sequences, for pairs of participants (Figure A.14). The participants are ordered according to their final rank in phase 2. One can observe that the MAE values are very correlated among top ranking participants, making their methods not complementary. The correlations are particularly strong between the top 4 participants. There is less correlation between the group of gradient boosted trees (first 6) and the group of neural network methods (last 4). Hence the gradient boosted trees and the neural networks could complement each other. However, ensembling experiments did not reveal significant better results when voting among the predictions of several models.



	PrevisionIO	AmiralTechnologies	AquilaData	ArionAI	MFGLabs	MagicLemp	FieldBoxAI	MPData	OseEngineering	StatInf
PrevisionIO	1.0	0.79	0.8	0.79	0.74	0.63	0.59	0.45	0.47	0.12
AmiralTechnologies	0.79	1.0	0.9	0.89	0.76	0.71	0.65	0.48	0.48	0.11
AquilaData	0.8	0.9	1.0	0.88	0.75	0.69	0.66	0.47	0.48	0.11
ArionAI	0.79	0.89	0.88	1.0	0.76	0.7	0.66	0.48	0.51	0.1
MFGLabs	0.74	0.76	0.75	0.76	1.0	0.61	0.65	0.46	0.44	0.12
MagicLemp	0.63	0.71	0.69	0.7	0.61	1.0	0.53	0.37	0.39	0.085
FieldBoxAI	0.59	0.65	0.66	0.66	0.65	0.53	1.0	0.43	0.46	0.11
MPData	0.45	0.48	0.47	0.48	0.46	0.37	0.43	1.0	0.48	0.44
OseEngineering	0.47	0.48	0.48	0.51	0.44	0.39	0.46	0.48	1.0	0.13
StatInf	0.12	0.11	0.11	0.1	0.12	0.085	0.11	0.44	0.13	1.0

Fig. A.14: **Correlation** between MAE values over all 471033 {output, sequence } pairs in phase 2.

D. *Performance by flight sequence*

Figure A.15 shows box-plots of the distribution the participant performances for all flight sequences in both phases. Overlayed are all the performance points, each one corresponding to the MAE result for one flight sequence (averaged over all outputs), grouped by flight sequence category: "maneuver", "phases", "turbulence", and "gust". We see that, while all top ranking models have a similar average MAE and performance dispersion, the model of Team FieldBox.aihad a dispersion of results, which is quite low in both phases, and that of Team Statinfis quite good in the second phase. These results suggest that performance dispersion could also be an interesting metric by which participants' solutions can be judged.

E. TSER benchmark

In Table V, we compared a number of methods, including AutoGluon [19], Rocket [18] and three algorithms from challenge participants, on the datasets of the benchmark "Time Series Extrinsic Regression Archive" [17]. Team Aquila-Supméca and AutoGluon are mostly based on ensembling techniques, and show good results in comparison to the state-of-the art, given that they were not fine-tuned for those applications. Note that, in order to be more generic, the feature engineering of Team Aquila-Supméca code was removed, leaving a simple XGBoost method. Deep learning methods (MLP, LSTM) obtained surprisingly good results on some datasets (*e.g.* AustraliaRainfall). The models obtained similar but not identical results on the three last datasets. This behavior was also observed on the original benchmark results [20]. The implementation of the TSER benchmark models can be found on GitHub[3]. The code of Team Aquila-Supméca and Team Statinf have been open-sourced.[4].

Acknowledgments

We would like to warmly thank the Ile-de-France region for the support and funding of this project. This work was also supported by the ANR (Agence Nationale de la Recherche, National Agency for Research) under AI chair of excellence HUMANIA, grant number Project-ANR-19-CHIA-0022.

[3]https://github.com/Anonymous-Regressors/TS-Extrinsic-Regression.

[4]Team Aquila-Supméca: https://tinyurl.com/wetz7sst; Team Statinf: https://github.com/Anonymous-teams/Challenge-TeamJ.

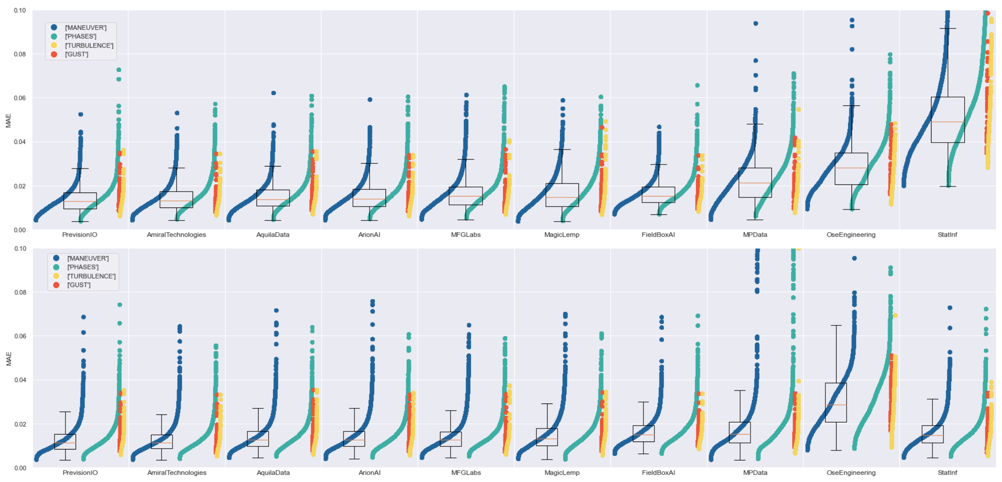


Fig. A.15: **Performance by flight sequence**. Phase 1 (top) and phase 2 (bottom), each point is the average MAE obtained on one flight sequence. Flight sequences are grouped in four categories: “maneuver”, “phases”, “turbulence”, and “gust”.

TABLE V: **TSER benchmark**: RMSE obtained by AutoGluon [19], Rocket [18] and three algorithms from challenge participants, on the datasets of the benchmark “Time Series Extrinsic Regression Archive” [17].

Dataset	AutoGluon	Aquila-Supméca	Statinf MLP	Statinf LSTM	Rocket
AppliancesEnergy	3.44	3.27	5.59	4.24	**2.30**
HouseholdPowerConsumption1	200.27	257.41	522.23	526.52	**132.80**
HouseholdPowerConsumption2	43.00	46.67	53.72	55.72	**32.61**
BenzeneConcentration	**0.76**	0.93	9.75	8.59	3.36
BeijingPM10Quality	98.02	**93.80**	110.24	101.14	120.06
BeijingPM25Quality	**59.58**	62.04	74.49	68.87	62.77
LiveFuelMoistureContent	43.38	47.80	47.08	44.06	**29.41**
FloodModeling1	0.01	0.02	0.02	0.02	**0.00**
FloodModeling2	**0.01**	**0.01**	0.03	0.02	**0.01**
FloodModeling3	0.02	0.02	0.02	0.02	**0.00**
AustraliaRainfall	10.03	10.26	8.62	8.18	**8.12**
PPGDalia	14.51	16.30	19.45	18.06	**14.05**
IEEEPPG	**30.89**	33.92	34.53	36.16	36.52
BIDMC32HR	13.52	14.76	14.31	*failed*	13.94
BIDMC32RR	4.37	4.38	4.69	*failed*	4.09
BIDMC32SpO2	4.44	4.50	4.94	*failed*	5.22
NewsHeadlineSentiment	**0.14**	**0.14**	**0.14**	**0.14**	**0.14**
NewsTitleSentiment	**0.14**	**0.14**	**0.14**	**0.14**	**0.14**
Covid3Month	**0.04**	**0.04**	**0.04**	**0.04**	**0.04**