Automated machine learning for remaining useful life estimation of aircraft engines

Kefalas, M.; Baratchi, M.; Apostolidis, A.; Herik, D. van den; Bäck, T.H.W.

Automated Machine Learning for Remaining Useful Life Estimation of Aircraft Engines

Marios Kefalas*, Mitra Baratchi*, Asteris Apostolidis†, Dirk van den Herik† and Thomas Bäck*
*LIACS, Leiden University, The Netherlands, {m.kefalas, m.baratchi, t.h.w.baeck}@liacs.leidenuniv.nl
†KLM Royal Dutch Airlines, Amstelveen, The Netherlands, {asteris.apostolidis, dirk-van-den.herik}@klm.com

***Abstract*—Remaining useful life (RUL) of an asset or system is defined as the length from the current time and operating state to the end of the useful life. It is of paramount importance for safety-critical industries such as aviation and lies in the heart of prognostics and health management (PHM). This paper investigates the usage of automated machine learning (AutoML) for RUL estimation, based on using classical machine learning algorithms for regression. The data is pre-processed by extracting statistical features from expanding windows of the signal in order to uncover the degradation that has been accumulating from the early life of the system or after an overhaul. We evaluate our methodology on the widely-used C-MAPSS dataset and compare our approach to the state-of-the-art deep neural networks (DNNs) and classical machine learning algorithms. The experimental results show that AutoML outperforms or is comparable to traditional machine learning techniques and standard neural networks, while being outperformed by specifically designed neural networks on datasets with multiple fault mode and operating conditions. These results show that with the correct pre-processing automated machine learning is able to accurately estimate the RUL, which implies that such approaches can be industrially deployed.**

I. Introduction

Prognostics and health management (PHM) is a methodology that aims at minimizing maintenance costs and predicting when a failure could occur by the assessment, prognosis, diagnosis, and health management of engineered systems [1]. The core of PHM is failure prognostics. This refers specifically to the phase involved with predicting future behavior and the systems useful lifetime left in terms of current operating state and the scheduling of required maintenance actions to maintain system health [2]. This useful lifetime left is often called the remaining useful lifetime (RUL) [1], the estimated time to failure (ETTF) or remnant life [3], and is defined as the length from the current time and operating state to the end of the useful life [4]. The notice of pending equipment failure, allows for sufficient lead-time so that necessary decisions, personnel, equipment and spare parts can be organized and deployed, thus minimizing both equipment downtime and repair costs.

By leveraging RUL estimation, industries, such as aerospace, can improve maintenance schedules to avoid catastrophic failures and consequently save lives and costs [5]. The industry has to also assure that its asset utilization is optimum by guaranteeing a timely - but not premature - maintenance. This ensures that the aircraft and its installed parts spend maximum time in service and are not exchanged prematurely.

The estimation of the RUL can be done in various ways. *Model-based*, *data-driven* and *hybrid* methods are the most prominent approaches [1], and in general all methods make some use of the sensor data of the equipment and/or maintenance history. Model-based methods (or physics-based methods) rely on an established mathematical model of the system in question and as a result call for a thorough understanding of the system's physics and processes, which can be prohibitively costly in terms of time and money. Data-driven methods are, on the other hand, relatively easier to develop as they do not call for (a lot of) expert knowledge, but they require large amounts of data. Lastly, hybrid (or fusion) methods leverage the advantages of the two previous methods, while trying to minimize their limitations. The previous make data-driven approaches available to a wider audience.

Most data-driven approaches either fall under the category of classic machine learning algorithms (such as random forests (RF)) or the more recently proposed deep neural networks (DNNs). In both cases, though, the estimation of the RUL is a challenging problem. The remaining useful life is not merely a target variable that can be predicted from sensor measurements, but more of a variable that needs to be inferred from a longer trend of degradation patterns and when those begin to occur. Main challenges of this problem, thus, lie in pre-processing the data and defining the target RUL variable for training efficient machine learning models. In addition, one needs to decide which learning algorithm to use from the vast number of options. The selection of a learning scheme, however, implicitly requires that the researcher (or end-user) is aware, able and has the time to make this choice. The design choices of the algorithm and its hyperparameters, next to the choices that need to be made during pre-processing, make this task challenging for end-users. This, often, leads also to selecting an algorithm a-priori or selecting one from a limited list of algorithms, during preliminary experiments. This can result in overlooking learning schemes that could potentially give good or comparable results and direct us to more suitable learning algorithms for the problem at hand.

This motivates our main research question: Can we *automatically* select a high-performing machine learning pipeline for the estimation of the RUL which can result in comparable

*This work is part of the research programme Smart Industry SI2016 with project name CIMPLO and project number 15465, which is partly financed by the Netherlands Organisation for Scientific Research (NWO).

or better results compared to the current techniques. More specifically, our contributions are as follows:

- We present a method for estimating the RUL, based on the use of automatic machine learning (AutoML) [6] which can automatically generate a suitable pipeline.
- We use a data pre-processing technique that involves extracting statistical features from expanded windows of the original (multivariate) time-series.

Our approach is validated on the widely used C-MAPSS dataset [7].

The rest of the paper is organized as follows. In Section II, we present related work done in this field and in Section III we formally define the problem of the RUL estimation. In Section IV, the proposed method and its modules are introduced and in Section V we present the dataset used and discuss the experimental results. Finally, in Section VI we conclude and discuss our limitations and our future work.

II. Related Work

The field of PHM has been widely credited in the past years with numerous contributions from researchers. Industrial applications as well as the scientific challenge of developing methods to forecast a failure, have been the driving forces. In this section, we will present related work in the field. This collection is by no means exhaustive as the amount of work in this field is vast. In this study, we discuss only data-driven approaches and refer the interested reader to [1] and [8] for a more thorough overview of scientific work on PHM.

The use of classic machine learning algorithms is a great example of data-driven methods. In [9] the authors make use of a multi-layer perceptron (MLP), support vector regression (SVR), and relevance vector regression (RVR) in order to estimate the RUL, by feeding the learning algorithms with every time-step. However, this neglects some useful temporal information that could improve prediction performance. To address this issue, the authors of [5] utilize a fixed time window to enclose multivariate data points sampled at consecutive time-steps. This means that during every specific time-step, multivariate data points within the window that covers the current time-step and its several preceding time-steps are fed into the prediction models used (such as support vector machines (SVM), least absolute shrinkage and selection operator (LASSO) regression, k-nearest neighbor regression (KNR), gradient boosting (GB), random forests (RF)).

To cope with potentially highly non-linear relationships, the use of deep neural networks (DNNs) has also been introduced in the field of PHM. In [9], the authors present the first attempt for estimating the RUL using CNN-based regression (for CNN see [10]). The deep architecture allows the network to learn features that provide a higher-level abstract representation of low-level sensor signals, by employing the convolution and pooling layers to capture the salient patterns of the sensor signals at different time scales. However, considering that the collected machinery features are usually from different sensors, the relationship between the spatially neighboring features is not significant. In [11], Li et al. addresses this issue by proposing to use 1-dimensional convolution filters in their CNN. Zhang et al. [12] investigated the use of CNN with extended time window to tackle the RUL estimation problem under varying operating conditions. Furthermore, to improve the prognostic robustness and avoid the sensitivity to the abnormal data, CNN and extreme gradient boosting (XGB) are fused with model averaging (CNN-XGB). Long short term memory networks (LSTM; see [10]) are other widely used approaches in PHM. They, generally, differ from CNNs in that LSTMS belong to the broader category of recurrent neural networks (RNNs). They are designed to effectively process sequential data (such as time-series data) by leveraging their temporal nature. In [13] and [14], the authors developed an LSTM network for the estimation of RUL. In a similar manner, the method proposed in [15] uses an LSTM and proposes a dynamic differential technology to extract inter-frame information to cope with complex operating conditions. Authors of [16] investigate the effect of unsupervised pre-training in RUL predictions utilizing a semi-supervised setup to extract degradation related features from raw unlabeled training data automatically. The results suggest that unsupervised pre-training is a promising feature in RUL predictions subject to multiple operating conditions and fault modes.

These recent studies have made a great contribution to the field of PHM. However, the design choices of the algorithm and its hyperparameters, next to the choices that need to be made during pre-processing, make this task challenging are often selected a-priori. This can lead to the overlooking of some techniques, that could potentially give good or comparable results. In this work, we present an approach for estimating the RUL, based on the use of automatic machine learning (AutoML) [6] which can suggest to us a suitable pipeline. As a first step towards automatically selecting a machine learning pipeline, in this study we are *only* using classic machine learning algorithms in the generated pipelines.

III. Problem Definition

The estimation of the remaining useful life (RUL) is of paramount importance for safety-critical industries, such as aviation, and lies in the heart of prognostics and health management (PHM) [1]. The RUL of an asset or system is defined as the length from the current time and operating state to the end of the useful life [4]. Because the adjective *useful* is subjective, the previous definition can be extended to the time when the extent of deviation or degradation of the performance from its expected normal operating conditions exceeds a *predefined* threshold [17], when the system needs to be repaired or retracted. Based on this, we can define the RUL at time $t \in \mathbb{R}_{\geq 0}$ as:

$$RUL(t) = \inf\{s \in \mathbb{R}_{\geq 0} : s \geq t \wedge \mathbb{1}_{\{CI(s) \in \mathbb{S}^{\complement}\}}\} - t \ ,$$

where $\mathbb{1}$ is the indicator function, CI represents a user-specified condition index defined at every time-step, and $\mathbb{S}$ is a user-defined system operational envelope, outside of which ($\mathbb{S}^{\complement}$) the system must be repaired or maintained.

IV. Proposed Method

The proposed framework is summarized in Figure 1. We start by pre-processing the data, removing any redundant signals and normalizing the remaining sensor values, before transforming the data using an expanding window. After the expanding window transformation, we extract features from each expanded window and construct the RUL-targets (or labels) needed, in order to approach this problem as a regression problem. The previous steps result in a tuple of (features, target/labels): $\langle f_1, \ldots, f_n, t \rangle$ where each f_i is a feature and t is the target/label, that will be used by the learning algorithm. The next step involves feature selection, to remove any redundant features from the created dataset. Finally, we feed the transformed dataset into an automatic machine learning module, which will use the data in order to automatically suggest a pipeline that will efficiently solve the task at hand.

A. Pre-processing

Given a set of training instances (or units) U, for each instance $u \in U$ we consider multivariate time-series of sensor readings $\boldsymbol{X}_u = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{T(u)}]^T \in \mathbb{R}^{m \times T(u)}$, with $T(u)$ time-steps where the last time-step corresponds to the end-of-life (EoL) of the unit u. Each point $\boldsymbol{x}_t \in \mathbb{R}^m$ is an m-dimensional vector corresponding to readings from m sensors at time-instance t.

Sensor selection is an initial step of pre-processing multivariate time-series data. It involves filtering the available data from sensor measurements which, for example, either do not exhibit any correlation with the target or have strong correlations with other sensors. In the latter case, we usually discard some of the correlated features. Furthermore, even if no correlation is present but the sensors do not exhibit any variation, it is often the case that these features can be discarded as they do not add any valuable information. What is more, having a large number of sensors is not always beneficial for training models as it increases that chance of overfitting.

Pre-processing also involves normalizing the available data, in order to mitigate any effect that different ranges of values or large deviations can have in the subsequent learning phase. Two of the most often used normalization methods are Z-normalization and Min-max normalization:

- Z-normalization (or standardization): This normalization transforms the data into having 0 mean and unit variance as: $x' = (x - \mu)/\sigma$;
- Min-max normalization (or rescaling): This normalization maps the range of the data into $[0, 1]$ or more generally into $[a, b]$ as: $x' = a + \frac{(x - \min(S))(b-a)}{\max(S) - \min(S)}$,

where S is a feature (e.g., a sensor), x, x' are the value and the transformed value of the feature S, and μ, σ are the mean and standard deviation of S, respectively. In addition, a, b are the lower and upper bounds of the projection, and $\min(S)$, $\max(S)$, are the minimum value and maximum value of S, respectively. Normalization is applied on every sensor/feature independently.

As a next step, for each $\boldsymbol{X_u}$, we start by taking the first w time-steps (sensor readings) and perform what we call an expanding window transformation. We do this, by expanding a window of size w from the initial time-step ($t = 0$), until we reach the last time-step. In Algorithm 1, we describe this transformation.

In general time-series problems, the aim is to forecast future time-steps based on the recent history or predict/identify anomalous recordings. These problems can rely on a moving or rolling window in the recent time from when we would like to make a prediction. The RUL estimation, however, is an intrinsically more complicated task. We are dealing with (usually) multi-variate, non-stationary data, where degradation has been accumulating due to usage. Thus, all previous time-steps can be relevant for the problem at hand. The reason for using an expanding window, rather than a moving or rolling window, is that RUL at a particular time-step reflects not only the degradation at that time-step or its w previous time-steps. Instead, it carries also the degradation that has been accumulating from the early stages of the unit's usage or after an overhaul, assuming that there are no major maintenance steps.

Algorithm 1: Expanding window algorithm

```
Data: X_u, w      // Sensor measurements, window size
Result: W^u       // List of expanded windows of unit u
W_u ← ∅; T(u) ← |X_u|; increment_size ← w;
for i ← 1 to T(u) do
    if increment_size < T(u) then
        W_i^u ← X_u[0 : increment_size];
        W^u ← W^u ∪ W_i^u;
        increment_size ← increment_size + w;
    else if increment_size ≥ T(u) then
        W^u ← W^u ∪ X_u; break;
end
Return W^u
```

B. Target-RUL Construction

We would like to tackle this problem as a regression problem. However, one of the main challenges of RUL estimation is the lack of ground-truth values [9]. In the majority of cases, the only available data are the data from the sensor measurements. These data, though, are not labeled with any information regarding the RUL, such as maintenance times. The latter is important and needed for the training procedure as it caries important information that will allow the learner to uncover rules that estimate the RUL given sensor measurements. There are two popular ways to create these, namely linear and piece-wise linear [9]. The former interprets the RUL in the strictest sense, as time to failure. Thus, every time-step is mapped to a value equal to $EoL - t$, where t is the current time-step. This approach, however, implies that the health of the system degrades linearly with usage [9], reflecting the fact that initially the degradation is negligible and after a specific point in time it becomes more evident (see Figure 2 for an

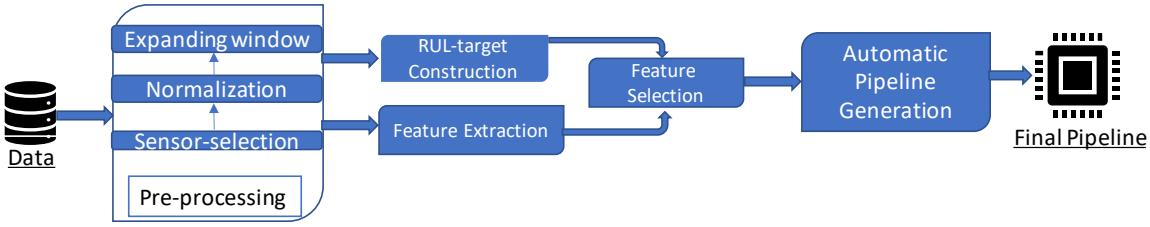

Fig. 1. Diagram of the proposed framework.

example). The point after which the RUL degrades linearly is called the *reflection point* [14].

This way we can construct a RUL curve for each $u \in U$. We do this by mapping each expanding window W_i^u to a $Y_i^u \in \mathbb{N}$ representing the RUL at the end of that window, for every $i = 1, \ldots, k_u$, where $k_u = |W^u|$.

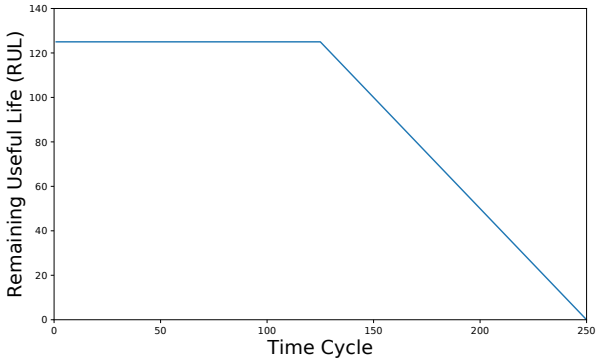

Fig. 2. Toy example of a piece-wise linear RUL target function. The reflection point is at time cycle 125.

With the previous steps, the original data is transformed into a tuple (W, Y), with $W = \cup_{u \in U} W^u \subseteq \mathbb{R}^{m \times T}, Y = \cup_{u \in U} Y^u \in \mathbb{N}^{n \times 1}$, $n = \sum_{u \in U} k_u$, $T = \max\{T(u) : u \in U\}$.

C. Feature Extraction

Feature extraction is the process of extracting a set of new features from the original features through functional mappings that will be used as input in the learning algorithm [18]. Without informative features, it is not possible to train a model that generalizes well, but if relevant features can be extracted, then even a simple method can show remarkable results [19]. In addition, in this particular method, feature extraction allows to translate data from different window sizes (expanding window) into feature vectors of the *same* size, which is needed for the learner.

In this study, the feature extraction $\boldsymbol{\mathcal{F}}$ uses the expanding windows W_i^u of each unit as input and constructs a d-dimensional (d is the number of features) real-valued feature vector, $\boldsymbol{\mathcal{F}}$: $\mathcal{A} \subseteq \mathbb{R}^{T \times m} \rightarrow \mathbb{R}^d$: $\forall(u, i)$, $W_i^u \mapsto \boldsymbol{\mathcal{F}}(W_i^u)$. Thus, each tuple (u, i) results in a feature vector which can be denoted as $\boldsymbol{\mathcal{F}}^{(u,i)}$. This feature vector represents the input for the feature selection phase.

D. Feature Selection

The feature selection phase deals with the selection of relevant features from the, possibly massive, number of extracted features of the input data. It does this while reducing effects from noise or irrelevant variables and still providing good prediction results for the task at hand [20]. Feature selection can allow for shorter training times. It also increases generalization by reducing overfitting and makes the models simpler by using those features that are relevant for the model construction phase. Furthermore, it allows for a better understanding of the data [20]. Numerous feature selection methods have been proposed, which can be divided into (i) *wrapper methods*, (ii) *embedded methods* and (iii) *filter methods* [20].

With the aforementioned steps, the RUL estimation task turns into a regression problem, where input data corresponds to the statistical features from each expanded window and the respective labels are the generated RUL values.

The next step is to automatically find an optimal pipeline for our transformed data. This way, a machine learning algorithm can learn from the statistical features when the end of life is approaching. This assumption is based on the fact that degraded signals must manifest statistical properties that reflect the state of the unit.

E. Automatic Modeling

Automated machine learning (AutoML) deals with the automation of the process of applying machine learning to real-world problems. In general, AutoML covers the complete pipeline from processing the raw data to the deployment of the model, and it was proposed as an artificial intelligence-based solution to the ever-growing challenges of applying machine learning in an efficient manner [6]. In more detail, AutoML aims to solve the so-called CASH problem, standing for combined algorithm selection and hyperparameter optimization [21]. This is essentially the task of choosing the right machine learning model for the dataset at hand, along with the right pre-processing method(s) and the various hyperparameters of all involved components in the pipeline, without requiring human intervention [6].

AutoML systems, however, do not support the pre-processing steps that we introduced in the previous paragraphs. This is why it is important to bring the original raw data into a form that can be processed further by an AutoML system. AutoML can, furthermore, target various stages of the machine

learning process from pre-processing to model selection and hyperparameter optimization.

V. Experimental Setup and Results

We are interested to see if the use of AutoML for automatically selecting a pipeline, in combination with using statistical embeddings from expanding windows in the pre-processing phase, yield better or comparable results to existing methods of RUL estimation. Experiments, dataset and comparison to state-of-the-art methods are described in this section.

A. Data

In this study, we use the widely used C-MAPSS benchmark dataset [7]. The dataset was released in 2008 [17] and it has been used in the field of PHM ever since, in order to develop techniques and methods for estimating the RUL [8], [22]. It is a simulated turbofan engine degradation dataset from NASA's Prognostics Centre of Excellence[1]. The dataset consists of four subsets: FD001, FD002, FD003, and FD004. Each of these datasets is arranged in an $n \times 26$ matrix where n corresponds to the number of data points (samples) in each unit and 26 is the number of columns/features. Each row is a snapshot of data taken during a single operating time cycle. Regarding the 26 features, the 1st represents the engine number, the 2nd represents the operational cycle number. Features $3-5$ represent the operational settings, and columns $6-26$ represent the 21 sensor values. Engine performance can be significantly affected by the three operating settings. More information about these 21 sensors can be found in [23]. What is more, each subset exhibits a different number of faults (see Table I).

Each of these subsets are further split into training set and test set (see Table I for details). For each engine trajectory within the training sets, the last data entry corresponds to the end-of-life (EoL) of the engine, i.e., the moment the engine is declared unhealthy or in failure status. The test sets contain data up to some time before the failure and the aim here is to predict the RUL for each of the test engines.

These multivariate time-series are from a different engine i.e., the data can be considered to be from a fleet of engines, of the same type though, and each trajectory is assumed to be the life-cycle of an engine. Every engine starts with different degrees of initial wear and manufacturing variation which is unknown to the user. This wear and variation is considered normal, i.e., it is not considered a fault condition.

To compare the model performance on the test data, we need some objective performance measures. In this study, we used two measures: the *Scoring function S* (also known as *Timeliness* in literature), and the *Root Mean Square Error (RMSE)* [9], [11], [13], [14], [16]. We introduce them below (n denotes the number of samples):

- The Scoring function S (see also [17]), is defined as:

$$S = \begin{cases} \sum_{i=1}^{n} (\exp(-d_i/13) - 1) & \text{if } d_i < 0 \\ \sum_{i=1}^{n} (\exp(d_i/10) - 1) & \text{if } d_i \geq 0 \end{cases}$$

- RMSE (root mean squared error) is defined as $RMSE = \sqrt{1/n \sum_{i=1}^{n} d_i^2}$, where $d_i = \hat{RUL}_i - RUL_i$, $\hat{RUL}_i$ is the estimated RUL and RUL_i is the ground truth RUL for instance (engine) i, respectively.

The scoring function S penalizes more an overestimation than an underestimation. The scoring algorithm is asymmetric around the true time of failure, such that late predictions are more heavily penalized than early predictions. In both cases, the penalty grows exponentially with increasing error. The asymmetric preference is controlled by parameters 13 and 10 in the scoring function, as introduced in [17]. This is logical, as an overestimation of the RUL for a turbofan engine can have catastrophic results.

TABLE I
CMAPSS dataset details

Data-Set	FD001	FD002	FD003	FD004
Train trajectories	100	260	100	249
Test trajectories	100	259	100	248
Operating conditions	1	6	1	6
Fault conditions	1	1	2	2
Max train trajectory (cycles)	362	378	525	543
Min train trajectory (cycles)	128	128	145	128
Max test trajectory (cycles)	303	367	475	486
Min test trajectory (cycles)	31	21	38	19
Training samples	20631	53759	24720	61249

B. Experimental Setup

The experiments[2] were executed on 64 cores of 2 *Intel® Xeon® Gold* 6142 CPU, 2.60GHz and 256GB of *DDR4* memory. Source code has been developed in *Python* V3.6.9[3].

1) Pre-processing: Following the steps of Section IV we start by selecting relevant sensors. In detail, sensors $1, 5, 6, 10, 16, 18,$ and 19 in subsets FD001 and FD003 exhibit constant sensor measurements throughout the engine's lifetime. Constant sensor measurements do not provide any useful degradation information for determining the RUL [16]. In addition, these subsets operate under a single operating condition. Thus, the three operational settings are dropped. In this view the sensor measurements retained for subsets FD001 and FD003 are $2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20$ and 21. As a result, 14 sensor measurements out of the total 21 are used as the raw input features, as in [11], [16], [24]. Subsets FD002 and FD004 are more complex due to more operating conditions, making it more challenging for the algorithm to detect degradation patterns in the input data. Thus, for these subsets we decided to retain all three operational settings and all sensor measurements, as in [16]. We continue by pre-processing the data by Z-normalizing (standardizing) the sensor values of the training set and using the learnt parameters to standardize the test set. Next, we apply the expanding window transformation on the data. Typically, a larger window size results in less samples, but allows for a greater overview of

[1] https://ti.arc.nasa.gov/tech/dash/groups/pcoe/

[2] The source code of the experiments can be found at https://github.com/MariosKef/automated-rul .

[3] We used *tsfresh*(0.17.0), *TPOT*(0.11.7), *scikit-learn*(0.24.1), *pandas*(1.1.5), *numpy*(1.19.5).

the degradation process as there is more information available for the target RUL. A smaller window size results in less information being available to map to the respective RUL target, but allows for more samples. As a result, it is also more computationally expensive. To ease the computational burden we use a window size $w = 10$ in this study. Regarding RUL-target construction, we use the piece-wise linear approach, as we consider it to reflect more accurately a degradation of a turbofan engine [24], since these machines are designed to sustain multiple cycles and excessive loads and stress. Furthermore, this is still the most common approach in literature [16]. The values of the initial, constant, RUL were selected from [16], and the reflection point is selected as EoL$/2$ [14].

TABLE II
HYPERPARAMETERS USED IN THE EXPERIMENTS

Hyperparameter	FD001	FD002	FD003	FD004
Window size (w)	10	10	10	10
Initial RUL	115	135	125	135
Reflection point	EoL/2	EoL/2	EoL/2	EoL/2
Generations	10	10	10	10
Population size	20	20	20	20
CV	5	5	5	5
Objective	Score S	Score S	Score S	Score S

2) Feature Extraction: In the feature extraction phase, we use the *tsfresh* pipeline (Time Series Feature extraction based on scalable hypothesis tests) [25], since one of our main research questions concerns statistical embeddings of the signal in question and specifically if they can reflect the degradation process. *Tsfresh* extracts 63 time series characterization methods (e.g., auto-correlation, kurtosis, skewness). By taking different parameterizations (i.e., different time lags when calculating the auto-correlation) for each feature function, it computes 794 features for each time-series[4]. The use of *tsfresh* allows extraction of a multitude of features by non-experts and it allows for the identification of features that might be more informative from traditional ones in a given field. *tsfresh* has been applied with *EfficientFCParameters* as its extracted features list to reduce the computational cost. The rest of the input parameters are left in their default settings [5].

3) Feature Selection: In the subsequent step, we select the relevant features for the overall regression task, in order to reduce the massive number of extracted features in $\boldsymbol{\mathcal{F}}\left(W_i^u\right)$. We decided to use a filter method for this phase, in order to select a subset of features independently from the learning scheme. We check the significance of all extracted features from the previous step to the target RUL values. We return a possibly reduced feature matrix only containing relevant features for the subsequent steps. For this step *tsfresh.select_features*, which calculates the feature significance of a real-valued feature to a real-valued target as a p-value, using Kendalls tau. The algorithm has been applied with its default settings[6].

4) Automatic Modeling: We approach this regression problem without the use of an a-priori selected pipeline, aiming to automatically identify the pipeline that gives the best cross-validated score on the training set. To tackle this, we used *TPOT* (Tree-based Pipeline Optimization Tool) [6], [26]. Based on genetic programming (GP) [27], *TPOT* develops and optimizes machine learning pipelines in an automatic manner. The pipeline's operators (pre-processing, feature selection, models) with the respective hyperparameters are combined in a pipeline. Based on GP the whole sequence and each operator are evolved, optimizing a performance metric. *TPOT* is designed for Pareto optimization, in order to optimize the pipeline according to a performance measure (e.g., accuracy, MSE) and simultaneously minimize its complexity. Compared to basic machine learning approaches, *TPOT* is considered efficient and competitive [26], [28]. We used a population size of 20 for all datasets and evolved the pipelines for 10 generations. The *TPOT* default settings, 5-fold cross validation and maximum evaluation time of 5 minutes, were used. Lastly, we should note here that *TPOT* allows for different scoring functions to be defined and used as an objective in its optimization process during training. We performed our experiments using the *Scoring function S* in *TPOT* during the training process. The remaining hyperparameters used in *TPOT* were kept in their default setting[7]. In Table II we show all the hyperparameters used in this study and their values.

5) Baseline: We also performed a baseline experiment in order to evaluate our main ideas and the pre-processing. Our baseline disregards the temporal aspect of the problem at hand and as a result no expanding window transformation takes place or feature extraction. Moreover, since this is a time agnostic method, we also decided not to use the piece-wise linear function for the RUL construction, but instead the linear scheme. We also did not use feature selection prior to using *TPOT* like in the proposed method, as we did not extract features. All other pre-processing remains the same (sensor selection, standardization of sensor values). The transformed dataset is fed again in an AutoML learning scheme. We use this baseline in order to to show the benefits of using the expanding window and the statistical features together with the specific RUL construction in an AutoML setting.

C. Results

After *TPOT* terminates, it returns the optimal pipeline with respect to its cross-validated score. The optimal pipeline is then applied to the test data. The test data have also been transformed in the feature extraction phase just like the training data. However, the test data *do not* undergo an expanding window transformation prior to the feature extraction phase, because we are interested in estimating the RUL of the test instance and therefore statistical features extracted from the

[4]See https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html, for a complete list of features.

[5]https://tsfresh.readthedocs.io/en/latest/_modules/tsfresh/feature_extraction/extraction.html$\#$extract_features

[6]https://tsfresh.readthedocs.io/en/latest/_modules/tsfresh/feature_selection/selection.html\#select_features

[7]http://epistasislab.github.io/TPOT/api/\#regression

entirety of the test recordings are needed to make the prediction. Thus, we make predictions on $100, 259, 100$ and 248 test trajectories from all 4 datasets, respectively (see Table I). Regarding our baseline experiment the inference on the test set is simply applied to the final time-step of its trajectory.

In Table III we show the results of both of our experiments (the proposed method and the baseline), on all 4 test datasets. To mitigate any random artifacts, we ran the experiments 10 times. We chose 10 due to the fact that *TPOT* is extremely time consuming. We show both the average and the standard deviation of the values of *Scoring function S* and *RMSE* of our predictions, as well as the average execution times. In **bold** we show that the proposed method has a statistically significant smaller mean compared to the baseline, both in terms of the score S and the RMSE, on all 4 of the datasets. We assessed this by bootstrapping our samples per dataset a total of 10^5 times to create sampling distribution of the means. We then performed a one-sided Wilcoxon signed-rank test with a significance level of $a = 0.01$, in order to check if the sampling distribution of the means of our proposed method is significantly smaller than the sampling distribution of the means of the baseline. The resulting p-value is 0.00[8]. We used a non-parametric test as the sampling distribution of the means of both experiments is not normal. The specific test was further selected as it takes into account that we have paired samples, as both the proposed method and the baseline were evaluated on exactly the same training and test data.

We further compare our proposed method to some of the state-of-the-art methods. In more detail, we compare against selected methods that employ deep neural networks (DNN) (such as CNN and LSTM) and classic machine learning (such as random forests (RF), support vector machines (SVM))), and as such represent the vast majority of employed methods for this problem. The selected algorithms are good representatives of their respective categories as they either serve as the first attempts [9] or have achieved remarkable results [11], [13], [14], [16]. Regarding the application of classic machine learning techniques on this problem, since there have not been many attempts, we report on all of those that to our knowledge exist (random forest (RF), LASSO regression, support vector regression (SVR), support vector machine (SVM), gradient boosting (GB), KNeighbors regressor (KNR), relevance vector regression (RVR), extra tree regressor (ETR)). In Table IV, in **bold**, we show the average results of our proposed method and the instances it outperforms.

1) Classic Machine Learning: The results show that the proposed method outperforms the multilayer perceptron (MLP), support vector regression (SVR), relevance vector regression (RVR) [9], LASSO regression, SVM and KNR regression [5] on all 4 datasets both in terms of the score function S and the RMSE. Moreover, the proposed method outperforms the RF [5] algorithm in terms of both the score and RMSE on FD001 and FD002, of RMSE on FD003 and of score function on FD004. In addition, it outperforms the GB [5] algorithm on FD002 and in terms of RMSE FD001. Lastly, it can outperform ETR [5] on all datasets, except FD004 in terms of RMSE, where it is comparable. In general, we see that in terms of the score S, the proposed method outperforms *all* 9 of the classic machine learning algorithms considered here, on $2/4$ datasets (FD001 and FD002) by at least 19%[9] (on FD001) and outperforms $6/9$ of these algorithms on all 4 datasets, by at least 13.2% (on FD003). In terms of the RMSE, our proposed method outperforms *all* 9 of the classic machine learning algorithms considered here, on $1/4$ datasets (FD002) by 3.1% and outperforms $6/9$ of these algorithms on all 4 datasets, by at least 1.9% (on FD004).

2) Deep Neural Networks: When compared to DNNs our method outperforms the first CNN approach [9] on all cases except on FD004. When compared, however, to LSTM [13], [14], [16] and a recent CNN approach with 1D convolution [11] our algorithm is outperformed or comparable in terms of RMSE. In more detail, our proposed method outperforms the CNN [9] on FD001, FD002 and FD003 by at least 22.1% (on FD002) in terms of the score S and by at least 0.6% (on FD003) in terms of the RMSE. Our method is also comparable on FD004 in terms of the RMSE. Regarding LSTMs, our results are comparable to those in [14]. In more detail, our method outperforms [14] by 1.23% on FD001 in terms of the score S, by 5% in terms of the RMSE and on FD002 by 0.8% on the score S and by 4.2% in terms of the RMSE. Our method is also comparable to [14] on FD003 in terms of the RMSE and the score S and on FD004 in terms of the RMSE. Furthermore, it outperforms by 1.5% the algorithm of [13] on FD001 in terms of the RMSE. The proposed method is also outperformed by the other LSTMs [13], [16] and CNN [11] by at least 1.5% (on FD002) in terms of the score S and at least 13.2% in terms of the RMSE (on FD002)[10]. The reason is that the usage of advanced LSTM (e.g., using unsupervised pre-training) and CNN with 1D convolution allow for learning highly non-linear relationships that might describe the mapping between the time-steps and the RUL more accurately, compared to classic machine learning schemes. This also leads to more favourable results on FD004 which incorporates 6 operating conditions and 2 fault modes.

From the previous results we can conclude that the usage of AutoML in combination with extracting statistical features, can outperform or achieve comparable results when compared to classic machine learning techniques. When compared to DNNs, however, our method is comparable or outperformed, one reason being that DNNs have the ability to learn highly non-linear relationships that might describe the mapping between the time-steps and the RUL. What is more, we should

[8]In practice, this means that the p-value returned by the software is a very small float rendering it practically 0.00. Statistically this signifies that the observed samples *can not* come from the same distribution.

[9]The percentage of improvement in this case is calculated as PI=$1 - \frac{\text{proposed\_method\_performance}}{\text{min(other\_methods\_performance)}} * 100\%$, since we are interested to see how much better we perform from the best method (lower is better).

[10]In this case, the percentage is calculated as $1 - \frac{\text{max(other\_methods\_performance)}}{\text{proposed\_method\_performance}} * 100\%$, since we are interested to see by how much the "worst" (lower is better) of the better methods outperforms us.

note here that DNNs were not included in our algorithm search space. Furthermore, using neural architecture search (NAS) based methods can be useful for this. However, some of these approaches use very complex handcrafted pipelines (e.g., using unsupervised pre-training) that cannot be efficiently automated with current NAS systems. Thus, considering NAS for this problem is much broader than the scope of this paper.

VI. Conclusion

In this study, we presented the first, to our knowledge, AutoML approach [6] for the estimation of the RUL of machinery. We investigated the usage of *TPOT* ([6], [26]) in automatically selecting a pipeline for this problem, as well as the usage of statistical embeddings of time-series in the pre-processing phase, using an expanding window transformation.

We evaluated the proposed method on the widely-used C-MAPSS dataset [7]. The gathered results show that the usage of AutoML in combination with extracting statistical features (embeddings) and constructing the RUL in a piece-wise linear manner can outperform or achieve comparable results when compared to classic machine learning techniques (such as SVR, LASSO, SVM). However, when compared to deep architectures such as CNN and LSTM, our method is able to outperform the first CNN approach on $3/4$ datasets, but in general is comparable or is outperformed. This suggests that the combination of statistical features and classic ML might not be able to uncover the highly non-linear relationship between the observed/measured values and the RUL. The proposed method also allows for a useful direction towards which classic machine learning algorithms would be more useful to use, as well as providing the optimal pipeline as a starting point for further research.

As indicated, a limitation of our approach is the investigation of *only* classic ML algorithms (e.g., no neural networks) and no hyperparameter optimization (e.g., for the window size).

As a next step, we plan to include neural networks in the AutoML approach, by means of NAS [29], as well as investigating effective dimensionality reduction techniques for the statistical embeddings. Finally, we want to augment this pipeline by adding a hyperparameter optimization wrapper.

References

[1] V. D. Nguyen, M. Kefalas, K. Yang, A. Apostolidis, M. Olhofer, S. Limmer, and T. Bäck, "A Review: Prognostics and Health Management in Automotive and Aerospace," *International Journal of Prognostics and Health Management*, p. 35, 2019.

[2] G. J. Vachtsevanos, Ed., *Intelligent fault diagnosis and prognosis for engineering systems*. Hoboken, N.J: Wiley, 2006, oCLC: ocm64442758.

[3] J. Sikorska, M. Hodkiewicz, and L. Ma, "Prognostic modelling options for remaining useful life estimation by industry," *Mechanical Systems and Signal Processing*, vol. 25, no. 5, pp. 1803–1836, Jul. 2011.

[4] X.-S. Si, W. Wang, C.-H. Hu, and D.-H. Zhou, "Remaining useful life estimation A review on the statistical data driven approaches," *European Journal of Operational Research*, vol. 213, no. 1, pp. 1 – 14, 2011.

[5] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective Deep Belief Networks Ensemble for Remaining Useful Life Estimation in Prognostics," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2306–2318, Oct. 2017.

[6] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning: Methods, Systems, Challenges*, ser. The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing, 2019.

[7] A. Saxena and K. Goebel, "Turbofan engine degradation simulation data set. NASA Ames Prognostics Data repository, NASA Ames Research Center, Moffett Field," 2008.

[8] M. S. Krishna and K. T. Baghaei, "Recent approaches in prognostics: State of the art," in *Int'l Conf. Artificial Intelligence (ICAI'19)* —, 2019.

[9] G. Sateesh Babu, P. Zhao, and X.-L. Li, "Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life," in *Database Systems for Advanced Applications*, S. B. Navathe, W. Wu, S. Shekhar, X. Du, X. S. Wang, and H. Xiong, Eds. Cham: Springer International Publishing, 2016, vol. 9642, pp. 214–228, series Title: Lecture Notes in Computer Science.

[10] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[11] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety*, vol. 172, pp. 1–11, Apr. 2018.

[12] X. Zhang, P. Xiao, Y. Yang, Y. Cheng, B. Chen, D. Gao, W. Liu, and Z. Huang, "Remaining Useful Life Estimation Using CNN-XGB With Extended Time Window," *IEEE Access*, vol. 7, pp. 154 386–154 397, 2019.

[13] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long Short-Term Memory Network for Remaining Useful Life estimation," in *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. Dallas, TX, USA: IEEE, jun 2017, pp. 88–95.

[14] C.-S. Hsu and J.-R. Jiang, "Remaining useful life estimation using long short-term memory deep learning," in *2018 IEEE International Conference on Applied System Invention (ICASI)*. Chiba: IEEE, Apr. 2018, pp. 58–61.

[15] Y. Wu, M. Yuan, S. Dong, L. Lin, and Y. Liu, "Remaining useful life estimation of engineered systems using vanilla LSTM neural networks," *Neurocomputing*, vol. 275, pp. 167–179, Jan. 2018.

[16] A. Listou Ellefsen, E. Bjrlykhaug, V. sy, S. Ushakov, and H. Zhang, "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture," *Reliability Engineering & System Safety*, vol. 183, pp. 240–251, Mar. 2019.

[17] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *2008 International Conference on Prognostics and Health Management*. Denver, CO, USA: IEEE, Oct. 2008, pp. 1–9.

[18] H. Liu and H. Motoda, *Feature Extraction, Construction and Selection: A Data Mining Perspective*. USA: Kluwer Academic Publishers, 1998.

[19] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative review," *J Mach Learn Res*, vol. 10, pp. 66–71, 2009.

[20] G. Chandrashekar, "A survey on feature selection methods," *Computers and Electrical Engineering*, p. 13, 2014.

[21] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms," *arXiv:1208.3719 [cs]*, Mar. 2013, arXiv: 1208.3719.

[22] E. Ramasso and A. Saxena, "Performance Benchmarking and Analysis of Prognostic Methods for CMAPSS Datasets," *International Journal of Prognostics and Health Management*, p. 15, 2014.

[23] C. Ordez, F. Snchez Lasheras, J. Roca-Pardias, and F. J. d. C. Juez, "A hybrid ARIMASVM model for the study of the remaining useful life of aircraft engines," *Journal of Computational and Applied Mathematics*, vol. 346, pp. 184–191, Jan. 2019.

[24] F. O. Heimes, "Recurrent neural networks for remaining useful life estimation," in *2008 International Conference on Prognostics and Health Management*. Denver, CO, USA: IEEE, Oct. 2008, pp. 1–6.

[25] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh A Python package)," *Neurocomputing*, vol. 307, pp. 72–77, Sep. 2018.

[26] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16. New York, NY: ACM, 2016, pp. 485–492.

[27] J. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and Computing*, vol. 4, no. 2, 1994.

TABLE III

PERFORMANCE METRICS AND WALL-CLOCK TIME (IN MINUTES) OF THE PROPOSED METHOD AND THE BASELINE. HERE THE *Scoring function S* HAS BEEN USED AS THE SCORING FUNCTION IN *TPOT* (*lower is better*). IN **BOLD**, WE SHOW THE OPTIMAL RESULTS.

		Proposed Method			Baseline	
Dataset	Score	RMSE	Execution Time	Score	RMSE	Execution Time
FD001	$\mathbf{383.9 \pm 7.6}$	$\mathbf{15.9 \pm 0.15}$	52.7 ± 25.4	107610.9 ± 64153.6	41.8 ± 5.4	18.4 ± 8.2
FD002	$\mathbf{10567.8 \pm 748.1}$	$\mathbf{28.2 \pm 0.45}$	81.7 ± 11.5	859641.7 ± 999547.1	44.7 ± 12.1	42.9 ± 4.4
FD003	$\mathbf{894.4 \pm 116.3}$	$\mathbf{19.7 \pm 0.5}$	67.4 ± 2.4	$5.32 \cdot 10^8 \pm 3.62 \cdot 10^8$	79.1 ± 14.5	19.5 ± 10.6
FD004	$\mathbf{26649.1 \pm 25617.4}$	$\mathbf{33.7 \pm 1.3}$	86 ± 8.4	$1.58 \cdot 10^8 \pm 1.60 \cdot 10^8$	60.8 ± 15.6	55.4 ± 2.8

TABLE IV

COMPARISON OF THE PROPOSED METHOD WITH OTHER METHODS IN TERMS OF SCORING FUNCTION S AND RMSE (*Lower is better*). IN **BOLD**, WE SHOW THE AVERAGE RESULTS OF OUR PROPOSED METHOD AND THE INSTANCES IT OUTPERFORMS. THE *Type* COLUMN INDICATES THE METHODS THAT BELONG IN THE CLASSIC MACHINE LEARNING DOMAIN AND THE ONES BELONGING IN THE DEEP NEURAL NETWORK CATEGORY.

		FD001		FD002		FD003		FD004	
Type	Algorithm	Score	RMSE	Score	RMSE	Score	RMSE	Score	RMSE
Classic Machine Learning	MLP [9]	**17972**	**37.56**	$\mathbf{7.8028 \cdot 10^6}$	**80.03**	**17409**	**37.39**	$\mathbf{5.6166 \cdot 10^6}$	**77.37**
	SVR [9]	**1381.2**	**20.96**	**58990**	**42**	**1598.3**	**21.05**	**371140**	**45.35**
	RVR [9]	**1502.9**	**23.8**	**17423**	**31.3**	**1431.6**	**22.37**	26509	**34.34**
	RF [5]	**479.75**	**17.91**	**70456.86**	**29.59**	711.13	**20.27**	**46567.63**	31.12
	LASSO [5]	**653.85**	**19.74**	**276923.89**	**37.13**	**1058.36**	**21.38**	**125297.19**	**40.70**
	SVM [5]	**7703.33**	**40.72**	**316483.31**	**52.99**	**22541.58**	**46.32**	**141122.19**	**59.96**
	KNR [5]	**729.32**	**20.46**	**450094.04**	**36.05**	**1030.29**	**22.59**	**234396.56**	**54.44**
	GB [5]	**474.01**	15.67	**87280.06**	**29.09**	576.72	16.84	17817.92	29.01
	ETR [5]	**1359.38**	**22.05**	**231030**	**33.01**	**1757.6**	**24.52**	**69771**	32.42
Deep Neural Networks	CNN [9]	**1286.7**	**18.45**	**13570**	**30.29**	**1596.2**	**19.82**	7886.4	29.16
	CNN [11]	273.7	12.61	10412	19.61	284.1	12.64	12466	23.31
	LSTM [14]	**388.68**	**16.74**	**10654**	**29.43**	822.19	18.07	6370.6	28.4
	LSTM [13]	338	**16.14**	4450	24.49	852	16.18	5550	28.17
	LSTM [16]	231	12.56	3366	22.73	251	12.10	2840	22.66
	Proposed method	**383.9**	**15.9**	**10567.8**	**28.2**	**894.4**	**19.7**	**26649.1**	**33.7**

[28] C. Wang, T. Back, H. H. Hoos, M. Baratchi, S. Limmer, and M. Olhofer, "Automated Machine Learning for Short-term Electric Load Forecasting," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. Xiamen, China: IEEE, Dec. 2019, pp. 314–321.

[29] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *arXiv:1808.05377 [cs, stat]*, Apr. 2019, arXiv: 1808.05377.