# Chapter 4

# Automated Machine Learning and Remaining Useful Life

In chapter 3 we introduced the field of PHM to the reader and discussed its significance in industry and society. We presented the notion of the remaining useful life (RUL) its importance in predictive maintenance (PdM) and classified the estimation approaches in four broad classes: model-based, data-driven, hybrid, and reliability-based approaches. We subsequently emphasized the importance of data-driven methods in the aerospace industry. We showcased various applications of PHM in the field and categorized them based on their underlying methodology. Finally, we presented drawbacks and opportunities for future improvements of data-driven methods.

In this chapter[1], we will discuss the difficulties and challenges that accompany the task of the RUL estimation in the data-driven domain. We will discuss the plethora of choices that the end-user or researcher has when choosing a method to estimate the RUL, and we will introduce the notion of automated machine learning (AutoML). We propose a task-specific data preprocessing technique that involves the extraction of statistical features from the data and an expanding window transformation that aims to collect the degradation information that has been accumulating from the early stages of a unit's usage. We evaluate our propositions against state-of-the-art methods in the field of data-driven prognostics and we validate our method on the widely used C-MAPSS dataset [198].

Lastly, the objective of this chapter is to present AutoML as a feasible tool in the data-driven estimation of the RUL.

[1]©2021 IEEE. Reprinted, with permission, from [112]; Marios Kefalas, Mitra Baratchi, Asteris Apostolidis, Dirk van den Herik, and Thomas Bäck. Automated Machine Learning for Remaining Useful Life Estimation of Aircraft Engines. In 2021 IEEE International Conference on Prognostics and Health Management (ICPHM), pages 1–9, Detroit (Romulus), MI, USA, June 2021. IEEE.

## 4.1 Introduction

The estimation of the RUL allows for managing *pending* equipment failure and grants 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 [253]. The industry also has to 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. We briefly mention them here (for more details see Section 3.3). *Model-based*, *data-driven*, and *hybrid* methods are the most prominent approaches [230], and in general all methods make some use of the sensor data of the equipment and/or maintenance history (see Paragraph 3.3 for more details). Among these approaches, data-driven methods are relatively easier to develop as they do not call for (a lot of) expert or domain knowledge and are, thus, available to a broader audience due to their domain-agnostic nature. Additionally, the recent advances in automated machine learning (AutoML) [97] (see also Section 4.3.5) have made data-driven modeling, in general, more accessible by providing methods and processes to make machine learning available for non-machine learning experts, to improve efficiency of machine learning and to accelerate research on machine learning[2].

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 RUL 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. The main challenges of this problem, thus, lie in pre-processing the data and defining the target RUL variable (if it doesn't exist) for training efficient machine learning models. Such pre-processing steps and related transformations are not readily available in AutoML methods. In addition, one needs to decide which learning algorithm to use from the vast number of options. However, the selection of a learning scheme 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 better or comparable results and direct us to more suitable learning algorithms for the problem at hand.

[2]`https://www.automl.org/automl/`

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 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 AutoML [97] 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.
- We evaluate the proposed method against the state-of-the-art data-driven methods and against a baseline experiment in order to investigate the effects of the suggested steps. Our approach is validated on the widely used C-MAPSS datasets [198].

The rest of the chapter is organized as follows. In Section 4.2, we present related work done in this field and in Section 4.3, the proposed method and its modules are introduced. In Section 4.4 we present the dataset used, the experimental setup, and discuss the experimental results. Finally, in Section 4.5 we conclude, discuss the limitations of our work, and suggest future work.

## 4.2 Related Work

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. This section will present related work in the field, concentrating only on data-driven approaches. This collection is by no means exhaustive, as the amount of work in this field is vast. We refer the interested reader to [230], and [128] for a more thorough overview of scientific work on PHM, as well as Chapter 3 of this dissertation.
Classic machine learning algorithms are a great example of data-driven methods. In [195] 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 [253] 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)).

The use of deep neural networks (DNNs) has also been introduced in PHM to cope with potentially highly nonlinear relationships. In [195], the authors present the first attempt for estimating the RUL using CNN-based regression (for CNN see [98]). 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 [143], Li et al. address this issue by proposing to use 1-dimensional convolution filters in their CNN. Zhang et al. [256] 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 [98]) 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) by leveraging their temporal nature. In [260] and [94], the authors developed an LSTM network for the estimation of RUL. Similarly, the method proposed in [245] uses an LSTM and proposes a dynamic differential technology to extract inter-frame information to cope with complex operating conditions. Authors of [146] 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 approach in RUL prediction problems 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 a challenging one. This can lead to overlooking some models with potentially high performance or pre-processing steps, that could consequently give better or comparable results. In this work, we present an approach for estimating the RUL, based on the use of automatic machine learning (AutoML) [97] which can suggest to users a suitable pipeline. As a first step towards automatically selecting a machine learning pipeline, in this work, we are focusing *only* on pipelines based on classic machine learning.

## 4.3 Proposed Method

The proposed framework is summarized in Figure 4.1. We start the process 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 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 the learning algorithm will use. 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 to automatically suggest a pipeline that will efficiently solve the task at hand.

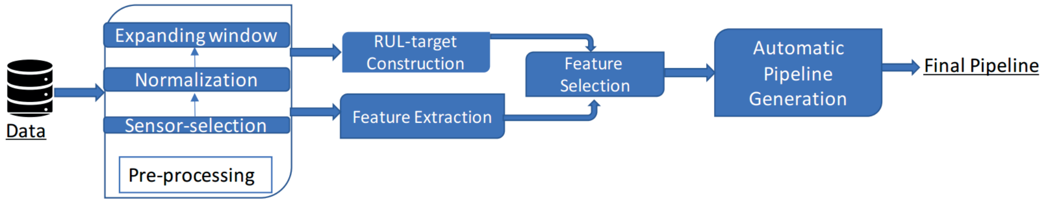

Figure 4.1: Overview of the proposed framework.

## 4.3.1 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 $t$.
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. Having a large number of sensors is not always beneficial for training models as it increases the chance of overfitting.
Pre-processing also involves normalizing the available data 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 $\mu, \sigma$ 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 4.1[3], 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 much more complicated task. We are dealing with (usually) multivariate, 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 also carries 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 in between.

**Algorithm 4.1:** Expanding window algorithm

```
   Data: $\boldsymbol{X}_u = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{T(u)}]^T, w$ ;        # Sensor measurements, window size
   Result: $W^u$ ;                                     # List of expanded windows of unit u
 1 $W_u \leftarrow [\,]$ ; $increment\_size \leftarrow w$ ;
 2 for $i \leftarrow 1$ to $T(u)$ do
 3     if $increment\_size < T(u)$ then
 4         $W_i^u \leftarrow \boldsymbol{X_u}[0 : increment\_size]$ ;
 5         $W^u \leftarrow W^u W_i^u$ ;                   # Appending arrays of increasing size
 6         $increment\_size \leftarrow increment\_size + w$ ;
 7     else if $increment\_size \geq T(u)$ then
 8         $W^u \leftarrow W^u \boldsymbol{X_u}$ ;                   # Appending arrays of increasing size
 9         break ;
10     end
11 end
```

## 4.3.2 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 [195]. In the majority of cases,

[3]Please note that some of the notations in the pseudocode of Algorithm 4.1 differ from the notations in the pseudocode of the original publication [112]. We did this for clarity, as well as for consistency among the chapters of this thesis.

the only available data are the data from the sensor measurements (e.g., in the form of time-series). However, these data 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 carries vital information that will allow the learner to uncover rules that estimate the RUL given sensor measurements. For example, if for a specific asset the times until maintenance are recorded, these can be used as the training labels. The learner then will try to uncover the relationship (if any) between the sensor measurements and the time to maintenance or, in general, the times to an event of interest.

There are two popular ways to create these labels, namely taking a linear and a piece-wise linear approach [195]. 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 [195]. The latter reflects the fact that initially the degradation is negligible, and after a specific point in time, it becomes more evident (see Figure 4.2 for an example). The point after which the RUL degrades linearly is called the *reflection point* [94].

This way, we can construct an 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 $k_u = |W^u|$.

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

### 4.3.3 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 [147]. 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 [229]. In addition, in this particular method, feature extraction allows translating data from different window sizes (expanding window) into feature vectors of the *same* size, a condition which is needed for the classic machine learning model.

In this work, 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. $\boldsymbol{\mathcal{F}}$ is, generally, defined as:

$$\boldsymbol{\mathcal{F}} \colon \mathcal{A} \subseteq \mathbb{R}^{m \times T} \to \mathbb{R}^d : \forall (u, i),\ W_i^u \mapsto \boldsymbol{\mathcal{F}}\left(W_i^u\right), \tag{4.1}$$

where $\mathcal{A}$ is any subset of $\mathbb{R}^{m \times T}$ of appropriate dimensionality[4].

[4]Remember that $T$ was defined as $T = \max\{T(u) : u \in U\}$.

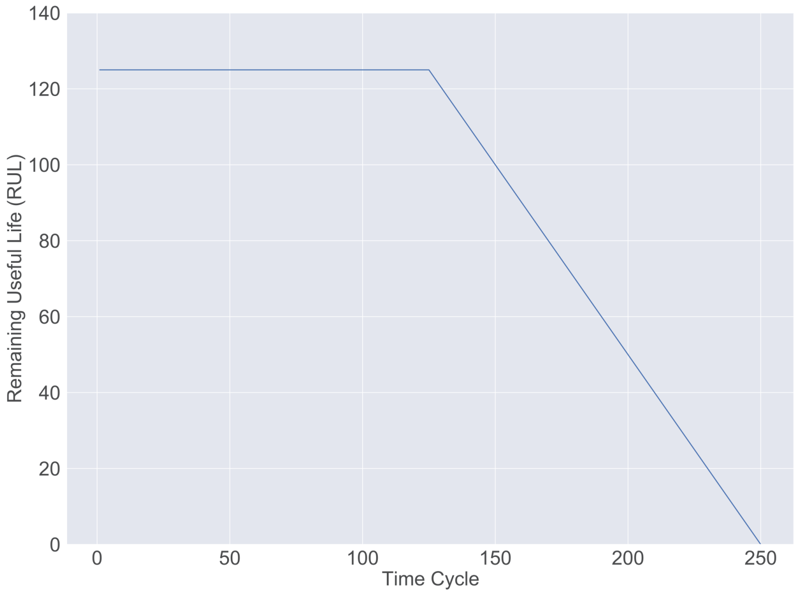

Figure 4.2: Toy example of a piece-wise linear RUL target function. The reflection point is at time cycle 125.

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.

### 4.3.4 Feature Selection

The feature selection phase deals with selecting 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 [36]. Feature selection can allow for shorter training times. It also improves generalization by reducing overfitting and simplifies the models by using those relevant features for the model construction phase. Furthermore, it allows for a better understanding of the data [36]. Numerous feature selection methods have been proposed, which can be divided into (i) *wrapper methods*, (ii) *embedded methods* and (iii) *filter methods* [36].

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 to find an optimal pipeline for our transformed data automatically. 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.

### 4.3.5 Automatic Modeling

Automated machine learning (AutoML) deals with the automation 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 [97]. In more detail, AutoML aims to solve the so-called CASH problem, standing for combined algorithm selection and hyperparameter optimization [221]. This is essentially the task of choosing the suitable machine learning model for the dataset at hand, along with the proper pre-processing method(s) and the various hyperparameters of all involved components in the pipeline, without requiring human intervention [97].
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.

## 4.4 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 yields better or comparable results to existing methods of RUL estimation. Experiments, datasets, and comparisons to state-of-the-art methods are described in this section.

### 4.4.1 Data

In this work, we use the widely used C-MAPSS benchmark dataset [198]. The dataset was released in 2008 [198] and it has been used in the field of PHM ever since, in order to develop techniques and methods for estimating the RUL [183, 128]. It is a simulated turbofan engine degradation dataset from NASA's Prognostics Centre of Excellence[5]. 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

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

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 features $6-26$ 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 [170]. What is more, each subset exhibits a different number of faults (see Table 4.1).

Each of these subsets is further split into training set and test set (see Table 4.1 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, unknown to the user. This wear and variation are 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 work, we used two measures: the *Scoring function S* (also known as *Timeliness* in literature), and the *Root Mean Square Error (RMSE)* [260, 146, 143, 195, 94]. We introduce them below ($n$ denotes the number of samples):

- The Scoring function $S$ (see also [198]), is defined as:

$$S = \begin{cases} \sum_{i=1}^{n}(\exp(-d_i/13) - 1) & \text{if } d_i < 0 \\ \sum_{i=1}^{n}(\exp(d_i/10) - 1) & \text{if } d_i \geq 0 \end{cases} \tag{4.2}$$

- RMSE (root mean squared error) is defined as $RMSE = \sqrt{1/n \sum_{i=1}^{n} d_i^2}$,

where $d_i = \widehat{RUL_i} - RUL_i$, $\widehat{RUL_i}$ 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 the constants 13 and 10 in the scoring function, as introduced in [198]. This is logical, as a turbofan engine's overestimation of the RUL can have catastrophic results.

Table 4.1: CMAPSS dataset details.

| **Dataset** | **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 |

### 4.4.2 Experimental Setup

The experiments[6] 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[7].

#### Pre-processing

Following the steps of Section 4.3 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 [146]. In addition, subsets FD001 and FD003 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 [146, 88, 143]. 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 [146]. 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 to the data. Typically, a larger window size results in fewer samples but allows for a greater overview of the degradation process as more information is 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 $w = 10$. Regarding RUL-target

[6]The source code of the experiments can be found at `https://github.com/MariosKef/automated-rul` .
[7]We used *tsfresh*(0.17.0), *TPOT*(0.11.7), *scikit-learn*(0.24.1), *pandas*(1.1.5), *numpy*(1.19.5).

construction, we use the piece-wise linear approach, as we consider it to reflect more accurately a degradation of a turbofan engine [88], since these machines are designed to sustain multiple cycles and excessive loads and stress. Furthermore, this is still the most common approach in literature [146]. The values of the initial, constant, RUL were selected from [146], and the reflection point is selected as EoL/2 [94].

### Feature Extraction

In the feature extraction phase, we use the *tsfresh* pipeline (Time Series Feature extraction based on scalable hypothesis tests) [43], 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 features (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[8]. 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[9].

### 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 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* is used, which calculates the feature significance of a real-valued feature to a real-valued target as a $p$-value, using Kendall's tau. The algorithm has been applied with its default settings[10].

### Automatic Modeling

We approach this regression problem without using an a-priori selected pipeline, aiming to automatically identify the pipeline that gives the best cross-validated score on the training set.

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

[9]*tsfresh* has been applied with *EfficientFCParameters* as its extracted features list to reduce the computational cost (see `https://tsfresh.readthedocs.io/en/latest/text/feature_extraction_settings.html#for-the-advanced-how-do-i-set-the-parameters-for-all-kind-of-time-series`). The rest of the input parameters are left in their default settings. (see `https://tsfresh.readthedocs.io/en/latest/api/tsfresh.feature_extraction.html#module-tsfresh.feature_extraction.extraction`)

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

To tackle this, we used *TPOT* (Tree-based Pipeline Optimization Tool) [169, 97]. Based on genetic programming (GP) [126], *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 evolve, 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 [169, 238]. 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 per generated pipeline 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* (see Equation 4.2) in *TPOT* as its loss function to be optimized (here minimized) during the training process. The remaining hyperparameters used in *TPOT* were kept in their default setting[11]. In Table 4.2, we show all the hyperparameters used in this study and their values.

Table 4.2: 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 $S$ |

### Baseline

We also performed a baseline experiment to evaluate our main ideas and the pre-processing. Our baseline disregards the temporal aspect of the problem at hand. As a result, we do not perform any expanding window transformation 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 we used 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 steps remain the same (sensor selection, standardization of sensor values). The transformed dataset is fed again to an AutoML learning scheme. We use this baseline to show the benefits of using the expanding window and the statistical features together with the specific

[11] `http://epistasislab.github.io/TPOT/api/#regression`

RUL construction in an AutoML setting.

### 4.4.3 Experimental 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 *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 4.1). Regarding our baseline experiment, the inference on the test set is simply applied to the final time-step of its trajectory.

In Table 4.3 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 a sampling distribution of the means. We then performed a Wilcoxon rank sum test (Mann-Whitney U test)[12] with a significance level of $a = 0.01$ to check if the sampling distribution of the means of our proposed method is significantly different than the sampling distribution of the means of the baseline[13]. We selected this test to take into account the non-normality of the data and the independence between the samples. The resulting $p$-value is $p < 0.01$ signifying that the observed samples cannot come from the same distribution (rejecting thus, the null hypothesis)[14].

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 [195] or have achieved remarkable results [94, 260, 146, 143].

[12]In the original publication [112], a Wilcoxon *signed-rank* test was erroneously performed, instead of the Wilcoxon *rank sum* test. Since the samples are not paired (an assumption of the Wilcoxon signed-rank test) we corrected the statistical test in this chapter. We should note that despite the mistake in the original publication, the conclusions of the - corrected - statistical test remain the same.

[13]The null hypothesis is that the sampling distributions of the two methods are the same.

[14]The returned $p$-value is $p = 0.00$. In practice, this means that the $p$-value returned by the software is a very small float rendering it practically 0.00.

Regarding the application of classic machine learning algorithms on this problem, since there have not been many attempts, we report on all of those that to our knowledge exist (e.g., 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 4.4, in **bold**, we show the average results of our proposed method and the instances it outperforms. Below, we discuss these results distinguishing between classic machine learning and deep neural networks.

### Classic Machine Learning

The results show that the proposed method outperforms the multilayer perceptron (MLP), support vector regression (SVR), relevance vector regression (RVR) [195], LASSO regression, SVM and KNR regression [253] on all 4 datasets both in terms of the score function $S$ and the RMSE. Moreover, the proposed method outperforms the RF [253] algorithm in terms of both the score $S$ and RMSE on FD001 and FD002, of RMSE on FD003, and of score function $S$ on FD004. In addition, it outperforms the GB [253] algorithm on FD002 in terms of the score function $S$ and the RMSE, outperforms FD001 on the score function $S$ and achieves comparable results on its RMSE. Lastly, it can outperform ETR [253] on all datasets, except for FD004 in terms of RMSE, where it achieves a comparable result. 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%[15] (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).

### Deep Neural Networks

When compared to DNNs, our method outperforms the first CNN approach [195] on all cases except on FD004. When compared, however, to LSTM [94, 260, 146] and a recent CNN approach with 1D convolution [143] our algorithm is outperformed or comparable in terms of RMSE. In more detail, our proposed method outperforms the CNN [195] 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 proposed in [94]. In more detail, our method outperforms [94] 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.

[15]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).

Our method is also comparable to [94] 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 [260] on FD001 in terms of the RMSE. The proposed method is also outperformed by the other LSTMs [260, 146] and CNN [143] 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)[16]. The reason is that the usage of advanced LSTM (e.g., using unsupervised pre-training) and CNN with 1D convolution allow for learning highly nonlinear 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 AutoML in combination with extracting statistical features can outperform or achieve comparable results 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 nonlinear relationships that might describe the mapping between the time-steps and the RUL. What is more, we should note here that DNNs were not included in our algorithm search space. Furthermore, using neural architecture search (NAS) based methods can be useful. 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 chapter.

## 4.5 Discussions and Conclusions

In this chapter, we presented the first, to our knowledge, AutoML approach [97] for the estimation of the RUL of machinery. We investigated the usage of $TPOT$ ([169, 97]) in automatically selecting a pipeline for this problem and the usage of statistical embeddings of time-series in the pre-processing phase, using an expanding window transformation. The role of the AutoML was to take as input this *transformed* dataset and output a pipeline where the pre-processing, feature selection and modeling are all selected automatically.
We evaluated the proposed method on the widely used C-MAPSS dataset [198]. The gathered results show the benefits of using AutoML in combination with extracting statistical features (embeddings) and constructing the RUL in a piece-wise linear manner. In detail, the results indicate that such an approach can outperform or achieve comparable results compared to classic machine learning techniques (such as SVR, LASSO, SVM [195, 253]). However, when compared to deep architectures such as CNN and LSTM [94, 260, 146, 143], our method is able to outperform the first CNN approach on 3/4 of datasets. In general, it achieves a comparable performance or is outperformed by other deep learning baselines. This suggests that the combination

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

of statistical features and classic ML might not be sufficient to uncover the highly nonlinear relationship between the observed/measured values and the RUL. The proposed method also allows for a helpful direction towards which classic machine learning algorithms would be more beneficial, 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).
For future directions, we recommend the inclusion of neural networks in the AutoML approach, by means of NAS [60], as well as investigating effective dimensionality reduction techniques for the statistical embeddings or other effective representations that are able to retain the necessary degradation information. Finally, augmenting such approaches with a hyperparameter optimization wrapper is always vital to reduce the unwanted bias of the hyperparameter selection or add domain knowledge to the process.

Table 4.3: 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 | $\mathbf{383.9 \pm 7.6}$ | $\mathbf{15.9 \pm 0.15}$ | $52.7 \pm 25.4$ | $107610.9 \pm 64153.6$ | $41.8 \pm 5.4$ | $18.4 \pm 8.2$ |
| FD002 | $\mathbf{10567.8 \pm 748.1}$ | $\mathbf{28.2 \pm 0.45}$ | $81.7 \pm 11.5$ | $859641.7 \pm 999547.1$ | $44.7 \pm 12.1$ | $42.9 \pm 4.4$ |
| FD003 | $\mathbf{894.4 \pm 116.3}$ | $\mathbf{19.7 \pm 0.5}$ | $67.4 \pm 2.4$ | $5.32 \cdot 10^8 \pm 3.62 \cdot 10^8$ | $79.1 \pm 14.5$ | $19.5 \pm 10.6$ |
| FD004 | $\mathbf{26649.1 \pm 25617.4}$ | $\mathbf{33.7 \pm 1.3}$ | $86 \pm 8.4$ | $1.58 \cdot 10^8 \pm 1.60 \cdot 10^8$ | $60.8 \pm 15.6$ | $55.4 \pm 2.8$ |

Table 4.4: 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 [195] | **17972** | **37.56** | **$7.8028 \cdot 10^6$** | **80.03** | **17409** | **37.39** | **$5.6166 \cdot 10^6$** | **77.37** |
| | SVR [195] | **1381.2** | **20.96** | **58990** | **42** | **1598.3** | **21.05** | **371140** | **45.35** |
| | RVR [195] | **1502.9** | **23.8** | **17423** | **31.3** | **1431.6** | **22.37** | 26509 | **34.34** |
| | RF [253] | **479.75** | **17.91** | **70456.86** | **29.59** | 711.13 | **20.27** | **46567.63** | 31.12 |
| | LASSO [253] | **653.85** | **19.74** | **276923.89** | **37.13** | **1058.36** | **21.38** | **125297.19** | **40.70** |
| | SVM [253] | **7703.33** | **40.72** | **316483.31** | **52.99** | **22541.58** | **46.32** | **141122.19** | **59.96** |
| | KNR [253] | **729.32** | **20.46** | **450094.04** | **36.05** | **1030.29** | **22.59** | **234396.56** | **54.44** |
| | GB [253] | **474.01** | 15.67 | **87280.06** | **29.09** | 576.72 | 16.84 | 17817.92 | 29.01 |
| | ETR [253] | **1359.38** | **22.05** | **231030** | **33.01** | **1757.6** | **24.52** | **69771** | 32.42 |
| Deep Neural Networks | CNN [195] | **1286.7** | **18.45** | **13570** | **30.29** | **1596.2** | **19.82** | 7886.4 | 29.16 |
| | CNN [143] | 273.7 | 12.61 | 10412 | 19.61 | 284.1 | 12.64 | 12466 | 23.31 |
| | LSTM[94] | **388.68** | **16.74** | **10654** | **29.43** | 822.19 | 18.07 | 6370.6 | 28.4 |
| | LSTM[260] | 338 | **16.14** | 4450 | 24.49 | 852 | 16.18 | 5550 | 28.17 |
| | LSTM[146] | 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** |