Data-driven strategies for predictive maintenance: Lesson learned from an automotive use case

Data-Driven Strategies for Predictive Maintenance: Lesson Learned from an Automotive Use Case

Danilo Giordano[a,*], Flavio Giobergia[a], Eliana Pastor[a], Antonio La Macchia[a], Tania Cerquitelli[a], Elena Baralis[a], Marco Mellia[a], Davide Tricarico[b]

[a] *Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy danilo.giordano@polito.it, eliana.pastor@polito.it, flavio.giobergia@polito.it, antonio.lamacchia@studenti.polito.it, tania.cerquitelli@polito.it, elena.baralis@polito.it, marco.mellia@polito.it*

[b] *Punch Torino, Turin, Italy davide.tricarico@punchtorino.com*

Abstract

Predictive maintenance is an ever-growing topic of interest, spanning different fields and approaches. In the automotive domain, thanks to on-board sensors and the possibility to transmit collected data to the cloud, car manufacturers can deploy predictive maintenance solutions to prevent components malfunctioning and eventually recall to the service the vehicle before the customer experiences the failure. In this paper we present PREPIPE, a data-driven pipeline for predictive maintenance. Given the raw time series of signals recorded by the on-board engine control unit of diesel engines, we exploit PREPIPE to predict the clogging status of the oxygen sensor, a key component of the exhaust system to control combustion efficiency and pollutant emissions. In the design of PREPIPE we deeply investigate: (i) how to choose the best subset of signals to best capture the sensor status, (ii) how much data needs to be collected to make the most accurate prediction, (iii) how to transform the original time series into features suitable for state-of-art classifiers, (iv) how to select the most important features, (v) how to include historical features to predict the clogging status of the sensor. We thoroughly assess PREPIPE performance and compare it with state-of-art deep learning architectures. Our results show that PREPIPE cor-

*Corresponding author
Email address: danilo.giordano@polito.it (Danilo Giordano)

rectly identifies critical situations before the sensor reaches critical conditions. Furthermore, PREPIPE supports domain experts in optimizing the design of data-driven predictive maintenance pipelines with performance comparable to deep learning methodologies while keeping a degree of interpretability.
Keywords: predictive maintenance; data-driven; machine learning; time series; automotive

---

1. Introduction

Before introducing the Internet of Things paradigm, data in vehicles were collected and processed locally by the Engine Control Unit (ECU) for routine operations, management and monitoring. However, the limited storage and the little computational capabilities of the ECU made the collection and the analysis of a massive amount of data infeasible. Currently, thanks to connected vehicles, these limitations vanished. Indeed on-board data can be sent to a remote storage location for later analysis with better tools. Hence, automotive manufacturers have started leveraging the data collected by on-board systems to offer additional services and deploying predictive maintenance solutions. Predictive maintenance, or prognostics, aims to identify possible malfunctions ahead of time, allowing a prompt intervention before the actual problem arises. Both manufacturers and customers can benefit from this kind of prediction. The former can issue vehicle service calls only when needed and before irreversible damage occurs. The latter will not experience unexpected vehicle malfunctions and will perform maintenance operations only when needed. For these reasons, automotive companies are actively investing in effective predictive maintenance solutions.

This paper focuses on studying predictive maintenance solutions for the oxygen sensor (also known as lambda sensor). This sensor, placed on the exhaust system of combustion engines, measures the fraction of oxygen in the output gas. This information allows the ECU to optimally regulate the ratio of fuel and combustion air for the catalyst, reduce the emission of pollutants, and op-

timize the injection system's performance. Due to the imperfect burning of the combustion, the engine ejects some soot, which accumulates and clogs the oxygen sensor. As a result of clogging, slower and incorrect oxygen measurements cause a sub-optimal performance of the injection system and increase harmful emissions. Some engine operations can clean the oxygen sensor. These could be triggered periodically or when a pre-alarm status is identified. If not correctly handled, the oxygen sensor gets too clogged, and the ECU turns the check engine light on, forcing the driver to go to the service for costly maintenance operations. Hence, the early prediction of the pre-alarm status of the oxygen sensor clogging is fundamental to trigger the cleaning operations. Unfortunately, measuring or predicting the oxygen sensor status is a complex task due to the many factors that drive the soot accumulation process, including driving style, engine age, vehicle load, fuel quality, weather conditions, etc.

Here, we propose a generic data-driven approach and apply it to detect the clogging status of the oxygen sensor. We exploit a dataset collected by General Motors (GM[1]), in a test bench environment where a diesel engine runs for one hour while on-board and bench sensors collect data in the form of time series. With the help of an accurate procedure designed by domain experts, we get accurate labels for all the 388 experiments (Giobergia et al., 2018) that we use to design PREPIPE.

Based on the preliminary results presented in (Giobergia et al., 2018), in this paper we complete the design of PREPIPE (PREDICTIVE MAINTENANCE PIPELINE). It automatically takes the signals collected from on-board sensors and leverages a machine learning pipeline to predict the current status of the oxygen sensor. This research work provides the following main contributions.

- *In-depth analysis of the preprocessing steps.* We (i) evaluate and compare different methodologies to select the most important signals to model the status of the sensor, (ii) verify how much data and time are needed before

[1]GM is a leader in the application of automotive prognostics.

making a new reliable prediction, (iii) investigate different strategies to transform the monitored signals into features suitable for training state-of-art classifiers, (iv) select the best subset of features to model the status of the oxygen sensor, and (v) augment the model with previous observations to evaluate the possibility to improve the predictive maintenance capabilities of the framework.

- *Real-world data analysis application.* We thoroughly compare state-of-the-art classifiers ranging from decision trees to neural networks with different characteristics to process on-board data in the cloud with satisfactory results.

- *Time Dependence.* We check if the cumulative nature of the clogging phenomenon calls for methodologies that explicitly take into account the temporal sequence of the engine observations.

Our results show that accurate preprocessing is fundamental to achieve good predictive capability. Furthermore, we show how keeping track of the engine status for a long time is unnecessary, thus allowing a practical and scalable implementation even in a cloud environment. Finally, we compare our performance with state-of-art deep learning architectures to verify their applicability in our case study. We show that PREPIPE offers performance comparable to deep learning methodologies while offering domain experts an interpretable pipeline.

While we focus on a specific use case, we believe that our framework is general-purpose and adapted to other use cases. For this reason, we make available the code of PREPIPE as open-source at (Giordano et al., 2021b).

The rest of the paper is organized as follows. Sec. 2 compares our methodology with similar research activities, Sec. 3 defines our case study, the dataset, and the labeling procedure. Sec. 4 overviews PREPIPE framework while Sec. 5 gives an in-depth view of it. Sec. 6 describes the modeling and prediction activities and discusses the importance of the time sequence in the modeling phase. Sec. 7 presents the experimental results based on PREPIPE while Sec. 8

presents results based on state-of-art deep learning architectures. Finally, Sec. 9 presents the main takeaways and future directions.

2. Related Work

This section outlines related predictive maintenance works in the automotive domain, highlighting common challenges of such domain and the proposed solutions. We deepen the discussion on deep learning approaches, which recently are gaining momentum. Finally, we analyze specific studies facing the oxygen sensor diagnosis.

Predictive Maintenance in Automotive. The increasing capability to collect vehicles data has fostered many studies to monitor, detect, and define predict maintenance operations in the automotive industry. The authors of (Mesgarpour et al., 2013) present a complete overview of prognostics and health management in transportation and the automotive industry.

Due to ECU's limited memory, modest computational capability, and bandwidth constraints, automotive application multiple approaches tackle the problem by leveraging data dimensionality reduction techniques. Similarly to our proposed pipeline, some works focus on the feature selection step for machine learning techniques using approaches such as multiway partial least squares (Choi et al., 2008), common factor analysis (Jun et al., 2006), a combination of domain expertise and PCA (Shafi et al., 2018), wrapper feature selection, and filter method based on the Kolmogorov-Smirnov test (Prytz et al., 2015), or minimum redundancy maximum relevance algorithms (Giordano et al., 2021a). Differently from these works, here we tackle the problem of feature engineering by integrating and evaluating a wide range of signal selection, feature extraction, and feature selection approaches, generalizing the problem. Compared to previous works, we aim to offer interpretable results to the domain experts who double-check the process with their domain expertise and improve the understanding of the phenomenon under study.

To complete the predictive maintenance pipeline, we rely on well-established

machine learning algorithms commonly used in the predictive maintenance context (Carvalho et al., 2019).

Deep Learning Predictive Maintenance. Recently, deep learning models are getting popular for fault diagnosis and prognosis. Such shift is driven by the generally superior classification performance and their capability of handling high-dimensional data in predictive maintenance and health management scenarios (Ran et al., 2019; Khan & Yairi, 2018; Zhang et al., 2019). These approaches directly work on the raw input data without the need for any feature engineering.

Restricting to automotive applications, the authors of (Wolf et al., 2018) propose a data-driven deep learning diagnostic approach based on a combination of convolutional (CNN) and long short-term memory (LSTM) neural networks. They use ECU data to detect pre-ignition, i.e., ignitions before the spark plug fires. The authors in (Luo et al., 2019), instead, introduce a combination of the dual-tree complex wavelet transform, coupled again with CNN and LSTM, to monitor the health status of a vehicle suspension. Given the need for domain experts to understand both the prediction process and the phenomenon characteristics, we prefer to follow a well-designed predictive maintenance pipeline. We show it offers performance comparable with state-of-art deep learning architectures while satisfying the interpretability requirements.

Oxygen sensor diagnosis. The oxygen sensor plays a vital role in reducing exhaust emissions, and it is crucial to monitor and diagnose its status and detect and predict faults. To this extent, different works focus on the predictive maintenance of such key elements. For instance, authors of (Moser et al., 2014) propose a model-based solution and use the exhaust pressure pulsation to detect deterioration of oxygen sensor dynamics caused by sensor clogging. While the authors of (Ekinci & eniz Erturul, 2019) leverage a machine learning model to detect faults. They leverage data collected by a UEGO (universal exhaust-gas oxygen) sensor in a controlled environment to build a PCA methodology to identify the most relevant signals and a feed-forward neural network to detect if the oxygen sensor is faulty.

Unlike these works, we propose a data-driven approach that leverages readily available sensor data recorded by the on-board ECUs. We explore both supervised and unsupervised approaches to select the most suitable input signals to model the problem. A relevant distinction regards the target of the works. Rather than deterioration and fault detection, we focus on predicting a discrete status of the oxygen sensor in three health conditions. From the clean, the partially clogged to the fully clogged status, these three operating conditions provide a more in-depth view of the sensor diagnosis.

3. The oxygen sensor case study

The oxygen sensor is a device used to measure the proportion of oxygen in the exhaust gas of an internal combustion engine. This information is fundamental to lower the exhaust gas pollutants and optimize the performance of the injectors' fueling system and engine in general.

Due to the accumulation of the unburnt hydrocarbon soot contained in the exhaust gas, the oxygen sensor is subject to clogging. When the sensor is clogged, slower and incorrect oxygen measurements cause a sub-optimal combustion efficiency resulting in more harmful emissions released into the environment. Currently, the ECU can only diagnose when very slow oxygen measurements occur, i.e., when the oxygen sensor has reached a critical state, and its readings are unreliable. When this situation occurs, the ECU turns the check engine light on, and the driver has to go to the service for the required maintenance operations, i.e., a costly manual sensor cleaning operation.

While the clogging process is a known problem, its non-linear and non-monotone trend makes its prediction a complex task. Indeed, while it is well known that the clogging increases slowly over time due to the soot accumulation, driving conditions, fuel quality, and driving styles affect this process. For instance, sudden abrupt accelerations, vibrations, or specific engine operations like active regeneration (Xin, 2013) can suddenly clean the sensor by burning or detaching some soot from the oxygen sensor. Some of these operations can be triggered by the ECU to clean the sensor before it reaches a critical status.

As such, an early prediction of the oxygen sensor's clogging status is fundamental to run these specific engine operations and clean the sensor to avoid malfunctioning.

3.1. Our dataset

In this paper we face a data-driven approach to predict the oxygen sensor clogging. We employ a test bench in which an actual diesel engine equipped with the standard on-board and some additional bench sensors allows us to collect data. The testbed lets us emulate real driving conditions with different vehicle loads and conditions. In detail, in each experiment called *cycle*, the emulator follows a "driving cycle", i.e., a predefined sequence of gas pedal presses and releases coupled with different engine loads to reproduce different driving situations (e.g., urban, extra-urban, highway). For our experiments, we follow the driving cycle derived from the Real Driving Emissions (RDE) test procedures (Donateo & Giovinazzi, 2017). We focus on the RDE procedures as, currently, they are used to create the standard homologation cycles for testing particles and exhaust emissions in real traffic and environmental conditions. They are designed to represent as much as possible real driving situations.

We employ two different data loggers to monitor the engine under test, namely Program A and Program B. They have different characteristics, and we leverage them for similar – yet complementary – purposes. Program A monitors the engine during the entire cycle as in a real on-board scenario. It records a set X of 50 signals x_i related to on-board and bench sensors – see Appendix A for an overview of the signals' categories.

Each signal x_i is a time series where samples $x_i(t)$ are collected with a frequency of 1 Hz. Program B, instead, monitors only the final 5-minute of each cycle when a specific input sequence is imposed (described later) and provides more details about the engine and oxygen sensor behavior by collecting hundreds of bench signals at high frequency, i.e., 320 Hz. Guided by the domain experts, we use the data from Program B to measure the actual clogging status of the oxygen sensor and get thus a label for the entire cycle. Fig. 1 depicts the trace

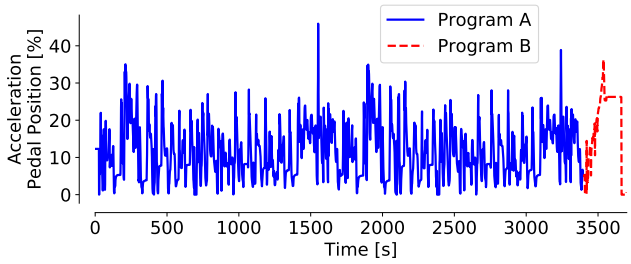


Figure 1: Acceleration pedal signal, recorded with both Program A (in blue) and Program B (in red).

of the gas pedal pressure during the entire cycle highlighting in red the part monitored by both programs when the specific maneuver that allows measuring the sensor clogging level. Each cycle last 62.5 (57.5+5.0) minutes. In total we obtain a dataset D with 388 cycles. Tab. 1 reports the main characteristics of these programs.

	Program A	**Program B**
Duration	3750 s	300 s
Sampling frequency	1 Hz	320 Hz
Number of signals	50	440
Number of cycles	388	388

Table 1: Program A and Program B characteristics.

Not all 50 signals recorded by Program A are useful to predict the clogging status of the oxygen sensor. As such, we perform a preliminary *a-priori* data selection procedure to select only those signals (i) that are available on-board, (ii) may capture helpful information of the clogging phenomenon under study, and (iii) are not redundant. Firstly, we remove all the test bench signals that are not available in vehicles. Secondly, we discard signals unrelated to the problem with the support of domain experts that properly consider the informativeness of the signals. Next, we discard signals with constant values (e.g., alarms) that carry no information. At last, we keep only one among possible pairs of strongly correlated signals (having a Pearson correlation close to 1). Finally, we remove the signal related to the oxygen sensor itself to avoid data leakage. Intuitively,

we remove it to avoid having any signal direct correlated with the addressed problem. As a result of this a-priori data selection procedure, we remain with the set $\hat{X}$ of 30 signals. More details about these are available in Appendix A.

3.2. Labeling procedure

Predicting the clogging state can be seen as a classification problem where the input consists of the engine data (as defined by the signals collected from the sensors), and the output is the current status of the oxygen sensor. With domain experts' help, we assign each cycle a label based on the data collected by Program B at the end of each cycle ends when the engine performs a short "cut-off" maneuver. It consists of quickly bringing the engine to maximum RPM, followed by a sudden release of the gas pedal (see last part of Fig. 1). This maneuver is fundamental to accurately measure the *Response Time*, i.e., the time needed for the oxygen level to complete the transition from the initial value (where little oxygen is present in the gas due to the high regime and combustion phase) to the $\frac{2}{3}$ of the final value where the percentage of oxygen shall correspond to the natural atmosphere level (since no combustion happens). Intuitively, a reliable oxygen sensor shall measure this transient time as short as possible. Clogging makes this transition longer. In a nutshell, the longer the Response Time, the more the oxygen sensor is clogged. Being the Response Time in the order of the second, the high sampling frequency of Program B is mandatory to get a correct estimation. Along with the domain experts, we define two Response Time thresholds, 1.3 s and 1.66 s, respectively. We then label the oxygen sensor status as:

- green: the oxygen sensor is clean if response time is shorter than 1.3 s;
- yellow: the oxygen sensor is partially clogged if response time is in the $[1.3, 1.6)$ s. This state corresponds to a *silent check*, i.e., a pre-warning, that the ECU may use to trigger possible cleaning operations;
- red: the oxygen sensor is fully clogged if the response time exceeds 1.6 s and the engine shall perform cleaning operations.

Notice that we define the oxygen sensor status based on the measurement done at the end of each cycle and then label the whole cycle. Given the duration of each cycle and of the soot accumulation process (which needs days to accumulate), the domain experts consider the labeling process consistent for the whole experiment duration.

Tab. 2 reports the number of available cycles for each class in our dataset D. Further details about Response Time and labeling procedure are available in (Giobergia et al., 2018). Notice that most labels fall in the Green and Yellow classes, while the Red class consists of only 15% of all cases. Such class imbalance is typical of predictive maintenance use cases, where the "faulty" class is much less frequent than the "regularly working" one. Note that the introduction of the yellow class is instrumental both to help the classifier in training the model, and to trigger some early cleaning operations by the ECU, e.g., the active regeneration.

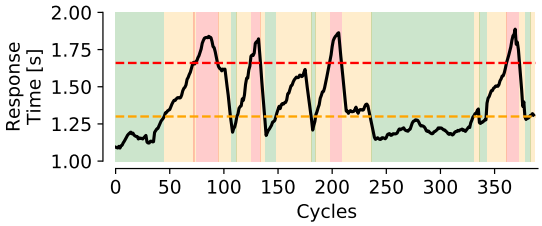


Figure 2: Cycle labeling based on the Response Time measure.

Class	Cardinality
Green	164
Yellow	163
Red	61

Table 2: Cycle cardinalities for the three clogging classes.

In Fig. 2 we report the Response Time measured at the end of each cycle along with the class label. Cycles are sorted by time. In general, the response time gradually increases over time due to soot accumulation, with labels that move from green to yellow to red. However, the Response Time trend is not monotonic. Occasionally it decreases for some cycles, e.g., from experiment 30 to 40, due to external conditions or drastically drops, e.g., on experiment 94 due to the action of the active regeneration operation that the ECU typically runs when the particulate filter is blocked. As it can be seen even in the test bench controlled environment, the clogging phenomenon depends on many factors, including particular engine maneuvers, regeneration operations, external condi-

tions (humidity, temperature, fuel quality), the time between cycles, etc. Hence, these unpredictable factors and the impossibility of measuring the response time on board call for ingenuity to design a proper predictive maintenance solution that triggers those cleaning operations before the sensor gets compromised.

3.3. Problem Definition

Given all signals recorded by the ECU, our goal is to identify whether the oxygen sensor is subject to clogging so that the car can run the required cleaning operation before the oxygen sensor reaches an unreliable state. As such, we could either tackle it as a remaining useful life (RUL) problem in which we predict how long it takes until the oxygen sensor is fully clogged or as a prediction of the current oxygen sensor status. Since we aim to forecast the state of the oxygen sensor, we discard the RUL approach. Secondly, we could formulate the problem either as a regression task in which we predict the response time or as a classification task in which we predict its discrete status. Since our main aim is to predict when the oxygen sensor status reaches a clogged situation, a continuous regression-based prediction is not required, and a classification task suffices. Existing lines of works in the automotive literature evaluate multi-targets (as in Last et al. (2010)) or assign multi-labels to detect multiple and simultaneous faults as in Vong et al. (2014)). We instead concern ourselves with a multi-class classification scenario, targeting to predict the status of the oxygen sensor in terms of three mutually exclusive classes. Lastly, we could either deploy a complete predictive maintenance pipeline with an in-depth feature extraction process or rely on the most recent deep learning solutions to build a model directly from the raw data. Since the carmaker is ultimately interested in better understanding the clogging phenomenon by discovering which signals and features are the most important to predict the oxygen sensor status, we focus on the first solution. We compare two state-of-art solutions with our pipeline to understand the possible benefits of recent deep learning solutions. Notice that the deep-learning solution would require either to equip the ECU with sufficient processing capabilities either transmit all the required signals to the cloud.

Figure 3: The PREPIPE predictive maintenance framework.

4. The predictive maintenance pipeline

We designed the PREPIPE early prediction pipeline for the oxygen sensor clogging represented in Fig. 3.

In the following, we summarize each step of the PREPIPE pipeline, while a detailed description is provided in Sec. 5.

Signal selection. The input to the pipeline is the set of signals recorded in each engine cycle by the on-board sensors. Given the large number of available signals, we exploit different supervised and unsupervised learning algorithms to select the best subset of signals describing the oxygen sensor status.

Windowing. We determine the correct size of the time window in which to observe the monitored variables. The window size should allow an accurate evaluation of the oxygen sensor status.

Feature extraction. Signals are recorded as time series representing the variable values during the cycle. Signals are transformed into *features* by means of different feature extraction strategies. These features allow us to represent characteristics of the time series that would not be visible in a sample by sample representation.

Feature selection. Several features represent each signal, some of them possibly redundant or uncorrelated with the target variable. We reduce the number of these features through a supervised feature selection stage.

Historicization. Each time window describes the current status of the oxygen sensor, and it does not include any historical information about previous windows and the status of the sensors. Hence, we add historical features related to the past cycles evaluating the benefit (if any) of including past observations.

Model training, tuning and validation. To model the clogging status we integrate four state-of-the-art classification algorithms i.e., Decision Trees (Breiman

et al., 1984), Random Forest (Breiman, 2001), SVM (Cortes & Vapnik, 1995) and neural networks (Bishop et al., 1995)). Given the cumulative nature of the clogging phenomenon, we use two validation strategies to determine whether the temporal order plays a crucial role in the prediction of the oxygen sensor status or each sensor status decision is independent.

At each step, we select the best option and the parameter setting through a *wrapping approach* (Blum & Langley, 1997) in which a classifier is used to identify the best choice by comparing the predictive performance with the variation of the step configurations. In a nutshell, we run the complete pipeline from the signal selection to the model training, tuning and validation, by sequentially optimizing one step at a time. Initially, we assign *default values* at each step to identify a baseline of the performance. Then, we locally optimize the parameters of one step per time following the sequence of steps in our learning process.

5. Pre-processing: from raw data to features

The initial steps of the processing pipeline address all the tasks required to prepare the data of a given cycle for the model training tuning and validation step.

5.1. Signal Selection

The first step consists of selecting the best subset of signals to feed the classifier.

This brings several advantages: (i) *improved data collection on the field* by reducing the costs required for the on-board hardware and the bandwidth needed for the data transmission to the centralized server, and (ii) a *more concise representation of each cycle* makes the entire problem easier for the classifiers, which avoids considering useless inputs.

For signal selection, we aim at discarding the whole signal, i.e., all samples $x_i(t)$. For this, we employ different unsupervised and supervised learning algorithms. Each produces a ranked list of signals, from the most to the least important one, that we use to decide which signal to include and discard. In the end, starting from the set $\hat{X}$ we aim to get the best possible subset $\bar{X}$.

Unsupervised approaches

Unsupervised signal selection algorithms find the best subset of signals by analyzing the hidden structure in unlabeled data (Solorio-Fernández et al., 2020). These algorithms exploit solutions such as correlation (Giobergia et al., 2018), similarity (Mitra et al., 2002) to reduce as much as possible data redundancy, or data transformation (Lu et al., 2007) to identify those signals that primarily represent the phenomenon under study. Here - we compare three approaches that we briefly describe:

Feature Similarity (FSFS). (Mitra et al., 2002) uses a metric called *maximal information compression index* to reduce redundancy in the dataset. This algorithm requires a single parameter k representing the desired reduction. For each value of k, the algorithm returns a different subset of signals. In a nutshell, it produces the optimal subset of signals whose amount of information is k time lower than the original complete dataset. We identify the best subset of signals by searching which value of k maximizes the *Representation Entropy* (Mitra et al., 2002). This metric represents how equally the information is distributed among the signals.

Correlation-Based Feature Selection (CORR-FS). (Giobergia et al., 2018) reduces the redundancy in the data too. Unlike the FSFS, this algorithm exploits Pearson's correlation coefficient to identify the best subset of signals.[2] This algorithm requires a single parameter r_{min} representing the threshold above which two signals are considered as strongly correlated. Given any pair of signals $x_i(t)$ and $x_j(t)$, we compute all correlations and iteratively remove those signals that on average are more correlated than r_{min}, keeping only one representative signal at each iteration. To find the best subset of signals, we run the algorithm with $0 \leq r_{min} \leq 1$ and identify the best value by using the knee point identification proposed by (Satopaa et al., 2011).

[2]We publicly release the implementation at Giordano et al. (2021b).

Principal Feature Analysis (PFA). (Lu et al., 2007) exploits an algorithm based on the *Principal Component Analysis* (PCA) (Wold et al., 1987) and the *k-means* clustering algorithm (Hartigan & Wong, 1979) to identify the subset of signals retaining most of the dataset information. This algorithm requires two parameters: p, the number of components used by the PCA to represent each signal x_i, and q, the desired number of clusters computed by k-means. The algorithm selects only one signal for each cluster; hence, q also represents the number of signals selected at the end of the selection process. We identify the best value of p by evaluating the knee point between the number of components used by the PCA and the *Cumulative Explained Variance* i.e., the total amount of dataset variability represented by those components. Then, we find the best value of q by optimizing the clustering quality metrics, i.e., optimizing the *SSE* or the *silhouette* index (Han et al., 2011) which report at the end of the clustering process how points within each cluster are cohesive compared with cluster separation.

Supervised approaches

Supervised signal selection differs from the above approaches since we use the classification algorithm to select the subset of the most important signals. In a nutshell, here, we exploit different classifiers, providing a different subset of the signals. The main advantage of these solutions is that they evaluate the combined predictive capabilities of the input variables and optimize the choice for each classifier. On the downside, it is much more time-consuming, given the need to build a complete pipeline.

To use these algorithms, we transform each signal $x_i \in \hat{X}$ into a set F_i of n features, such as $F_i \in \mathbb{R}^n$. We provide details about the transformation in the next section. Then, each algorithm gives us information about the importance of each feature separately, i.e., allowing us to rank signals. We use this information to reassemble the importance of each signal and use it to select the subset of the most important signals. Here we consider the following classifiers:

Random Forest (RF). (Genuer et al., 2008) exploits a ranking algorithm based on the capability of the random forest to highlight the importance of each feature via the *Feature Importance (FI)* index. This metric defines how much each feature contributes to the classification process at the end of the training process. To extract it, we build a model by using all cycles D and all features F_i derived from $x_i \in \hat{X}$. To then gauge the signal rather than single feature importance, we compute the *Signal Importance* (SI) as the sum of all the *Feature Importance* (FI) of the signal's features as follows:

$$SI(x_i) = \sum_{j=1}^{n} FI(F_i(j)).$$

Finally, we rank the signals according to SI and compute the Cumulative Signal Importance, i.e., the sum of all signal importance while increasing the number of signals considered. As before, we select the best subset of signals by using the knee point identification proposed by (Satopaa et al., 2011).

Random Forest - Recursive Feature Elimination (RF-RFE). (Diaz-Uriarte & de Andrés, 2005) is based again on the signal importance but recursively eliminates the least important signals considered during the training phase.

We start from a set of signals $\bar{X} = \hat{X}$. Then, we train a model by using all cycles and features F_i of all signals $x_i \in \bar{X}$. At the end of the training phase, we record the classification performance of this model. For each signal in $\bar{X}$ we compute the Signal Importance as described before, and discard the $n\%$ least important signals, i.e., $\bar{X} = \bar{X} - n\%\bar{X}$. We build a new model and compute performance and iterate until only a single signal is available in $\bar{X}$. To compare the performance of different signal subsets, we compute the difference in the performance of two specifically trained models, i.e., the *out-of-bag error*. In the end, we select the subset of signals having the lowest out-of-bag error.

SVM - Recursive Feature Elimination (SVM-RFE). (Rakotomamonjy, 2003) leverages the same recursively eliminating algorithm but exploits a Support Vector Machine (SVM) classifier. Similar to Random Forest, SVM returns at

the end of the learning process the importance of each feature utilizing the *feature weight*. For each signal, we then compute the $Signal\ Weight$ (SW) as the norm of the vector composed by the $Feature\ Weight$ (FW) of the signal's features as

$$W(x_i) = ||FW(F_i(j))||\ j \in 1, .., n.$$

We repeat the recursive elimination algorithm by discarding the $n\%$ least important signals according to the Signal Weight. Since SVM does not expose any performance metric at the end of the learning process, here we optimize the *F1-Score*. At the end of the process, we select the subset of signals having the highest F1-Score.

5.2. Windowing

Each signal $x_i(t)$ is a continuous time series. Here we consider the opportunity to split time into independent and not overlapping windows $w(k)$ of duration ΔT, at the end of which the classifier predicts the label. In a nutshell, given a signal $x_i \in \hat{X}$ and a sample $x_i(t)$, the sample is assigned to a time window $w(k)$ such as $w(k * \Delta T) \leq t < w((k+1) * \Delta T)$.

The rationale is that typically in predictive maintenance, one deals with a slow process (clogging in our case), so it is useless to process data sample by sample. Instead, it is better to observe the system's evolution for a given time and periodically decide. In addition, the overall evolution of the system is affected by external variables, e.g., different driving routes, styles, external conditions, etc., which affect the engine working points (cfr. Fig. 1). A key decision is how frequently one should decide, i.e., how big ΔT shall be. On the one hand, a frequent evaluation of the clogging status would speed up the prediction's time. On the other hand, shorter system monitoring may cause a drop in predictive performance due to unreliable information. Here we optimize ΔT to choose the optimal performance by using the complete loop of model training, tuning and validation step.

5.3. Feature Extraction

Given a window $w(k)$ and the set of selected signals $\bar{X}$, we perform different strategies to extract a set of features F_i. We consider well-established methods for time series, and propose a specific methodology specifically tailored to the characteristics of our problem.

Time Series Feature Extraction (TSFEL). (Barandas et al., 2020) extracts more than 60 features from the original time series, including the statistical, temporal and spectral characteristics.

VEST: Automatic Feature Engineering for Forecasting (VEST). (Cerqueira et al., 2021) employs several steps to extract features out of a time series data: it groups observations in batches, summarizes each batch with statistical characteristics, and then returns the most relevant features according to a ranking criteria.

Tsfresh. (Christ et al., 2018) offers 63 time series characterization methods, including continuous wavelet analysis, fast Fourier transform, time series length, mean, max, and median, etc., to extract up to 794 time series features out of each time series.

Ad-hoc. Guided by the rationale that the clogging process is very slow, and that we are willing to compute simple features that could be computed on board, we define an *ad-hoc* strategy that summarizes each signal x_i using statistics of samples belonging to a time window. In detail, we compute:

- *Mean*: as the clogging of the oxygen sensor may introduce an offset on a signal which is proportional to clogging;
- *Standard deviation*: as the clogging of the oxygen sensor may slow down the signal variability, also due to the ECU compensation;
- *Percentiles*: the percentiles summarize the cumulative distribution function (CDF) of the signal values over time. They allow the system to iden-

tify those phenomena that change the signal values distribution, possibly at the extreme part of the CDF.

Notice that by just considering the distribution of the signal values in a given time window, we loose the correlation over time. To then consider also the variability of signal over short time, we compute the same features also for the discrete derivative $x'_i(t) = x_i(t) - x_i(t-1)$ of each signal x_i. These features may reintroduce useful information on the time component, e.g., defining how fast signals change.

5.4. Feature Selection

After the feature transformation, each signal is represented by f features, possibly some of them redundant or useless. Using all of them to train a model, we risk the course of dimensionality, resulting in a low-quality model with poor predictive performance. Fundamental is the feature selection stage that selects the best subset of the features for model training.

Starting with a set F composed of all the signals' features,

$$F = \bigcup_{i \in \bar{X}} F(x(i)).$$

we rank features $f \in F$ by using a supervised learning algorithm producing a ranking R. This ranking allows us to understand the combined predictive capability of the features without using an exhaustive search. Then, we iterate through subsets $S(j)$ composed by the top-j features in the ranking R:

$$S(j) = \bigcup_{i=1}^{j} R(i) \quad j \in \{1, .., |R|\}$$

At last, we select subset $S(j)$ that produces the best performance for the model training and validation steps.

5.5. Historicization

Given the typical cumulative effect of the predictive maintenance phenomenon under study, the intuition suggests considering not only the information of the current window $w(k)$ but also of past windows $w(l)$, $l < k$, i.e., also consider

historical features. This historicization step evaluates this aspect by including the past features recorded in the previous time windows.

Given a window $w(k)$, represented by the set $S(j,k)$, we add the feature $S(j,k-h)$ up to h_0 previous observations.

$$S(j,k) = \bigcup_{h=0}^{h_0} S(j,k-h).$$

When $h_0 = 0$, only present features are considered. Notice that we include only the features of the previous windows but not the past window labels. Indeed, the introduction of predicted labels as input data would introduce possible misclassified labels polluting the data itself.

While including the historicization features does not impact the complexity of on-board computation, it increases the number of features to consider, eventually making the model training mode difficult and ultimately impact the model performance. Thus, for each historicization h_0, we run the Feature Selection step to consider the most important ones for model training.

6. Model Training, Tuning and Validation

In this step, we compare different classification algorithms to identify the best model to be used in practice. Here we consider off-the-shelf algorithms and perform a thorough model validation and hyperparameter selection. We consider the following algorithms: *Decision trees* (Breiman et al., 1984) and *Random Forest* (Breiman, 2001) for their usual high performance and interpretability features, while Support Vector Machines (SVM) (Cortes & Vapnik, 1995) and *Artificial neural networks* (Bishop et al., 1995) with a multilayer perceptron (MLP) architecture for their capability to deal with high-dimensional and non-linearly split data.

Since each classifier has several hyperparameters, we use an extensive grid search to find the combination that maximizes performance. For this, we build and assess the performance of thousands of models, trained and tested as follows.

6.1. Model Validation

Given the cumulative nature of the clogging phenomenon, we consider two specific validation techniques - the traditional *k-fold cross validation* (Kohavi et al., 1995) and a *time series cross validation* (Hart, 1994).

These techniques allow us to explore how the temporal evolution plays a relevant role in the clogging phenomenon.

k-fold cross validation is considered the best practice, especially when the use case has independent data instances. However, we deal with slowly deviating processes, which may introduce temporal correlation among cycles. k-fold random split of cycles may lead in having a cycle at time i in the test set and the cycles at time $i-1$ and $i+1$ in the training set. If cycles are not independent, this may cause a data leakage, resulting in overestimating the model performance and generating an incorrect model.

Time series cross validation is designed to overcome this problem. In this case, we consider a continuous window of data for training and the subsequent second window for testing (Hart, 1994). As time advances, and as long as new labeled data is available, either both windows shift forward, or an expanding window strategy is used to enlarge the training data window (adding the new labeled instance) while shifting the testing window forward as well. A new model is trained then on the new training window and validated consequently. As in the k-fold validation, the overall performance is computed by averaging over all experiments. This solution evaluates the performance by predicting a fixed-size window of future data instances (the validation window). As such, there are no direct means to evaluate whether the model can predict any further in the future. In our case, we do not have new labeled data once in deployment. Hence, our solution will not be practical if recent labeled data instances are needed to rebuild a good model.

As such, we consider both validation approaches and look for the best model that performs well in both cases. For this, we divide the available cycles D into two parts, namely $D1$ and $D2$, following the temporal order, i.e., cycles in $D1$ are recorded before cycles in $D2$ – see Appendix B for a complete overview on how

we split the cycles. Given a classifier and one combination of hyperparameters, we first train and test its performance using $D1$. We consider both k-fold cross validation and time series cross validation. Next, we perform a *hold-out validation* in which we train the model using all data in $D1$ and validate performance using instances in $D2$. Intuitively, the performance on $D2$ captures the model's ability to label the sensor status in future cycles. Finally, we choose the classifiers that perform the best in both cases.

Intuitively, since $D2$ contains cycles that happen in the future with respect to $D1$, we can evaluate if cycles can be considered independent or not. Due to possible data leakage among cycles in $D1$, a model performing well with the k-fold cross validation in $D1$ would have poor performance with $D2$. Similarly, if having recent labeled data were fundamental, a model performing well with time series cross validation in $D1$ will struggle with instances more in the future (i.e., the cycles in $D2$).

In our use case, we implement a 10-fold cross validation. For the time series cross validation, we consider a sliding window of step 3, with 100 cycles for training and 100 cycles for validation.

6.2. Performance metrics

To evaluate the performance, we rely on the per-class F1-Score, and the overall accuracy. Both are commonly accepted metrics to gauge performance in classification problems - see Appendix C for a complete definition of these metrics.

In this study, we are firstly interested in the best prediction performance of the clogged class, i.e., the red class, which will cause the car to underperform. Secondly, we are interested in the overall performance of the model. As such, in the following, we focus on the F1-Score of the red class and then the overall accuracy.

For each validation technique we have four independent quality metrics are available: (i) F1-Score$_{red}$ in $D1$, (ii) Accuracy in $D1$, (iii) F1-Score$_{red}$ in $D2$, and (iv) Accuracy in $D2$. Since we are interested in finding the classifier with

Classifier	Parameter	Values
Decision Trees	Impurity decrease	[0, 0.02] step 0.005
	Min samples leaf	[5, 35] step 5
	Min samples split	{2, 20}
	Split criterion	gini
	Max features	{auto, log2 , None}
	Max depth	{30, 50 , None}
Random Forest	Impurity decrease	[0, 0.02] step 0.005
	Min samples leaf	{1,5,10,15,20,25,30,35}
	Min samples split	{2, 20}
	Estimators	{10, 50, 100, 500, 1000, 1500}
	Split criterion	gini
	Max features	{auto, log2 , None}
	Max depth	{30, 50 , None}
	Bootstrap	{False, True}
SVM	Kernel	{linear, poly, rbf}
	C	$[10^{-3}, 10^{3}]$ step 100 in a log scale
	γ	$[10^{-3}, 10^{3}]$ step 100 in a log scale
MLP	Input layer	$\|F\|$
	1^{st} hidden layer	{2,5,11,17,19,23,27,40,46,54}
	2^{nd} hidden layer	{2,5,11,17,19,23,27,40,46,54}
	Output layer	3
	Activation	{logistic, tanh}
	Seed	100 random values
	Solver	Adam
	Tolerance	10^{-4}

Table 3: Grid Search hyperparameter configuration.

good performance both in $D1$ and $D2$, we summarize each metric using the *harmonic mean*. Intuitively, if the validation in $D1$ has similar performance with respect to $D2$, the harmonic mean will take a value close to the single performance values. Otherwise, since the smallest value polarizes the harmonic mean, the performance will drop. This validation process allows us to solve the problems raised in Sec. 6.1.

6.3. Classifier Hyperparameter Optimization

We rely on a wrapping approach in which a classification algorithm is used to evaluate the performance of each optimization step. For each classifier, we compute the best performance through a grid search approach. In detail, for the Decision Tree and Random Forest, we cover a wide range of possible hyperparameter configurations. Instead, given the high number of hyperparameters

and lengthy training time of the SVM and MLP, we reduce the time needed to explore the hyperparameters based on (Hsu et al., 2003) for the SVM and on (Huang, 2003; Stathakis, 2009) for MLP. For MLP, we set the number of neurons in the input layer according to the number of features $|F|$, and the output layer as the number of classes. Tab. 3 details the ranges we used for each hyperparameter and algorithm.

To speed up the hyperparameter tuning process, we exploit a parallel computing system that allows us to train and test thousands of models in parallel. We rely on Python3.7, Pandas, and NumPy libraries for data manipulation and the Scikit-learn machine learning library for the implementation. Code is available at (Giordano et al., 2021b).

7. Pipeline Results

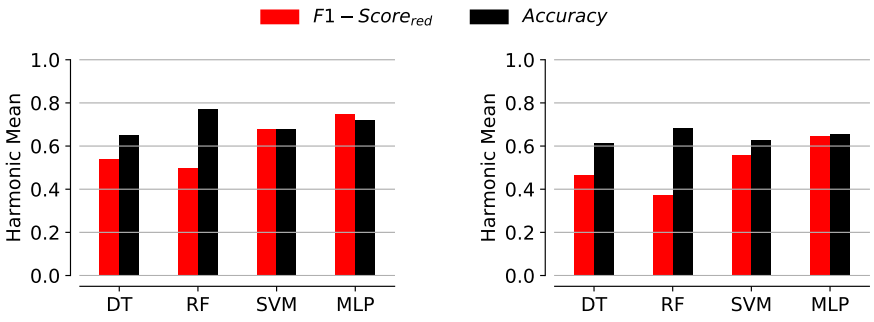
We have implemented the proposed pipeline and thoroughly tested it using the data collected in the bench environment described in Sec. 3.1. In this section, we present experimental results. We start from assigning to each processing step a *default value* and derive the *baseline* performance of an not-optimized pipeline. Then, we run the optimization steps to identify the best configuration of each one. In the end, we summarize the contribution given by each step to the performance improvement.

7.1. Baseline: the not-optimized pipeline

For the *signal selection*, we use the complete set of 30 signals that we obtain running only the manual a-priori data cleaning. For the *windowing* step, we map each cycle into one window. As *features extraction*, we consider the *ad-hoc*

Step	**Default Value**
Signal Selection	30 signals
Windowing	1 window per cycle
Feature Extraction	ad-hoc
Feature Selection	all features
Historicization	only current window

Table 4: Default value for each optimized step.



(a) 10-fold cross validation

(b) Time series cross validation

Figure 4: Performance of the baseline with no optimization.

approach with mean, standard deviation, and 9 deciles from the 10^{th} up to the 90^{th} percentile. In total, we have 11 features for the original signal and 11 more derived from the discrete derivative. Next, for the *feature selection*, we keep all the features, i.e., we do not run any selection. Finally, for the *historicization* we use $h_0 = 0$, so we use only current features. Tab. 4 summarizes the default value for each step.

Armed with this configuration at the input, we run the hyperparameter optimization step for each classifier. Recalling that in $D1$ we perform either 10-fold cross validation or time series cross validation, while in $D2$ we perform the hold-out validation in both cases, here we report the performance of the best model considering the harmonic mean of the $F1-Score_{red}$ of the red class and the overall accuracy. In detail, Fig. 4 shows results when we use the 10-fold cross validation, while Fig. 4b reports the case when we use the time series cross validation.

Firstly, we observe that the performance of 10-fold cross validation is higher than the one observed for the time series cross validation. All models indeed perform 5-10% consistently better, both when focusing on the red class and overall accuracy.

Recalling that the performance considers both the validation strategies in $D1$ and $D2$, this result suggests that both methodologies do not suffer from the

	Algorithm	Input Parameter	Optimization	# Selected Signals
Unsupervised	CORR-FS	r_{min}	Knee Point	13
	FSFS	k	Representation Entropy	5
	PFA-SSE	p, q	Knee Point	12
	PFA-Silhouette	p, q	Silhouette	10
Supervised	RF	RF-Config	Knee Point	9
	RFE-RF	RF-Config	OOB Error	2
	RFE-SVM	SVM-Config	$F1 - Score_{red}$	4

Table 5: Signal Selection.

temporal relationship among experiments.

Secondly, both SVM and MLP offer relatively higher performance than simple and interpretable models such as DT and RF. In more detail, both the DT and RF exhibit higher accuracy than F1-Score, hinting that both tend to have problems with the class imbalance (cfr. Tab. 2). Intuitively, they tend to optimize the majority of correct answers at the cost of giving the wrong classification for the minority class (the red class in our case). SVM and MLP suffer less from this problem.

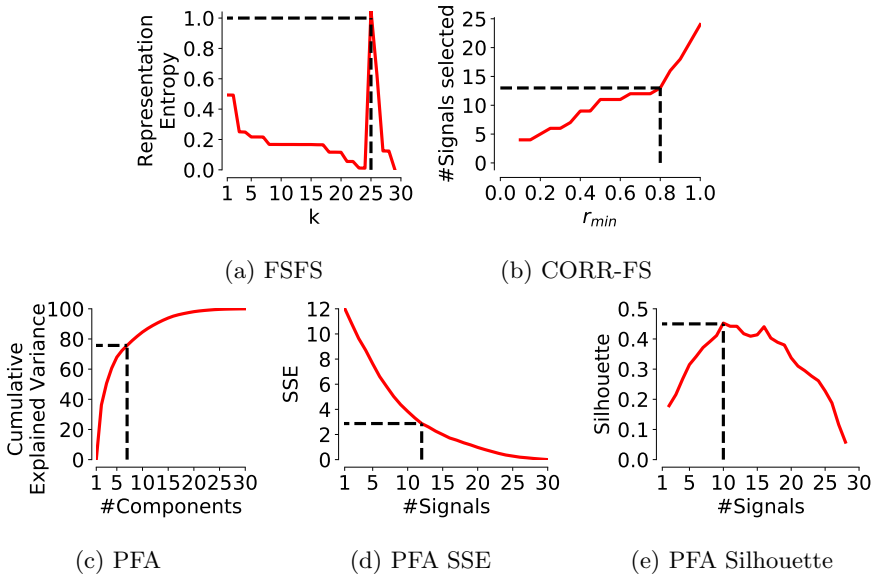
7.2. Impact of Signal Selection

Let us explore if signal selection brings benefits to the classifier performance. Here, we evaluate the different unsupervised and supervised approaches to identify the best subset of signals to model the clogging status of the oxygen sensor. In Tab. 5 we summarize the algorithm, the parameters, optimization criteria, and the number of selected signals at the end of the process. Notice that each algorithm can select a different subset of signals. Appendix D presents a complete overview of the signals selected by each algorithm and their nature.

Unsupervised approaches

We can exploit the full dataset D for the unsupervised algorithms since we do not need to run the whole pipeline with these algorithms. In the following, we describe how we proceed with each algorithm.

Feature Similarity (FSFS). We rely on the official implementation of the algorithm proposed in (Mitra, (accessed June 4th, 2020)). Firstly, we concatenate



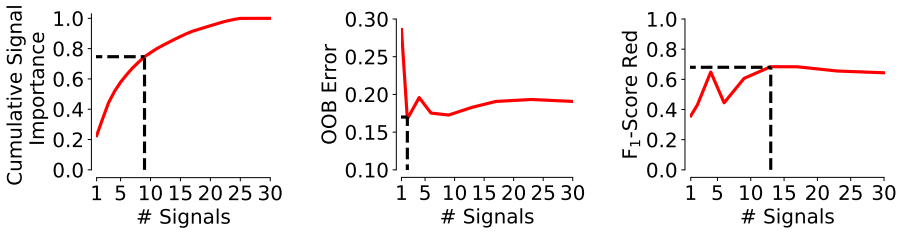
(a) FSFS (b) CORR-FS

(c) PFA (d) PFA SSE (e) PFA Silhouette

Figure 5: Identification of the best number of signals with unsupervised algorithms.

the samples of each signal from all cycles following the temporal order. Notice that this concatenation is instrumental in running the feature similarity process and that different orders always return consistent results. Then, we run the selection algorithm by varying the parameter k. For each value of k we extract the dataset composed by the selected signals and we compute the *Representation Entropy* as suggested in (Mitra et al., 2002). Fig. 5a reports the Representation Entropy value while increasing the parameter k. From this, we identify the maximum *Representation Entropy* when $k = 25$ hence where 5 signals are selected.

Correlation-Based Feature Selection (CORR-FS). For the correlation analysis, we evaluate a revised version of the algorithm presented in (Giobergia et al., 2018). In particular, at each step, we select among the remaining signals, the one having the highest sum of the *squared correlation coefficients* evaluated over all the signals available at the beginning of the algorithm. This modification allows us to reduce signal redundancy while keeping as much information as possible of the overall dataset. We automatically choose the best r_{min} based on



(a) RF (b) RFE-RF (c) RFE-SVM

Figure 6: Identification of the best number of signals with unsupervised algorithms.

the knee point identification. Fig. 5b reports the number of signals selected for different values of r_{min} which suggests $r_{min} = 0.8$, corresponding to 13 signals.

Principal Feature Analysis (PFA). Similar to the FSFS algorithm, we concatenate all signal cycles following the temporal order to obtain a single time series for each signal. Next, we normalize each concatenation using a z-score methodology, obtaining a zero mean representation. Finally, we run the PFA algorithm to identify the best number of components p representing the dataset. Fig. 5c reports the Cumulative Explained Variance while increasing the number of components. We identify 6 components that represent most of the dataset information by looking at the knee point. As the last step, we run the k-means clustering algorithm to select the subset of signals. We choose k by optimizing either the SSE or Silhouette scores. Fig. 5d and Fig. 5e reports the trend of the SSE and Silhouette scores, respectively. We find two suggested signal subsets composed of 12 and 10 signals, respectively.

Supervised approaches

We now run the signal selection using the supervised approaches.

Random Forest. (RF). We train a Random Forest configured with hyperparameters as suggested in (Genuer et al., 2008). We train it using the whole dataset D and all 22 features extracted from all 30 signals. At the end of the training, we compute the *Signal Importance* (SI) of each signal. Fig. 6a reports the signals ordered by their SI and the value of the Cumulative Signal Importance.

The knee point identification suggests selecting the first 9 signals as a possible subset of signals.

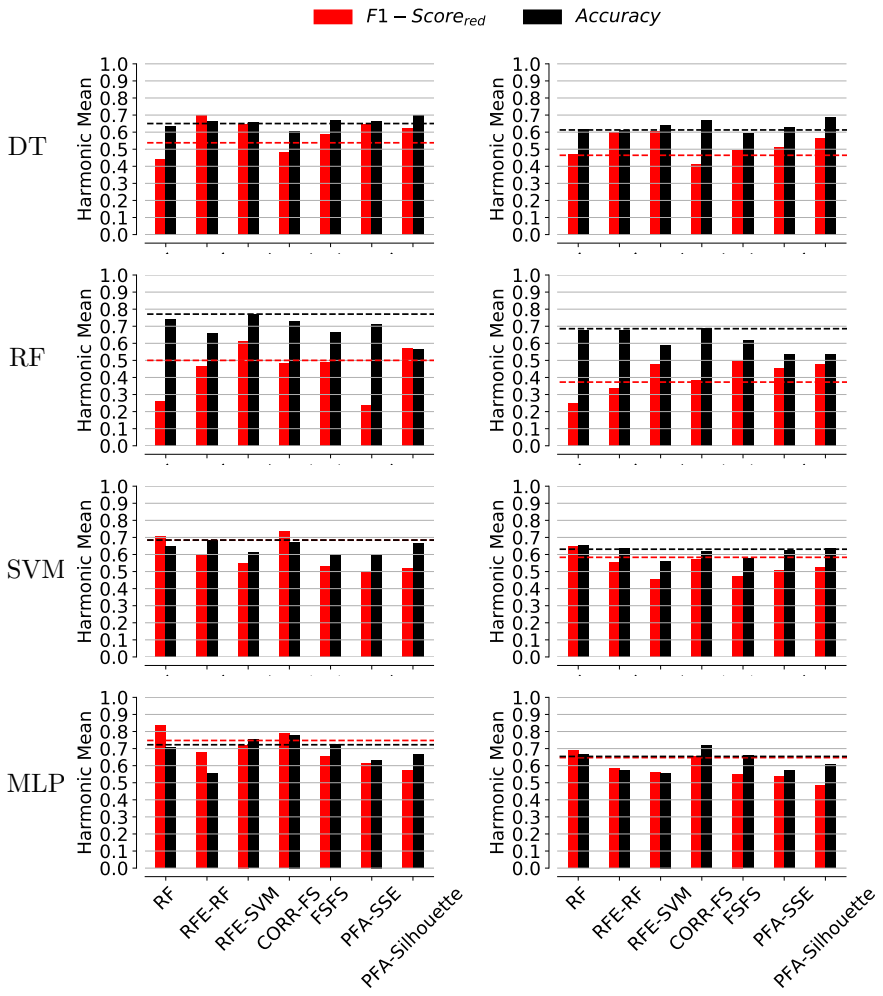
Random Forest - Recursive Feature Elimination - RFR-RF. This is an iteratively eliminating algorithm. We train a new Random Forest at each iteration by using all dataset D and all features from the signals available in the current set (initially all of them). The RF uses default hyperparameters as in Tab. 4, except for the number of estimators that we set to 2 000 as suggested in (Diaz-Uriarte & de Andrés, 2005). At each iteration, we discard the 20% least important signals according to their SI. Fig. 6b reports the value of the out-of-bag (OOB) error for the number of remaining signals. Here we select the subset having the lowest error, i.e., a selected subset of just 2 signals.

SVM - Recursive Feature Elimination - RFE-SVM. Similar to the previous algorithm, we remove 20% of the least important signals according to the signal weight at each iteration. Given the complexity of tuning the SVM hyperparameters, we evaluate 10 different hyperparameter configurations at each iteration. In detail, we use a linear kernel and equally sample the C space with a log-scale. [3] At each iteration, we evaluate the classification performance of each model with a standard 10-fold cross validation by using all experiments in D and the features derived from the remaining signals. Fig. 6c reports the trend of the best F1-Score of the red class with a different number of signals. The performance is maximized when a subset of 4 signals is considered.

Benefits of Signal Selection

We now observe the impact of the signal selection stage. For this, we run the rest of the pipeline with default values but with signals selected by each algorithm. Fig. 7 reports the harmonic F1-Score of the red class (red bar) and the harmonic accuracy (black bar) of the best hyperparameter configuration of all classifiers for each signal selection algorithm. Dashed line report the best

[3] Feature' weight in the Scikit-learn implementation is available only in the case of the linear kernel.



(a) 10-Fold Cross Validation (b) Time Series Cross Validation

Figure 7: Performance with the different subset of signals due to signal selection algorithms.

harmonic F1-Score of the red class and harmonic accuracy of the baseline model, which does not implement any signal selection.

We observe a very noisy picture regarding the signal selection, with some algorithms that benefit from it and others that worsen their performance. While DT and RF solutions tend to suffer from the lack of information caused by the removed signals, SVM and MLP show some sizeable benefits (e.g., with RF

and CORR-FS signal selection policies). Overall, RF and CORR-FS algorithms tend to show the most remarkable improvement - with the latter having more stable performance, i.e., similar values for the harmonic accuracy and F1-Score of the red class. Recalling that we are interested in increasing the prediction performance on the red class while keeping good performance overall, we can safely select the subset of CORR-FS selection.

Comparing the performance among validation strategies, 10-fold cross validation (Fig. 7a) shows higher performance than implementing time series cross validation (Fig. 7b). To further investigate this aspect, we compare the F1-score of the red class for all the 20 thousand models built while performing the grid search – with the CORR-FS signals. For each of these models, we compare the 10-fold cross performance in $D1$ with the hold-out performance in $D2$. Not show here for the sake of brevity, the largest fraction of the models show similar and consistent performance in $D1$ and $D2$, with a F1-score of the red class equal centered around 0.76.

This result is consistent with the previous observation for the baseline case, strengthening the observation that experiments can be considered independent, and considering the time evolution does not bring any benefit.

At last, considering classification algorithms, the MLP classifier confirms the best performance almost with all subset of signals in both validation techniques, with typically much better performance on the F1-Score of the red class. As such, from now on, we compute the performance only for the MLP classification algorithm and by using in $D1$ only the 10-fold cross validation technique.

7.3. Windowing

We now investigate the impact of different windowing policies, i.e., the ΔT parameter. For this, we divide each cycle (approximately 60 minutes long) into independent time windows having different duration ΔT: from 60 minutes (1 window per cycle) down to 2 minutes (30 windows per cycle). We label each window as the cycle it belongs to (assuming the clogging status is consistent through all the 60 min of the cycle).

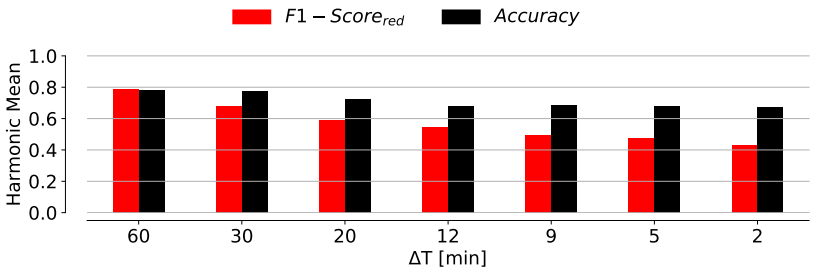


Figure 8: Performance improvement due to windowing.

Fig. 8 reports the MLP classifier performance using, as usual, the harmonic mean F1-Score of the red class and the harmonic mean of accuracy versus the number of cycles. A clear trend emerges: the shorter the cycle, the worst the performance. This trend is particularly true for the F1-Score of the red class, whose performance degrades quite sizeably when we extract features using shorter time windows.

In our specific use case, the best choice is to monitor the clogging status every 60 minutes, hinting that the clogging phenomenon is better observed on long time scales rather than frequently. This result is also beneficial in deployment. Indeed the ECU can reduce the amount of data to transmit, and the cloud would have fewer classification decisions to take per unit of time.

7.4. Feature Extraction

Next, we move to the feature extraction step. Here, starting from the 13 signals collected with a $\Delta T = 60min$, we extract one set of features for *tsfresh*; a second set for *VEST*; 4 sets for *TSFEL*; one set for *Ad-hoc*. For TSFEL we consider i) *All* the possible features; ii) *All-corr* all those not strongly correlated features; iii) *Statistical* features only; and iv) *Temporal* features only. Tab. 6 reports the number of features extracted by each strategy. Since the tsfresh library returns more than 10 000 features, we discard it given the small number of experiments we have which would make the model training cumbersome. Fig. 9 summarizes the performance obtained by each strategy - with a MLP classifier after the grid search. Interestingly, the variety of features exposed by the Vest

and the TSFEL packages does not help to describe the clogging phenomenon. The higher the number of features, the lower the performance with respect to the Ad-hoc features, which consistently outperform other strategies. We consider thus the Ad-hoc feature strategy in the following.

	Approach	#Features
	Ad-hoc	286
	Tsfresh	10231
	VEST	640
TSFEL	All	5070
TSFEL	All-corr	3964
TSFEL	Statistical	468
TSFEL	Temporal	234

Table 6: Number of features per feature extraction strategy.

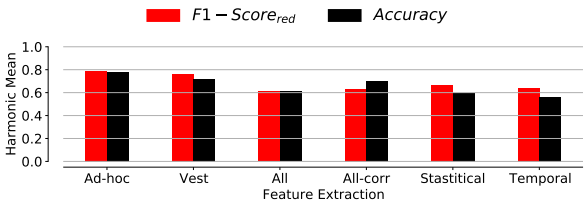


Figure 9: Performance per feature extraction strategy.

7.5. Feature Selection

Let us move to the feature selection step. Here we exploit the RF algorithm to rank the features according to their Feature Importance (FI). As before, we compare the performance of different feature combinations by running the training tuning and validation step for each model. Given the large number of training and testing operations, this step results in a very CPU-expensive phase. Specifically, we rank features according to their FI. Let $R(j)$ the $j-th$ ranked feature. We start with an initial subset $S(1) = R(1)$ comprising the most important feature only. Then we incrementally add to the current subsets $S(j)$ one feature at a time, i.e., $S(j+1) = S(j) \cup R(j+1)$. Then, we train, tune and validate the model using all features in $S(j)$.

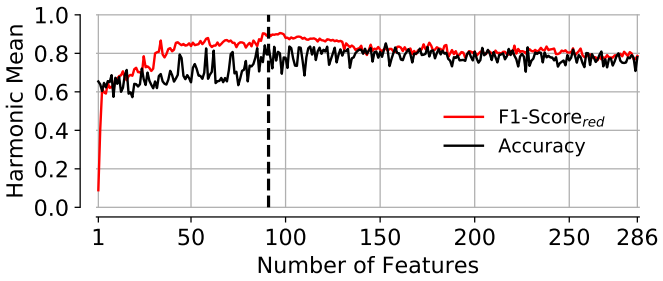


Figure 10: Performance improvement due to Feature Selection.

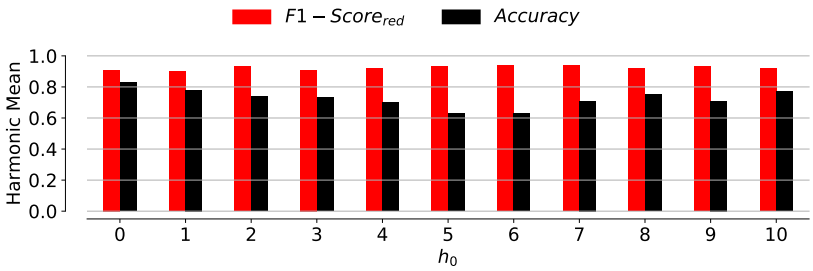


Figure 11: Performance improvement brought by historicization.

Fig. 10 reports the performance achieved by the best MLP when fed with the subset $S(j)$. Initially, adding more features improves the performance - especially for the F1-Score of the red class. With 85-95 features, we obtain the best performance on the red class. After this, enlarging $S(j)$ causes a significant decrease in the F1-Score of the red class, while the accuracy tends to be less affected. As such, we select the value of $j \in [85-95]$ that maximizes the accuracy. We select 91 features out of the initial 286. This reduction exemplifies the curse of dimensionality. Furthermore, by reducing the features, we simplify the model training which has to explore a smaller space.

Notice that the feature selection step improves the performance quite significantly w.r.t. the baseline. For example, the F1-Score for the red class is now up to 0.9 from the 0.74, and accuracy tops 0.8 from 0.71. This improvement also includes the benefit of the signal selection step.

7.6. Historicization

Finally, we evaluate the impact of including past information. For this, we enrich each time window $w(k)$ with the features of previous time windows $S(j, k-h_0)$. We next run the feature selection step and, for each value of $h_0 \in [0, h_{max}]$, we train, tune and validate a MLP model to find the best possible performance.

Fig. 11 reports results. We observe minor changes in the F1-Score when increasing h_0 while the accuracy drops. In detail, we observe the best F1-Score when $h_0 = 7$ with an increment of 3%; however, the accuracy drops by -12%,

Step	F1 − Score			Accuracy
	Green	Yellow	Red	
Original	0.78	0.62	0.75	0.72
Signal Selection	0.82	0.72	0.79	0.78
Windowing	0.82	0.72	0.79	0.78
Feature Extraction	0.82	0.72	0.79	0.78
Feature Selection	0.83	0.81	0.91	0.83
Historicization	0.83	0.81	0.91	0.83

Table 7: Wrap-up of the best performance per optimization step.

while the number of features rises to 436. With larger h_0, the accuracy keeps dropping, eventually due to the difficulties of the model to handle the increased number of features that past windows add as input. In a nutshell, in our use case, historicization brings no benefits.

7.7. Wrap-up

To wrap-up the contribution of each step in the optimization pipeline, Tab. 7 summarizes results obtained by optimizing each step. We consider the MLP and the k-fold cross validation only. For completeness, we report the harmonic mean of the F1-Score of the green and yellow classes too. In our use case, the signal selection and feature selection are the two steps that significantly improve the F1-Score of the red class - our optimization target. Due to this, accuracy improves as well. Interestingly, the performance of the green and yellow classes (not specifically targeted) remains stable and sizeably improves for green and yellow classes, respectively. This secondary effect may help the ECU in the identification of the clogging status by triggering a *silent check* highlighting the need for future cleaning operations. Furthermore, we observe that the optimization also eases the MLP hyperparameter tuning as discussed in Appendix E.

8. Comparison with deep learning methodologies

In the last decade, deep learning (DL) methodologies have gained momentum, thanks to the increase in computing capabilities and data availability, and have led to breakthroughs in many machine learning tasks (LeCun et al., 2015).

Indeed, deep learning methodologies have been helpful in several fields such as image classification (Rawat & Wang, 2017; Ciregan et al., 2012), time-series prediction (Fawaz et al., 2019), and prognostics as well (Wang et al., 2020; Li et al., 2018). Such increase in popularity is driven by the capability of deep learning solutions to abstract the data without complex feature engineering (LeCun et al., 2015), and their good performance, e.g., high accuracy in classification problems.

Here, starting from the set of 34 signals[4] after the domain experts' signal selection, we apply a light preprocessing step to normalize the signals using a z-score normalization process and split our data following two alternative approaches:

- *Training on the whole cycles (Whole)*: we use each of the 388 cycles as a separate input for the model, i.e., we train the model using the entire cycle at each step.

- *Time windows (Windowing)*: as in Sec. 5.2, we divide each experiment into independent time windows, setting $\Delta T = 100$ seconds, i.e., 14 356 inputs. Then, we feed each window separately to the model. Each window is labeled with the same cycle label it belongs to (assuming the clogging status is consistent through all the 60 min of the cycle).

Finally, we validate the pipeline following the 10-fold cross validation in $D1$ and the hold-out technique in $D2$. In the case of *windowing*, we avoid data leakage by using all the windows of the same cycle either for training or for validation. Then, for each cycle, we decide the label following a majority voting scheme on its windows.

[4]This includes the 30 signals that have been retained after the domain-driven selection, plus an additional 4 that were removed in the proposed pipeline because correlated with a coefficient ≈ 1 with other signals since the goal of these experiments is to assess how well DL methodologies fare with as little feature engineering/selection as possible.

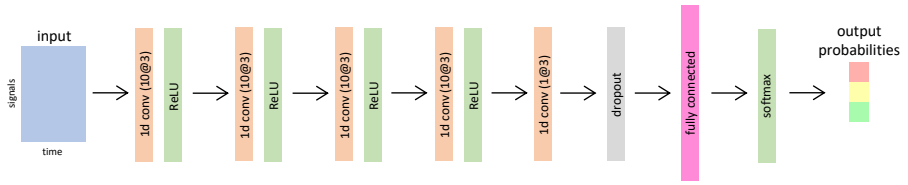


Figure 12: Convolutional neural network architecture based on (Li et al., 2018).

Classifier	Parameter	Values
CNN	# of conv layers	5
	kernel size (all conv layers)	3
	stride (all conv layers)	1
	output channels (conv layers 1-4)	10
	output channels (conv layer 5)	1
	dropout probability	0.5
LSTM	hidden state dimensionality	32
	# of recurrent layers	1

Table 8: Configuration used for the proposed deep learning architectures.

8.1. Convolutional Neural Network

In the prognostics field, authors in (Li et al., 2018) proposed to use a Convolutional Neural Network (CNN) architecture to approach a remaining useful life (RUL) task for a turbofan engine degradation problem. Here we use it as a first state-of-art architecture. Fig. 12 reports the complete DL architecture. Tab. 8 highlights the main hyperparameters used for the training of the model. Since we tackle a classification problem instead of a regression one, we insert a fully connected network with a softmax activation function as the last layer.

8.2. Long Short-Term Memory

For the Recurrent neural networks (RNN), we use a second state-of-art architecture, i.e., a bidirectional long short-term memory (LSTM) (Wang et al., 2018), which introduces a gating mechanism for retaining and discarding information. The bidirectionality of the LSTM implies that two LSTMs are trained simultaneously, using the signal in positive and negative time directions (Wang et al., 2018). This has been shown to provide better results in both regression and classification problems. Tab. 8 lists the hyperparameters used for the

Architecture	Preprocess	F1-Score			Accuracy
		Green	Yellow	Red	
CNN	Whole	0.792	0.571	0.587	0.680
CNN	Windowing	0.846	0.784	0.562	0.791
LSTM	Whole	0.775	0.526	0.360	0.654
LSTM	Windowing	0.861	0.812	0.817	0.836

Table 9: Wrap-up of the performance based on preprocessing methodology and deep learning architecture.

proposed LSTM model.

8.3. Results and discussion

Tab. 9 reports the results for each DL architecture and preprocessing approach. As in Tab. 7, we report the results in terms of the harmonic mean of the 10-fold cross validation and the hold-out validation techniques. Consider the CNN architecture first, in which both preprocessing approaches do not provide satisfying results on the red class. Interestingly, although we target the red class, we achieve the worst performance in it. Instead, on the LSTM architecture, we see how lightweight preprocessing plays a fundamental role, with the worst performance given by the *whole* approach. We believe this is due to the length of the signals (more than 3700 samples) raising the backpropagation problem. On the other hand, with the *windowing* approach, we obtain the best performance.

Comparing the best deep learning performance with the best one of our pipeline, we can see how only the bidirectional LSTM, with *windowing*, gets comparable results. The reason for this is the limited size of our dataset, which reduces the deep learning capability to extract general features. With additional data and with a more thorough fine-tuning, one may achieve better performance.

In a nutshell, while deep learning solutions may be considered, our pipeline reaches comparable, if not better, results. In addition, it allows us to keep a degree of interpretability required to provide feedback to domain experts. Moreover, the feature extraction process limits the amount of data the classifier needs, enabling us to deploy the classification process in the cloud, with only

the feature extraction performed on-board. On the contrary, deep learning approaches would require transferring much more data to the cloud, resulting in an unfeasible solution.

9. Discussion and Conclusions

We presented PREPIPE, a data-driven framework to perform predictive maintenance in a case study for the automotive field. In particular, we exploited our pipeline to predict the clogging status of the oxygen sensor in the automotive sector. Starting from a large number of time series reporting the car sensors' data, we extensively evaluate each preprocessing step to optimize the predictive performance of the framework. Our results show how domain experts can take advantage of our framework to select the best subset of signals and features to predict the system's status under analysis.

While the focus of this work was on a specific case study, the lessons we learned can be generalized to other real cases facing similar industrial problems. Besides the optimal configuration of the data-driven analytics pipeline, the proposed framework addressed the following relevant research issues.

- *Importance of signal and feature selection.* Interpretable results of the signal and feature selection algorithm (e.g., to check if the essential features for the classifier are identified) allow the domain experts to better understand the critical aspects of the clogging phenomenon by identifying the most important signals describing it. Furthermore, the interpretability of the process allows the manufacturer to understand how the data-driven algorithm works and trust its predictions.

  The feature selection algorithm narrows down the amount of data to be collected and computed on-board before transmitting data over the network. Thus, the amount of memory and computational capability required on the vehicle is reduced, as well as the bandwidth needed to transfer the data.

Considering the overall results of the pipeline, the signal and feature selection algorithm yielded better performance than the data augmentation (historicization). In our case, selecting the correct signals and features plays a more strategic role in performance improvement rather than increasing the amount of information available for the classification step. Finally, our pipeline demonstrates performance comparable with black-box state-of-art deep learning architectures but offering higher interpretability and lower resource requirements.

- *Evaluation of the temporal component of the problem.* In a wide range of real-world settings, the input data may include temporal patterns leading to possible data-leakage problems while building the data-driven model.

  We defined two validation strategies to assess whether we can consider independent input data (cycle) to address this issue. The validation of both strategies automatically unveils possible hidden dependence in the input data, helping the domain expert correctly select the best strategy to treat new data in a deployment scenario. The results obtained in the use case under analysis show that choosing the right strategy is fundamental to ensure reliable performance estimation of the deployed solution and suitability of the presented approach to deal with time-dependent data.

As future works, we aim to (i) validate the proposed data-driven methodology in different application scenarios to demonstrate its general-purpose features; (ii) enrich the proposed strategy with a specific solution to automatically detect when the prediction model must be rebuilt to properly predict the new unseen data (i.e., a concept-drift detection methodology).

Appendix A. Signal categories

To gain a better understanding of the types of signals that have been collected during the experiments, we group them into different categories based on which part of the engine they monitor. In detail, Tab. A.10 reports, for each category, the number of all signals exposed by Program A, after the *a-priori* signal selection and after the final signal selection by the *CORR-FS* algorithm. These signals are normally available at the on-board ECU, sampled with a 1 Hz frequency.

Category	**Initial** X	**a-priori** $\hat{X}$	**Final (CORR-FS)** $\bar{X}$
Fuel injection	12	10	4
Test bench	8	0	0
Exhaust gas temperature	5	3	2
Engine airflow	4	3	0
Catalytic converter	3	2	2
Exhaust manifold	3	3	1
Torque control	3	1	0
Diagnostic Trouble Codes (DTC)	2	0	0
Accelerator	1	1	0
Engine temperature	1	1	1
Pressure	1	1	0
Fuel rail	1	1	1
NOx emissions	1	1	1
Oxygen sensor	1	0	0
Combustion mode	1	0	0
Other	3	3	1
Total	50	30	13

Table A.10: Signal categories.

Appendix B. Cycle Division

To split D into $D1$ and $D2$ we first have to perform a *stratified division*, i.e., we do not want a class being highly represented in $D1$ and underrepresented in $D2$. Secondly, since $D1$ is used to perform both k-fold cross validation and time series cross validation, we need $D1$ to be larger than $D2$.

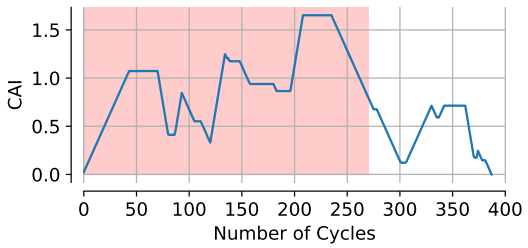


Figure B.13: CAI trend for cycle split.

To join these two constraints we evaluate the class imbalance of $D1$ and $D2$ respectively while increasing the number of cycles in $D1$. We use the *cumulative absolute imbalance* (CAI) as measure. Given the i-th iteration, the set of cycles $D1_i$ composed by the first i cycles from D, we evaluate for each class $c \in C = \{green, yellow, red\}$ the *class representation* $CR_i(c)$ as the fraction of cycles of class $c \in D1_i$ with respect to the number of cycles $c \in D$.

$$CR_i(c) = \frac{|D1_i(c)|}{|D(c)|},\ c \in C,\ i \in [1, |D|].$$

Intuitively, $CR_i(c)$ give us an indication of how well the class c is represented in $D1_i$. Next, we evaluate the cumulative absolute imbalance (CAI) as the sum of the absolute difference between each class representation with respect to all the classes.

$$CAI_i = \sum_{c1 \in C}^{\infty} \sum_{c2 \in C \setminus \{c1\}}^{\infty} |CR_i(c1) - CR_i(c2)|.$$

The higher the CAI_i, the more the classes in the i-th split are imbalanced. When all classes are equally represented, we have $CAI = 0$, which tops to 4 when only one class is represented. Here we choose the best split seeking for the trade-off between lowering as much as possible the CAI index and keep enough cycles in $D2$.

Fig. B.13 reports the value of the CAI while increasing the number of cycles in $D1$ in our use case. The red area highlights the minimum size for $D1$ to run a reliable cross validation (at least 2/3 of D in $D1$). After the red area ends,

the CAI has a decreasing trend up to 300 cycles, suggesting us to split D with 300 cycles in $D1$ and the remaining 88 cycles in $D2$.

Appendix C. Classification Metrics

Firstly, to compute these metrics in for multi-class classification problems (Sokolova & Lapalme, 2009) we need to find for each class c the:

- True Positives (TP_c): the number of instances belonging to c, correctly labeled in the c class;
- False Positives (FP_c): the number of instances not belonging to class c, wrongly labeled in the c class;
- False Negatives (FN_c): the number of instances belonging to c, wrongly labeled in a different class.

Then we derive:

- Precision$_c$: is a measure of exactness. It represents the percentage of instances labeled as belonging to class c that actually belong to it (Sokolova & Lapalme, 2009).
$$Precision_c = \frac{TP_c}{TP_c + FP_c}$$
- Recall$_c$: is a measure of completeness. It captures the percentage of instances of class c that are labeled as such (Sokolova & Lapalme, 2009).
$$Recall_c = \frac{TP_c}{TP_c + FN_c}$$
- F1-Score$_c$: is used to summarize precision and recall metrics. It is defined as the harmonic mean of precision and recall.
$$F1 - Score_c = 2 * \frac{Precision_c * Recall_c}{Precision_c + Recall_c}$$
- Accuracy: is used to summarize overall classification performance. It represent the percentage of instanced correctly labeled with respect to all the instances.
$$Accuracy = \frac{\sum_{c \in C} TP_c}{\sum_{c \in C} TP_c + FP_c}$$

Appendix D. Signals Selection Importance

Fig. D.14 reports, for each signal on the x-axis, the algorithms that select it on the y-axis. Signals are sorted according to the number of algorithms that select them. It is clear from the bar plot that some signals are selected more often than others. The domain experts confirmed that the most commonly selected signals are the most relevant for the problem at hand. More specifically, these include all signals related to the combustion process, i.e., they monitor the fuel, the oxidant, and the exhaust gases.

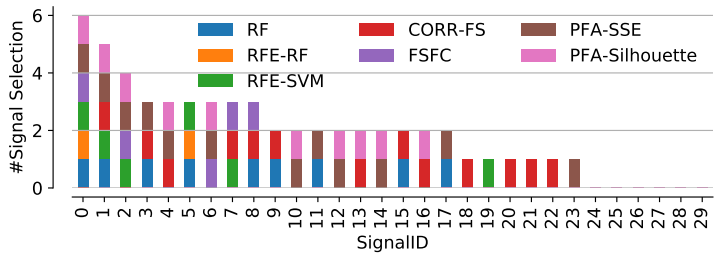


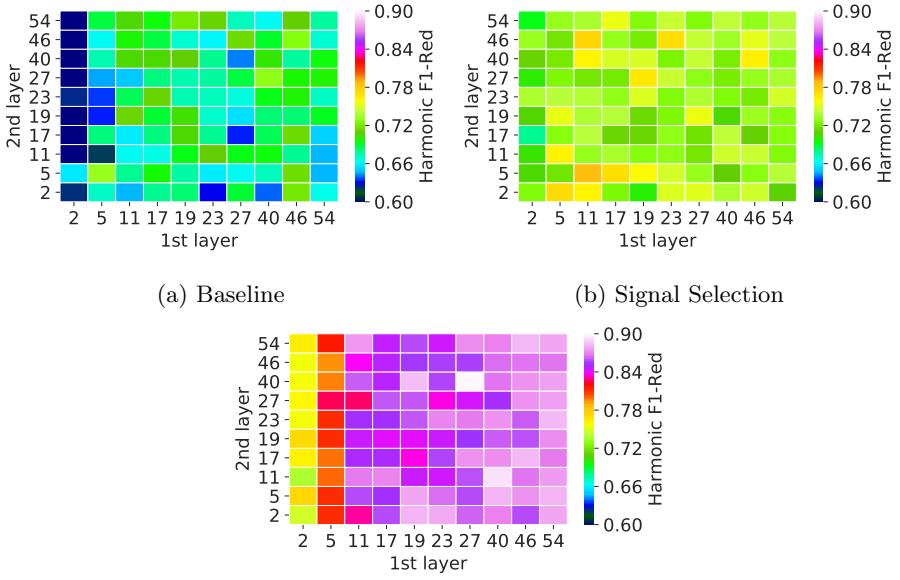
Figure D.14: Signals selected by the various signal selection algorithms.

Appendix E. MLP Stability

The optimization process brings also benefit in the MLP hyperparameter tuning. In particular, one is interested in checking how complex and robust hyperparameter optimization is. In a nutshell, if only one or few combinations result in optimal, then the model is not robust, and the search is difficult. Contrarily, if a considerable interval of parameters results in a good model, the choice is both robust and simple. Fig. E.15 highlights this. The x-axis and y-axis report the number of neurons in the first and the second hidden layers of the MLP. The cell color shows the F1-Score for the red class. Two considerations hold: First, we notice a consistent improvement from Fig. E.15a to (Fig. E.15b) and to Fig. E.15c. This result reflects the benefits of the signal and feature selection as seen in 7. Secondly and more interestingly, we observe that there is

a larger interval of hyperparameters for which performance are in the top values - as reflected by the more homogeneous coloring in Fig. E.15c.

In detail, the number of neurons in the first hidden layer needs to be large enough, i.e., at least 40. Then - any number of neurons at the second hidden layer provides performance higher than 0.84, peaking at more than 0.90. Compared to the other two cases, one can observe more scattered bad configurations. This result confirms how the proper selections of the input features ease the classification task.



(a) Baseline

(b) Signal Selection

(c) Feature Selection

Figure E.15: Improvement in MLP parameter tuning.

References

Barandas, M., Folgado, D., Fernandes, L., Santos, S., Abreu, M., Bota, P., Liu, H., Schultz, T., & Gamboa, H. (2020). Tsfel: Time series feature extraction library. *SoftwareX*, *11*, 100456.

Bishop, C. M. et al. (1995). *Neural networks for pattern recognition*. Oxford university press.

Blum, A. L., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial intelligence*, *97*, 245–271.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Carvalho, T. P., Soares, F. A. A. M. N., Vita, R., da P. Francisco, R., Basto, J. P., & Alcal, S. G. S. (2019). A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, *137*, 106024.

Cerqueira, V., Moniz, N., & Soares, C. (2021). Vest: Automatic feature engineering for forecasting. *Machine Learning*, (pp. 1–23).

Choi, K., Singh, S., Kodali, A., Pattipati, K. R., Sheppard, J. W., Namburu, S. M., Chigusa, S., Prokhorov, D. V., & Qiao, L. (2008). Novel classifier fusion approaches for fault diagnosis in automotive systems. *IEEE Transactions on Instrumentation and Measurement*, *58*, 602–611.

Christ, M., Braun, N., Neuffer, J., & Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, *307*, 72–77.

Ciregan, D., Meier, U., & Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE conference on computer vision and pattern recognition*.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Diaz-Uriarte, R., & de Andrés, S. A. (2005). Variable selection from random forests: application to gene expression data. ArXiv preprint. `https://arxiv.org/abs/q-bio/0503025`.

Donateo, T., & Giovinazzi, M. (2017). Building a cycle for real driving emissions. *Energy Procedia*, *126*, 891–898.

Ekinci, K., & eniz Erturul (2019). Model based diagnosis of oxygen sensors. In *Proocedings of the 9th IFAC Symposium on Advances in Automotive Control AAC*.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, *33*, 917–963.

Genuer, R., Poggi, J.-M., & Tuleau, C. (2008). Random forests: some methodological insights. ArXiv preprint. `https://arxiv.org/abs/0811.3619`.

Giobergia, F., Baralis, E., Camuglia, M., Cerquitelli, T., Mellia, M., Neri, A., Tricarico, D., & Tuninetti, A. (2018). Mining sensor data for predictive maintenance in the automotive industry. In *Proceedings of the 5th International Conference on Data Science and Advanced Analytics*.

Giordano, D., Pastor, E., Giobergia, F., Cerquitelli, T., Baralis, E., Mellia, M., Neri, A., & Tricarico, D. (2021a). Dissecting a data-driven prognostic pipeline: A powertrain use case. *Expert Systems with Applications*, *180*, 115109.

Giordano, D. et al. (2021b). PREPIPE. `https://github.com/SmartData-Polito/PREPIPE`.

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

Hart, J. D. (1994). Automated kernel smoothing of dependent data by using time series cross-validation. *Journal of the Royal Statistical Society: Series B (Methodological)*, *56*, 529–542.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, *28*, 100–108.

Hsu, C.-W., Chang, C.-C., & Lin, C.-J. (2003). A practical guide to support vector classification. Techical Report. `https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

Huang, G.-B. (2003). Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE transactions on neural networks*, *14*, 274–281.

Jun, H.-B., Kiritsis, D., Gambera, M., & Xirouchakis, P. (2006). Predictive algorithm to determine the suitable time to change automotive engine oil. *Computers & Industrial Engineering*, *51*, 671–683.

Khan, S., & Yairi, T. (2018). A review on the application of deep learning in system health management. *Mechanical Systems and Signal Processing*, *107*, 241–265.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*. volume 14.

Last, M., Sinaiski, A., & Subramania, H. S. (2010). Predictive maintenance with multi-target classification models. In N. T. Nguyen, M. T. Le, & J. Świkatek (Eds.), *Intelligent Information and Database Systems*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*, 436–444.

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

Lu, Y., Cohen, I., Zhou, X. S., & Tian, Q. (2007). Feature selection using principal feature analysis. In *Proceedings of the 15th ACM international conference on Multimedia*.

Luo, H., Huang, M., & Zhou, Z. (2019). A dual-tree complex wavelet enhanced convolutional lstm neural network for structural health monitoring of automotive suspension. *Measurement*, *137*, 14–27.

Mesgarpour, M., Landa-Silva, D., & Dickinson, I. (2013). Overview of telematics-based prognostics and health management systems for commercial vehicles. In J. Mikulski (Ed.), *Activities of Transport Telematics* (pp. 123–130). Berlin, Heidelberg.

Mitra, P. ((accessed June 4th, 2020)). *Pabitra Mitra personal webpage*. `http://cse.iitkgp.ac.in/~pabitra/paper/fsfs.tar.gz`.

Mitra, P., Murthy, C., & Pal, S. K. (2002). Unsupervised feature selection using feature similarity. *IEEE transactions on pattern analysis and machine intelligence*, *24*, 301–312.

Moser, M. M., Onder, C. H., & Guzzella, L. (2014). Using exhaust pressure pulsations to detect deteriorations of oxygen sensor dynamics. *Sensors and Actuators B: Chemical*, *191*, 384–395.

Prytz, R., Nowaczyk, S., Rgnvaldsson, T., & Byttner, S. (2015). Predicting the need for vehicle compressor repairs using maintenance records and logged vehicle data. *Engineering Applications of Artificial Intelligence*, *41*, 139–150.

Rakotomamonjy, A. (2003). Variable selection using svm-based criteria. *Journal of machine learning research*, *3*, 1357–1370.

Ran, Y., Zhou, X., Lin, P., Wen, Y., & Deng, R. (2019). A survey of predictive maintenance: Systems, purposes and approaches. ArXiv preprint. `https://arxiv.org/abs/1912.07383`.

Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, *29*, 2352–2449.

Satopaa, V., Albrecht, J., Irwin, D., & Raghavan, B. (2011). Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *proceedings of the 31st international conference on distributed computing systems workshops*.

Shafi, U., Safi, A., Shahid, A. R., Ziauddin, S., & Saleem, M. Q. (2018). Vehicle remote health monitoring and prognostic maintenance system. *Journal of advanced transportation*, *2018*.

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, *45*, 427–437.

Solorio-Fernández, S., Carrasco-Ochoa, J. A., & Martínez-Trinidad, J. F. (2020). A review of unsupervised feature selection methods. *Artificial Intelligence Review*, *53*, 907–948.

Stathakis, D. (2009). How many hidden layers and nodes? *International Journal of Remote Sensing*, *30*, 2133–2147.

Vong, C. M., Wong, P. K., & Wong, K. I. (2014). Simultaneous-fault detection based on qualitative symptom descriptions for automotive engine diagnosis. *Applied Soft Computing*, *22*, 238–248.

Wang, B., Lei, Y., Yan, T., Li, N., & Guo, L. (2020). Recurrent convolutional neural network: A new framework for remaining useful life prediction of machinery. *Neurocomputing*, *379*, 117–129.

Wang, J., Wen, G., Yang, S., & Liu, Y. (2018). Remaining useful life estimation in prognostics using deep bidirectional lstm neural network. In *Proceedings of Prognostics and System Health Management Conference*.

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, *2*, 37–52.

Wolf, P., Mrowca, A., Nguyen, T. T., Bäker, B., & Günnemann, S. (2018). Pre-ignition detection using deep neural networks: A step towards data-driven automotive diagnostics. In *Proceedings of the 21st International Conference on Intelligent Transportation Systems*.

Xin, Q. (2013). Diesel aftertreatment integration and matching. In *Diesel Engine System Design* (pp. 503–525).

Zhang, W., Yang, D., & Wang, H. (2019). Data-driven methods for predictive maintenance of industrial equipment: a survey. *IEEE Systems Journal*, *13*, 2213–2227.