Dissecting a Data-driven Prognostic Pipeline: A Powertrain use case

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

[a] *Politecnico di Torino, Turin, Italy*
danilo.giordano@polito.it, eliana.pastor@polito.it, flavio.giobergia@polito.it, tania.cerquitelli@polito.it, elena.baralis@polito.it, marco.mellia@polito.it
[b] *Punch Torino (Former GM Torino), Turin, Italy alessandra.neri@punchtorino.com, davide.tricarico@punchtorino.com*

Abstract

Nowadays, cars are instrumented with thousands of sensors continuously collecting data about its components. Thanks to the concept of connected cars, this data can be now transferred to the cloud for advanced analytics functionalities, such as prognostic or predictive maintenance. In this paper, we dissect a data-driven prognostic pipeline and apply it in the automotive scenario. Our pipeline is composed of three main steps: (i) selection of most important signals and features describing the scenario for the target problem, (ii) creation of machine learning models based on different classification algorithms, and (iii) selection of the model that works better for a deployment scenario. For the development of the pipeline, we exploit an extensive experimental campaign where an actual engine runs in a controlled test bench under different working conditions. We aim to predict failures of the High-Pressure Fuel System, a key part of the diesel engine responsible for delivering high-pressure fuel to the cylinders for combustion. Our results show the advantage of data-driven solutions to automatically discover the most important signals to predict failures of the High-Pressure Fuel System. We also highlight how an accurate model selection step is fundamental to identify a robust model suitable for deployment.

Keywords: predictive maintenance; automotive; machine learning; classification; svm; neural network.

1. Introduction

With the introduction of the Internet of Things paradigm, data that was previously constrained at the device level can now be transferred and processed elsewhere in the cloud. This enables new possibilities, particularly in those fields where the main limitations were not due to a shortage of data, but rather a limitation in the on-board computing power.

Automotive is one such field: vehicle sensors generate large amounts of data, typically processed by on-board Electric Control Units, with very limited hardware capabilities. Today, this data could be transferred to the cloud, and become a precious mine of useful information to exploit advanced data analytics functionalities. Among these, prognostics, or predictive maintenance, is seen as the most interesting opportunity to reduce costs and improve customer satisfaction.

This study focuses on the High-Pressure Fuel System (HPF), a key part of the diesel engine responsible for delivering high-pressure fuel to the cylinders for combustion. To guarantee efficient combustion, the HPF is responsible for the injection timing, quantity, and pressure (DieselNet. Diesel fuel injection., 2009). These are key factors to guaranteeing both good performance and to limit emissions of polluters. Malfunctioning in the HPF results in the car suddenly stopping while running.

This work aims at engineering a prognostic pipeline to detect the initial symptoms of a drift from the HPF expected behavior. Currently, on-board engine data is recorded by the *Engine Control Unit* (ECU), which is only capable of detecting significantly degraded conditions i.e., to trigger a *Diagnostic Trouble Code* (DTC), and to alert the driver that the vehicle immediately needs service. Through a prognostic approach, we aim to detect the problem before a serious issue happens, implementing a predictive maintenance strategy i.e., recalling to the service a car before the DTC fires, thus allowing for an early intervention.

For this work, we collaborate with General Motors (GM). GM is a leader in the application of automotive prognostics, which is marketed in the US since 2015 under the name of OnStar Proactive Alerts, available on millions of production vehicles. Here we focus entirely on a data-driven approach to prevent the DTC alert. We run experiments in a controlled test bench, where all the engine data is collected. Domain experts control the engine settings to recreate the conditions of a faulty or healthy HPF. Our goal is to use the collected engine data to under-

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

stand what the underlying condition of the HPF is (and whether it is transitioning to a faulty state).

There are lots of challenges to build a production-suitable system. Among them, two prove to be particularly complex: (i) the *limited computing resources* and (ii) the *constraints on the quality* of the trained model. More specifically, the first problem is due to the ECU not being suitable for machine learning purposes. Because of this, a cloud-based approach has to be adopted. The price to pay is that the more interesting data to be analyzed needs to be transferred from the ECU to a remote server, carefully considering the cost to data transfer. Hence, a careful data selection is required to minimize the cost of the transfer. This has a significant impact on the definition of the machine learning solution to consider. As for the second problem, the model must guarantee certain levels of quality to make it deployable in production. Limited performance, would result in calling to the service either too many cars, increasing costs and harming customers' trust; or too few cars, making the model useless. These translate into tight constraints on precision and recall.

With these considerations in mind, we design, develop, and thoroughly test a complete prognostic pipeline for the HPF system. We start from hundreds of sensor signals collected by the ECU. We select the most representative signals based on, domain knowledge, data analysis, and correlation analysis. Next, we transform collected signals into features to train multiple classifiers obtaining different models. We then carefully evaluated these models in terms of performance and robustness to identify the final model suitable for a deployment scenario.

As a final validation experiment, we re-apply the engineered pipeline to a completely new set of data and on a different engine (i.e., a different model for a different vehicle). Our results, demonstrate the promising direction of the approach, achieving performance widely above the constraints imposed by the carmaker on both the engines.

The paper is organized as follows: in Sec. 2 we introduce our dataset, the labeling policy used to identify malfunctions in the HPF system and define our problem. Next, in Sec. 3 we describe how we preprocessed the engine data to select only the most important signals, and then how we extract and select the classification features. In Sec. 4 we present our classification methodology, while in Sec. 5, we discuss how we select the final model usable in deployment. In Sec. 6 we apply the full pipeline on our dataset to validate our approach. In Sec. 7, we present practical challenges related to the on-board implementation and discuss possible decision-making policies about recalling a car to the service for predictive maintenance. In Sec. 8 we summarize the related work. Finally, in Sec. 9 we conclude the paper.

2. Scenario

In this section, we detail the data collection, the experimental setting, the dataset, and the labeling policy. Then, we define our problem and outline the proposed prognostic pipeline.

2.1. Bench Experiment Methodology

To explore the different conditions in which the HPF system works, we run a thorough experimental campaign using a test bench environment. In the test bench, we instrument an actual engine with its on-board sensors and simulate real driving scenarios. Test benches are commonly used to collect large amounts of data without having a pilot driving an actual car. This introduces some significant advantages: first, the same scenario and engine configuration can be reproduced easily as it is not affected by aleatory events; second, large amounts of data can be collected at a reduced cost (e.g., an engine in a test bench can run continuously and overnight). This, however, introduces some complications: test bench data is not affected by some variables that are present while driving a real car (e.g., the effect of vibrations of components, weather conditions, bumpy roads, etc.). These deviations from the "real-world" scenario, though, are often negligible and the quality of data is in most cases more than satisfactory. As a matter of fact, test bench data is even used in the early stages of the preparation for homologation tests.

To monitor different HPF system working conditions, we impose different engine conditions by manually tuning the HPF parameters of two critical components, namely the high-pressure fuel pump reference level and the valve timing aperture. In particular, we manually force these components to work in condition drifting from the specifications. This drift affects the fuel pressure and the fuel flow simulating common HPF malfunctions. This translates into emulating a faulty HPF system up to a point where the ECU triggers a major DTC failure and our engine suddenly stops. In each experiment, we follow 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). We focus on three standard homologation cycles, plus two real driving cycles obtained by recording actual pilots during regular driving sessions. For the homologation cycles, we use the Real Driving Emissions (RDE) (Suarez-Bertoa et al., 2019), Worldwide Harmonized Light Vehicles Test Cycle (WLTC) (Tutuianu et al., 2015) and Artemis standard test cycles (Andr, 2004). Each cycle lasts from 30 minutes up to 1 hour as described in Table 1. We collect data for each HPF configuration, and for each cycle. Overall we run more than 230 experiments.

During the experiment, the test bench records more than 600 signals. These signals are collected from various locations in the engine, monitoring different aspects of it: from signals directly monitoring the fuel rail and fuel injection system, to signals monitoring the engine at large like the torque control, the engine airflow or the DTCs, up to signals monitoring the after treatment system such as NOx emissions, catalytic converter or exhaust temperature. Such a large collection of signals guarantees to

Table 1: Dataset description per driving cycle.

	Cycle	Duration	Green	Yellow	Red	Total
Homol	RDE	60	25	24	38	87
	WLTC	31	28	37	22	87
Real	ARTEMIS	54	15	7	4	26
	DRIVER$_1$	43	15	7	4	26
	DRIVER$_2$	66	19	1	2	22

Table 2: Signals overview.

Description	Number
Fuel Rail	184
Fuel injection	182
Engine airflow	33
NOx emissions	30
Oxygen levels	26
Torque control	21
Catalytic converter	15
Exhaust manifold	13
Exhaust temperature	12
Engine rotation	11
After treatment (Diesel particle system)	10
Diagnostic Trouble Codes (DTC)	5
Others	74
Total	616

our system a broad view of what could be the impact of a malfunctioning HPF system on the rest of the engine. Table 2 summarizes the signals into 13 categories based on what they monitor. While the nature of these signals is important from a domain expert perspective, we have handled them with a data-driven, domain-agnostic approach. Thus, in the following, we will avoid making domain-specific considerations.

Let $x_i(t)$ be the $i-th$ signal at time t, and $X = \{x_i(t), \forall_i\}$ the set of signals. The ECU records samples of signal $x_i(t)$ with two possible sampling strategies:

- **Linear sampling**, the ECU records samples of x_i at a constant frequency. The frequency span from 1 Hz, for slow signals, up to 160 Hz for fast signals.
- **Angular sampling**: the ECU records samples of x_i with a frequency that depends on the engine rotation speed. The faster the engine's rotation, the more frequent the samples are.

This poses practical challenges when processing signals having different sampling rates and strategies.

2.2. Labeling Policy

General Motors experts provide a labeling policy linked to the functioning of the physical system. The labeling policy is based on the highly non-linear error signal P_{error}. In detail, the P_{error} is computed in two steps. Firstly, domain experts compute, in any time (t), the absolute difference $X(t)$ between the target and the measured pressure in the common rail as follows:

$$X(t) = |P_{target}(t) - P_{measured}(t)|$$

Then, a smoothing function removes spikes due to wrong readings and local phenomenon as follows:

$$P_{error}(t) = (1-k) * X(t-1) + k * X(t)$$

The P_{error} is computed on-board with a sampling frequency of 160 Hz, i.e., generating 160 samples each second.[1]

Having the P_{error} for the entire experiment, domain experts label the experiment with a *class label* as follows:

- Red: the HPF is in a critical state, and the car must go to the service. This happen when the $P_{error} > \alpha$ for 5 seconds continuously any time during the cycle;
- Yellow: the HPF system starts drifting from the nominal behavior, but it is not in a critical state. This happen when the $P_{error} > \beta$ for 2.5 seconds continuously any time during the cycle;
- Green: the HPF system works normally, i.e., in all other cases.

Note that a malfunctioning HPF system can still work properly for some maneuver, and it can exhibit malfunctioning period only during some short period of time which correspond to specific maneuver (e.g., high demand of torque for a highway overtake, or for up-hill start). For this, we label each experiment separately, and the labeling policy requires the P_{error} to be consistently offset for quite sizable amount of time.

Finally, during each cycle domain experts check whether the engine triggered any DTC error related to the HPF system. If so, we discard those experiments as a DTC indicates that the HPF system was already compromised.

Since we want to have data about the HPF behaviour in different situations, for each configuration of the HPF parameters (i.e., high-pressure pump and valve), we performed from 2 to 5 different experiments with different driving cycles.

2.3. Problem definition

Given all signals recorded by the ECU, our goal is to identify whether the HPF system is drifting from the nominal working condition. Hence, we aim to give a prognosis of the current HPF system condition rather than predict the remaining useful life of the component. A correct prognosis would allow us to predict the needs of the maintenance, warning drivers that the engine must be checked, or the remote assistance to recall the vehicle.

[1]Full details about the error computation are not disclosed due to the sensitive nature of this information.

For this, we could formulate the problem either as a regression problem, in which we predict the P_{error} signal, or as a classification problem, in which we predict the label to assign to the engine. In the former case, we could achieve a very quick reaction speed by predicting sample by sample the P_{error} signal. However, recalling that this signal is sampled at a frequency of 160 Hz (i.e., every 6.25 ms), this would require, either to implement the regressor directly on-board, or to send to the cloud all the signals' samples at a very high frequency. Given the hardware constraints and the impossibility to implement machine learning functionality directly on-board, the former represents an infeasible solution in our scenario. Similarly bandwidth constraints make it unfeasible to send all data to the cloud (more details follow in Sec. 7.1). Al last, the carmaker is interested in understanding classes of malfunctions to make a prognosis of which car should be recalled to the service rather than understanding the punctual behaviour of the error signal. As such, here we address the prediction of HPF malfunctions by using a classification approach. For this, we design the complete data-driven pipeline described in Fig. 1. We get in input the raw engine data, and performs the following steps:

Preprocessing: we select the most important signals and transform them into features suitable for the classification task. The latters are then filtered via a feature selection step.

Model Training and Tuning: we build and assess the performance of different models exploiting multiple classification algorithms with an extensive hyperparameter tuning.

Model Selection: we select the final model suitable for the deployment.

3. Preprocessing

The preprocessing step aims to prepare the data for the classification task by performing three stages: (i) signal selection, (ii) data transformation, and (iii) feature selection.

3.1. Signal Selection

The ECU exposes hundreds of signals to monitor the engine status. Here, we start from a subset composed of about 600 signals. Not the all of them are useful to predict the HPF status. As such, we perform signals selection to keep only the most informative signals discarding the useless ones. To this aim, we exploit a mix of data-driven techniques and domain knowledge.

Firstly, with the help of domain experts, we discard all signals that are weakly related with the addressed problem. For instance, we remove signals related to the battery charge level or the engine cooling system that are almost independent of the HPF system. Secondly, we remove signals that are constant over time and do not bring any information. We are left with a subset of signals $\hat{X} \subseteq X$.

Next, we perform automatically a signal selection based on the correlation analysis by Giobergia et al. (2018 October) which exploits the Pearson's correlation coefficient between pairs of signals. It demands that the signal samples must be aligned. As we discussed in Sec. 2.1, given a pair of signals x_i and x_j, their sampling frequencies[2] F_i and F_j can be very different. As a result, their samples will be misaligned. When this appends, we must upsample the signal to the highest frequency. We apply the sample and hold technique to keep only real signal values. However, we must carefully consider whether compute the correlation between signal x_i and x_j is reasonable. Look for example Fig. 2, which reports three signals x_1, x_2, and x_3 with three different sampling frequencies F_1, F_2, and F_3. While x_2 and x_1 have similar frequencies, upsampling x_3 at F_1 results in a signal constant most of the time (x_3), degrading the correlation analysis. As such, we compute the correlation between pairs of signals only if the ratio of their frequencies is within a reasonable range. In a nutshell, given two signals x_1 and x_2, and their respective frequencies F_1 and F_2, with $F_1 < F_2$, we upsample F_2 only if $F_2 \leq \frac{F_1}{4}$.

Next, we use the algorithm proposed in (Giobergia et al., 2018 October) to aggregate signals $x \in \hat{X}$ in groups $g \in G$. We greedily create groups of strongly correlated signals by using a parameter called r_{min}. To tune this parameter, we employ the identification of the knee point proposed by Satopaa et al. (2011, June). As output, the algorithm exposes for each group $g \in G$ the signals $x \in g$. Rather than using a domain-agnostic heuristic to select which signals should be kept, the domain experts analyzed the groups $g \in G$ to select which signals should be selected as best representative for the group.

In detail, domain experts select the most general signals recorded by different ECUs. In this way, the learning process performed by the current signal selection process can be easily transferred to cars monitored by a different ECU. Secondly, some signals may derive from a mathematical model describing a phenomenon directly monitored by a sensor. The former is typically used to identify wrong sensor readings. In this case, domain experts discard the mathematical model signal in favor of the raw sensor data. Thirdly, two signals may be strongly correlated because the first one directly monitors raw data from a sensor, while the second is a compensation of the first signal. This appends when the sensor data only partially describes a phenomenon, hence other environmental information should be used to get the correct readings. For example, the oxygen signal monitored by the engine must be compensated with the airflow pressure to get the correct oxygen percentage. In this case, the domain experts select the compensated signal. Finally, in some cases, the same phenomenon can be monitored at different points of the engine, to cope with different software components. In

[2]For angular signals, the sampling frequency is computed as the *mean* sampling frequency.

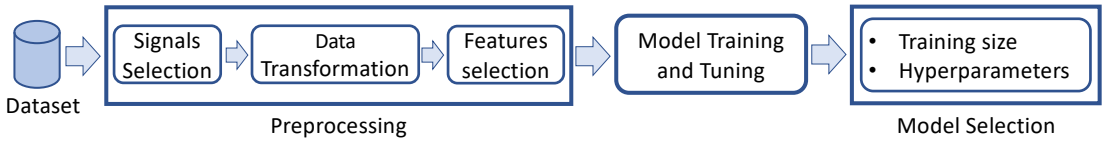


Figure 1: Overview of the predictive maintenance pipeline.

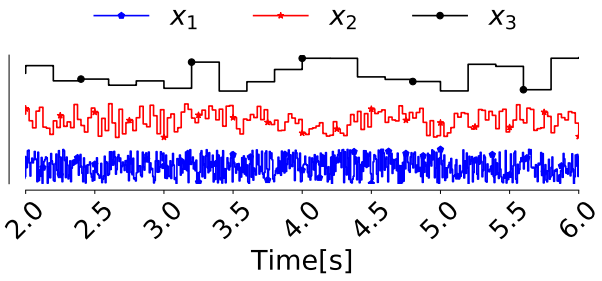
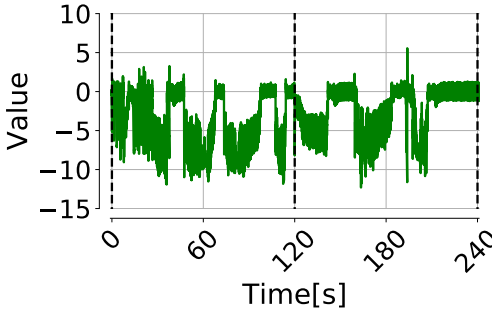
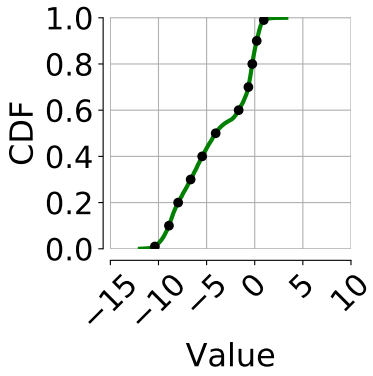


Figure 2: Sampling Difference



(a) Time Windows aggregation



(b) Feature Extraction

Figure 3: Data Transformation

this case, domain experts kept the most upstream signal.

After this stage, we remain with a subset of signals $\tilde{X} \subseteq \hat{X}$.

3.2. Data Transformation

After the signal selection stage, we model signals in features usable by the classifiers. Although the ECU records from thousands up to half a million samples per signal in an hour, the degradation phenomenon is not visible analyzing the data sample by sample, but rather, as discussed in Sec. 2.2, by analyzing how the engine runs for a longer period of time. As such, we exploit the ECU computational capabilities to, aggregate samples into time windows $w(k)$, and to summarize each signals with statistics. In a nutshell, given a signal x and a sample $x(t)$, the sample is assigned to a time window $w(k)$ such as $w(k * \Delta T) \leq t < w((k+1) * \Delta T)$, where ΔT is the *time window length*. For example, look at Fig. 3a, where we choose $\Delta T = 120s$, we aggregate all samples having $0 \leq t < 120$ in the window with ID $k = 0$. Then, we summarize each signal in the time window by means of statistics i.e., the *features*. While these features remove the temporal sequence among samples, they help representing characteristics of the time series that would not be visible in a sample by sample representation. For the representation of the time series, we select a set of N_p percentiles. In Fig. 3b, we summarize the signal in the time window with 11 percentiles (black dots). The percentiles can be seen as samples of the cumulative distribution function (CDF) of the time series. Studying and comparing CDFs makes it easy to identify those phenomena that manifest themselves in terms of variations in the distribution of the values. This brings our feature space to $F = \tilde{X} * N_p$.

3.3. Feature Selection

To reduce the data to process and to transmit, we perform a feature selection stage based on a *wrapping* approach (Blum & Langley, 1997). In a wrapping solution, the feature selection is executed by evaluating the performance of different feature subsets with a classification algorithm (Blum & Langley, 1997). The main advantage of this solution is that it directly shows the predictive performance of the feature subset highlighting the combined prediction capabilities of the selected features. However, finding the best feature selection demands an exhaustive search, as all possible feature subsets should be evaluated. Since this is infeasible, here we propose a heuristic approach that reduces the computational complexity from an exponential problem to a linear one.

Firstly, we rank the features $f \in F$ by using multiple ranking solutions, each one producing a different ranking R.

We exploit two algorithms to ranks the features.

- *mRMR*: it is a state of art a priori method. It ranks the features by means of the *Mutual Information Difference* (*MID*), a metric that combines the importance of each feature (measured as the correlation with the target class), with the redundancy that the feature would introduce (Ding & Peng, 2005);
- Feature importance for the *random forest*: we train a random forest model, and then we extract the Feature Importance (*FI*) which describes how much each feature contributes to the classification process (Breiman, 2001).

We rely on these two algorithms as the former balances the importance and the redundancy of each feature, independently on how these are used during classification, while the latter gives us insights about possible interactions among features during the decision process.

From these algorithms, we derive three distinct rankings R_i, namely:

1. *mRMR* ranking using the red class as the target;
2. *Random Forest* (RF) configured with hyperparameters as suggested in (Genuer et al., 2008), trained using the training set, and all the features in F;
3. *Random Forest Optimized* (RF-Optimized): as before, but we optimize the hyperparameters via a coarse grid search. In the details, given a hyperparameter set, we train the model with all the features and test it with the Validation set. We pick the hyperparameters that lead to the best model (highest F-measure on the red class) and consider the resulting feature ranking.

Once rankings are derived, we iteratively create different feature subsets $S_{R_i}(j)$, where each one is composed by the top j features from the ranking R_i:

$$S_{R_i}(j) = \bigcup_{k=1}^{j} R_i(k) \quad j \in (1, .., |R_i|)$$

Then, we evaluate each feature subset $S_{R_i}(j)$ with a classification algorithm. We train a model using $S_{R_i}(j)$ as input features and we assess each model performance. Finally, we use a *learning curve* (Sammut & Webb, 2011) to evaluate, for each ranking R_i, how the performance changes with increasing information i.e., for each $S_{R_i}(j)$ $j \in (1, .., |R_i|)$.

4. Model Training and Tuning

In this step, we employ different classic and recent classification algorithms to model the high-pressure fuel system behavior, namely, we use: Logistic regression (Hastie et al., 2001), Random Forest (Breiman, 2001), XGBoost (Chen & Guestrin, 2016), Support Vector Machines (Cortes & Vapnik, 1995), and Artificial Neural Networks (Schmidhuber, 2015).

- *Logistic regression* (LR) (Hastie et al., 2001) is a linear model that models the posterior probabilities of the outcomes of a dependent variable via linear functions on multiple independent variables. In our pipeline, we exploit multinomial logistic regression that generalizes logistic regression to multiclass problems and we use regularization to balance the bias-variance trade-off.
- *Random Forest* (Breiman, 2001) is an ensemble learning method that constructs a multitude of decision trees at training time and outputs the class that is the mode of the classes of the individual trees.
- *XGBoost* (Chen & Guestrin, 2016) is a gradient boosting technique (Friedman, 2001) applied to decision trees. Through different boosting rounds, a decision tree is trained to iteratively improve its performance on previously misclassified training points. In particular, XGBoost is designed to be an optimized distributed gradient boosting library.
- *Support Vector Machines* (SVM) (Cortes & Vapnik, 1995) are a set of discriminative classifiers that find the hyperplane that better categorize the data by maximizing the margin. SVMs can handle classes with complex non-linear decision boundaries.
- *Artificial neural networks* (ANN) (Schmidhuber, 2015) are a class of techniques inspired to the biological neural systems. ANN are based on a set of connected units, called artificial neurons. Neurons are usually structured in connected layers divided into an input layer, one or more hidden layers, and an output layer. In particular, in our pipeline, we exploit the multilayer perceptron (MLP) feed-forward artificial neural network.

We train each model with experiments belonging to a *training set*. Then, we evaluate the model performance through a separate set called *validation set*. To tune the classifier hyperparameters, we run a grid search optimizing the performance on the validation set. This allows us to find, for each classification algorithm, a *candidate model* suitable for the deployment phase.

Performance metrics and visualization

Classification performance is assessed through quality metrics of the trained model including precision, recall, and F-measure and visualization approaches.

To compute these metrics, first we need to find for each class c the:

- True Positives (TP): the number of instances belonging to c, correctly labeled in the c class;
- False Positives (FP): the number of instances not belonging to class c, wrongly labeled in the c class;

- False Negatives (FN): the number of instances belonging to c, wrongly labeled in a different class.

Then, we compute:

- Precision: is a measure of exactness. It represents the percentage of instances labeled as belonging to class c that actually belong to it (Han et al., 2012).

$$Precision = \frac{TP}{TP + FP}$$

- Recall: is a measure of completeness. It captures the percentage of instances of class c that are labeled as such (Han et al., 2012).

$$Recall = \frac{TP}{TP + FN}$$

- F-measure: is used to summarize precision and recall metrics. F-measure is defined as the harmonic mean of precision and recall and balances between precision and recall.

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Precision, recall, and F-measure allow one to summarize classification performance. To provide insights about overall mispredictions we rely on the Confusion matrix (Han et al., 2012). Instead, to check the mispredictions on single experiments, we propose the usage of a novel *visual representation*, referred as *mismatch matrix*. The mismatch matrix represents, for each experiment $e \in$ *validation set*, whether a time window $w_e(k)$ is misclassified or not. An example of a mismatch matrix is reported in Fig. 4. The x axis reports the experiments e under evaluation, grouped by class label (on top of x-axis). While the y axis reports the experiments time windows $w(k)$. The color of the cell depends on the predicted class. White means that the prediction is correct, otherwise the cell is colored with the mispredicted class color. Hence, the mismatch matrix allows us to inspect individual wrong classifications maintaining the concept of a cycle and time window within a cycle, thus highlighting possible patterns. For example, Fig. 4 shows that experiment 3 results are critical, with almost all windows $w_3(k)$ that are misclassified as red (instead of yellow). Inspecting Fig. 4 by row, it reveals frequent misclassification patterns at specif times e.g., time window 7 and 8 are frequently (mis)classified as yellow. This highlights where the model lacks predictive capabilities, hence where the model needs improvements.

In our analysis, we use precision, recall, and F-measure for the *red class* to optimize model parameters. We then use the confusion matrix and the mismatch matrix to easily identify classification errors. With the latter, being particularly useful for the final decision-making process as we will discuss in Sec. 7.2.

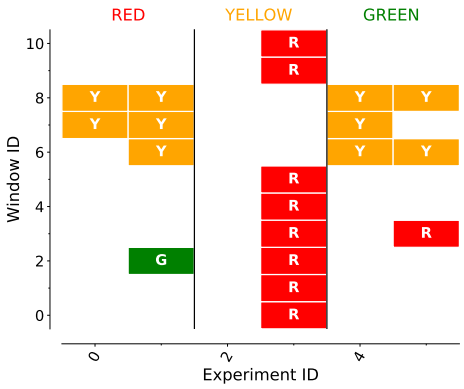


Figure 4: Mismatch matrix example.

5. Model selection

After finding the best candidate model for each classifier, we select the *final model* that will be used in deployment. We need to find both which family of models to use, and then which hyperparameters to set. For this, we look for models that: (i) achieve stable classification performance while changing the training set size, and (ii) where the hyperparameters lie in a part of the hyperparameters space where also other configurations achieve similar performance. Both conditions allow us to select a solution that is robust and generic, i.e., does not suffer for overfitting. Once we select the final model, we verify it with a new independent test set to verify generalization capabilities. In the following, we detail our approach.

Training size The study of the training set size is instrumental to balance the cost of producing training data and the model fitting process. Indeed, collecting experimental data has a cost in terms of data acquisition, labeling, and data preprocessing. This analysis shows how much we benefit from adding more training data, assessing the acceptable amount of data (Provost et al., 1999, August).

For this, we rely on the learning curve (Blum & Langley, 1997). Starting from an empty set of experiments $T(j)$, where j is the step we are performing, we iteratively add experiments e to $T(j)$ to grow the training set. At each iteration, we randomly add one experiment per class to keep the classes balanced. We then train a new model for each classifier. With the best hyperparameters selected when using the entire training set. Next, we evaluate precision and recall metrics, testing the model with the validation set, and with $T(j)$ itself. We continue the process until $T(j)$ includes all experiments. Since we are adding experiments randomly, we repeat the entire process N times. Finally, we plot the average learning curve in function of j.

Hyperparameters stability By optimizing hyperparameters for a given validation set we might suffer from overfitting, i.e., choosing a very specific set of hyperparameters. To avoid this, we study the impact of little

hyperparameter perturbation with respect to the one selected by the candidate models. Intuitively, this allows us to discover whether the candidate models lie in an area where also other configurations offer similar performance eventually highlighting instability.

6. Experimental results

We evaluate the proposed pipeline on the real data collected with the test bench described in Sec. 2.1. Collected data is split into three disjoint sets, the *training set* composed by only experiments from homologation cycles, the *validation set* composed only by experiments of DRIVER$_1$ cycle, and the *test set* composed by a mix of experiments from RDE cycles, DRIVER$_1$ cycles, and DRIVER$_2$ cycles. The test set includes a mix of cycles to analyze the performance of the proposed approach on a heterogeneous set of data. Tab 3 summarizes the cardinality of each set and the label distribution. We recall here that our focus is to reach high-quality metrics for the red class since mispredictions on this class have higher costs with respect to the other classes. Hence, the carmaker defined quality thresholds coming from the business requirements to send to maintenance as many red vehicles as possible, and possibly limit green or yellow ones. Thus, we subjected the classification setting to minimum thresholds of precision and recall for the red class. Specifically the candidate models must have at least a precision of 0.7 and a recall of 0.5.

Table 3: Sets description

Cycle	Green	Yellow	Red	Total
Training set	68	68	64	200
Validation set	15	7	4	26
Test set	19	1	2	22

6.1. Preprocessing results

Here we present the results for the preprocessing step. Table 4 briefly summarizes the results and their impact on the classification process.

Table 4: Preprocessing overview

Signal Selection

Step	Original	Selected	Precision	Recall	F-measure
1) Domain Driven	614	551	-	-	-
2) Data Analysis	551	285	0.534	0.989	0.693
3) Correlation	285	43	0.729	0.489	0.585

Data Transformation

Step	Original	Transformed
4) Summarization	43 Signals	473 Features

Features Selection

Data	Original	Selected	Precision	Recall	F-measure
5) Features	473	25	0.824	0.852	0.838
(Signals)	43	6			

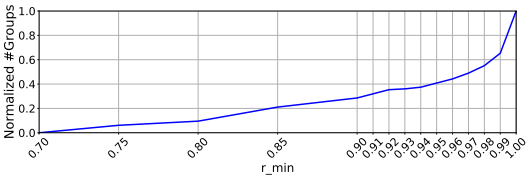


Figure 5: Choose R_{min}

6.1.1. Signal Selection

In each experiment, we collect more than 600 signals. We first remove signals that are totally unrelated with the addressed problem reducing to 551 signals (Table 4 signal selection (1)). Next, we discard signals not carrying any information as they take exactly the same value over all experiments. We are left with 285 signals (Table 4 signal selection (2)).

Next, we execute the correlation-based signal selection algorithm. This algorithm requires r_{min}, the threshold above which two signals are considered strongly correlated. We automatically choose the best r_{min} based on the knee point identification (Satopaa et al., 2011, June). Fig. 5 reports the normalized number of groups that will be created for different values of r_{min}. We choose $r_{min} \in [0.7, 1]$. The knee point identification locates the knee at $r_{min} = 0.95$. This value allows us to group only strongly correlated signals while limiting the number of groups to 40% more groups with respect to $r_{min} = 0.7$. From each group the domain experts select the most representative ones as discussed in Sec. 3.1. At the end of the process we are left with 43 representative signals (Table 4 signal selection (3)).

6.1.2. Data Transformation

To transform signals in features, the data transformation stage requires the definition of the window length ΔT and the percentiles to extract. In this stage, the choice of ΔT is driven by how often we want to verify the engine health condition. Intuitively, one desires to check it as frequently as possible. Yet the signals' frequency takes an important role as a short time window does not allow the system to collect enough samples to reliably estimate the signal distribution. Given the slowest signals are sampled at 1 Hz, to collect enough samples of all signals, we use a time window of length $\Delta T = 120s$ which results also reasonable by domain experts. A sensitivity analysis of this parameter will be given in Sec. 6.4.

Given a time window $\Delta T = 120s$, we compute the distribution of each signal is computed and summarize it with the percentiles. We consider the 9 deciles from the 10^{th} to the 90^{th}, plus the 1^{st} and the 99^{th} percentiles (11 percentiles in total). We consider 1^{st} and the 99^{th} percentile to sample the head and the tail of the distribution while avoiding to consider the minimum and maximum which are very sensitive to noise and outliers. After the data transformation stage, each experiment is described by $43 \cdot 11 = 473$ features.

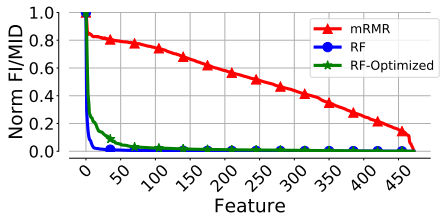


Figure 6: All Rankings

6.1.3. Feature Selection

We apply the heuristic approach for feature selection described in Section 3.3 exploiting the three rankings derived from mRMR, random forest feature importance, and its parameter optimization (respectively referred to as mRMR, RF, and RF-optimized).

Fig. 6 reports the three rankings with the importance of each feature normalized with a min-max normalization. Consider the RF rankings first. Despite their similar behaviours, they have very different trends and ranking results. The RF-Optimized tends to give non-negligible importance to a larger number of features with respect to the RF solution. This implies that in the RF-Optimized solution, different subsets of features have been used by the classification algorithm. Considering the mRMR ranking, instead, this ranking shows a very different trend. A few features have a very high score, almost linearly decreasing in importance. The difference in trends is given by the different ways the algorithms compute the importance. While RF identifies the most important features for the classification stage, mRMR searches for the subset of most important - yet not redundant - features. Given the very different trends among the rankings, choosing how many features to use is not obvious. As such, we rely on a *wrapping* approach that allows us to empirically evaluate the best feature subset by directly using a classification algorithm. As a classifier, we use a SVM model. Here we rely on the SVM classifier as it offers a good trade-off between capabilities of handling nonlinear problems (Cortes & Vapnik, 1995), and the ease of hyperparameters tuning given the lower number of parameters with respect to the other classifiers. Since we are not aware of which hyperparameter configuration may perform well, for each feature subset $S_{R_i}(j)$, we perform a lightweight grid search of 400 hyperparameter configurations creating 400 different models.[3] We train each model on the training set and evaluate it with the validation set. For each feature subset $S_{R_i}(j)$, we pick the hyperparameter setting having the highest F-measure of the red class. We also evaluate the performance of this model with the training set itself. This validation allows us to spot whether performance on the training is maintained in the validation set too. We perform 100 experiments for each ranking, training, and validation in total we compare 120 thousands models.

[3]For the hyperparameters we use a RBF kernel, and we equally sample C and γ spaces (Hsu et al., 2003).

Fig. 7 reports for the three rankings R_i, the classification performance for increasing j. In Fig. 7a, we observe how the mRMR algorithm requires more than 60 features before offering stable performance. RF (Fig. 7b) and RF-Optimized (Fig. 7c) demand fewer features. The latter shows more consistent performance. In particular, looking at Fig. 7c, with more than 25 features, performance increases in the training set, while decreases for the validation set. This is a symptom of overfitting to the training data. As such, since we are interested in building a model able to generalize, we select the first 25 features from the RF-Optimized ranking for the next evaluations. This allows us to drop the number of signals that must be monitored by the ECU from 43 to just 6 signals.

6.1.4. Impact of Data Selection

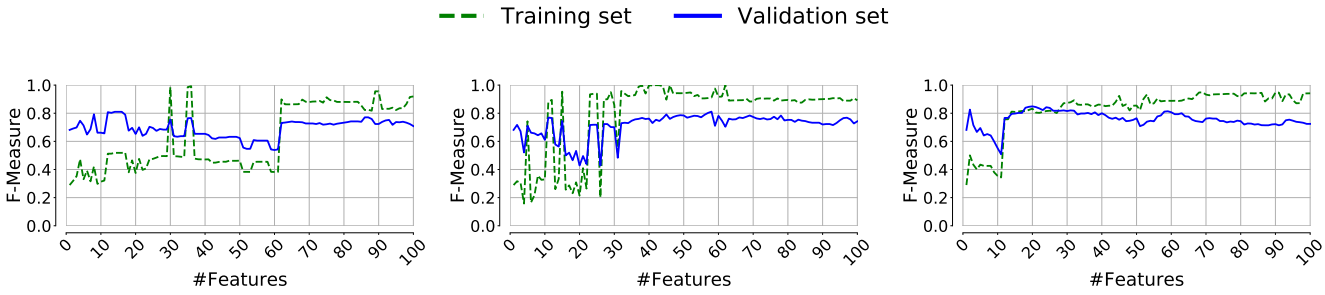
Here, we present a sensitivity analysis aimed at computing the impact on the classification task of each choice. For this, we first transform in features the signals selected after each step. We do not consider the signals before the data analysis selection as constant signals would not contribute to the classification task. Then, we perform a SVM grid search as in the Feature Selection step. Tab. 4 reports the best precision, recall, and F-measure achieved in the validation set for each step. After the data analysis the performance is low, with a precision well below the defined threshold. The correlation signal selection improves it at a disadvantage of the recall, now below the minimum threshold of 0.5. The feature selection achieves much better performance, with both precision and recall above the requirement thresholds. This, confirms the importance of the data selection phase.

6.2. Model Training and Tuning results

Here we compare the performance of five classifiers. We run a grid search process to select the hyperparameters and observe which classifiers meet the required minimum precision and recall thresholds. Next, we select the best model for each classifier, namely the *candidate model*, to finally select the *final model*.

For hyperparameter selection, we are interested in finding which hyperparameters produce the best precision and recall while generating also stable performance without suffering from overfitting. To run the grid search, we exploit a parallel computing system that allows us to train and test thousands of models in parallel. Table 5 details the ranges we use for each hyperparameter and algorithm. For each setup, we train the model using the training set and evaluate the performance using the validation set. Thanks to the parallel computation system the time required for the extensive grid search drops from days to hours for the MLP, from a day to less than an hour for SVM, and from hours to a few minutes for the Logistic regression, Random Forest and XGboost.

Fig. 8 reports the recall (x-axis) and the precision (y-axis) for the red class obtained by each model on the val-



(a) MRMR (b) RF (c) RF Optimized

Figure 7: Feature Selection

idation set. The red bars report the minimum threshold on the recall and on the precision.

As shown in Fig. 8a and Fig. 8b, none of the Logistic Regression and the Random Forest models meet the minimum performance thresholds. Hence, we discard the Logistic Regression and Random Forest classifiers.

Considering, XGBoost ((Fig. 8c), SVM Fig. 8d, and MLP (Fig. 8e) some of their configurations meet the target thresholds. Looking at XGBoost first, it demonstrates better performance than the classic Random Forest ensemble learning. However, only a few models slightly pass the precision threshold. Considering SVM an accurate tuning of the hyperparameters is needed as the resulting model may achieve very variable performance both in terms of precision and recall. Instead, the MLP classifier seems more stable in terms of performance. All hyperparameter configurations meet the minimum required recall, while precision widely varies.

Since SVM and MLP show the most promising performance we perform the model stability analysis with these two classification algorithms only. For this, we select the best models for SVM and MLP (i.e. the candidate models) as the models having the best F-measure on the red class.

Table 5: Grid Search

Classifier	Parameter	Values
Logistic Regression	Solver	{newton-cg, lbfgs, sag, saga}
	C	$[10^{-3}, 10^{3}]$ step 50 log scale
	penalty	l2
	multi_class	multinomial
	max_iter	[100, 500, 1000]
	class_weight	[balanced, None]
Random Forest	Impurity Decrease	[0, 0.02] step 0.005
	Min samples leaf	[5, 35] step 5
	Estimators	{10, 15, 20, 30, 50, 100, 150, 200, 250, 500}
	Split Criterion	entropy
	Max features	{auto, log2 , None, 0.5}
XGBoost	# of boosting rounds	1000
	Maximum tree depth	{2, 5, 7, 10}
	Learning rate	{0.01, 0.05, 0.25, 0.5}
	Minimum child weight	{1, 0.01, 0.05, 0.25, 0.5}
	γ	{0, 0.5, 1, 5, 10}
	subsample ratio	{0.1, 0.3, 0.5, 0.7, 0.9, 1.0}
	columns subsample ratio	{0.1, 0.3, 0.5, 0.7, 0.9, 1.0}
SVM	Kernel	rbf
	C	$[10^{-3}, 10^{3}]$ step 100 log scale
	γ	$[10^{-3}, 10^{3}]$ step 100 log scale
MLP	$1st$ Layer	[25, 225] step 25
	$2nd$ Layer	[25, 225] step 25
	Activation	{Logistic, tanh}
	Seed	100 random values
	Solver	Adam
	Tolerance	10^{-4}

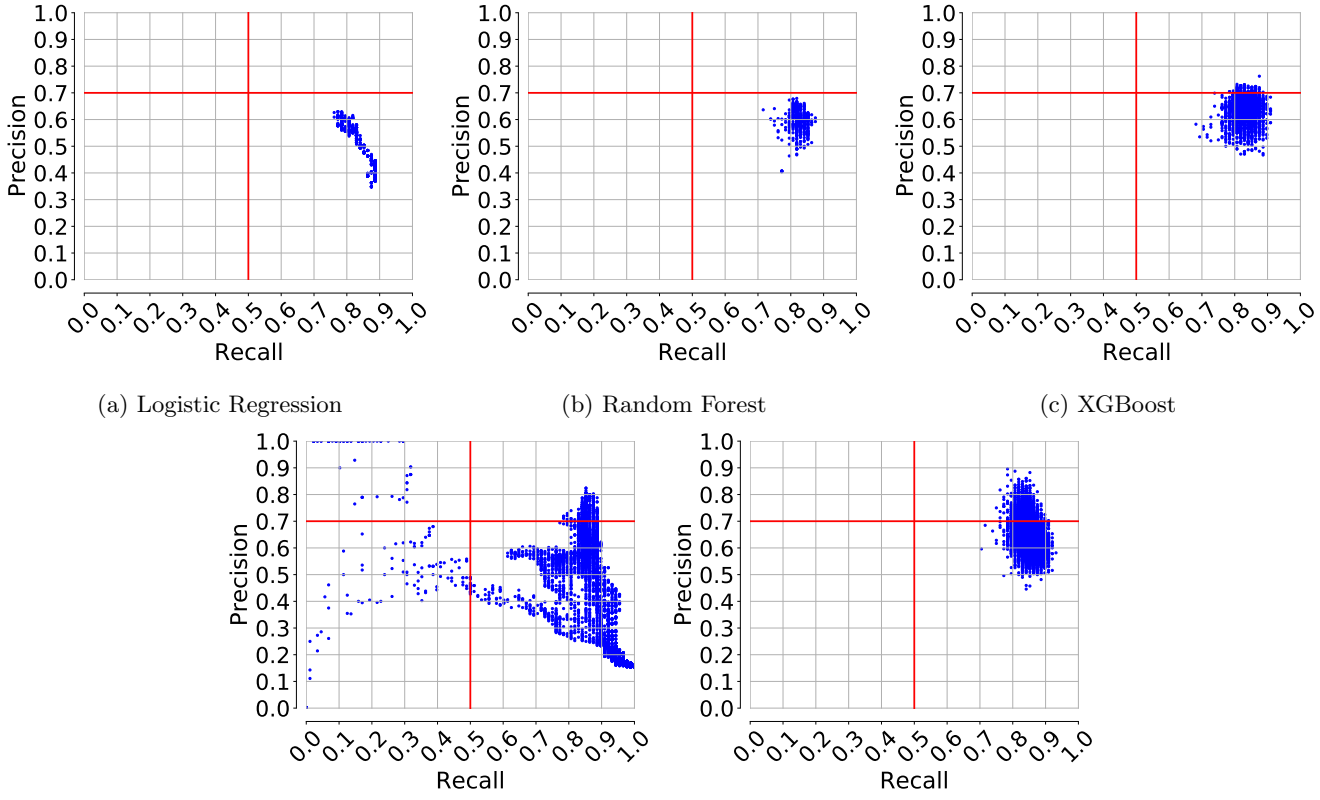
6.3. Model selection

We now check which model would be usable in a deployment scenario. For this, we evaluate the stability and generalization capabilities for each candidate model. In detail, we evaluate the sensitivity of the performance versus the size of the training set and we check if small changes in the best hyperparameters do not impact the performance, i.e., if the hyperparameters lie in a space where different configurations offer similar performance. Finally, we evaluate the performance of the final model with new and independent test set.

Sensitivity to training size

We build a learning curve by training the model with an increasing amount of data, i.e., creating at each step j, a training set $T(j)$, with $|T(j)| = j$. Here we use the hyperparameters of the candidate models. We assess the performance of the model created at step j by using the validation set, and $T(j)$ itself. For each step, we consider 100 different subsets by randomly extracting j experiments from the original training set. We compute the learning curve calculated using $T(j)$ for testing as it provides an indication of how well the model is learning. Intuitively, the more data we provide during training, the better we expect the performance. On the other hand, we consider the curve calculated using the validation data set to observe how well the model generalizes. Intuitively, very good performance on training set do not guarantee good performance on the validation set, i.e., the model may suffer from overfitting of $T(j)$.

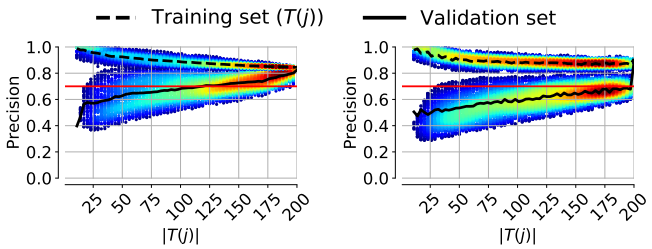
Fig. 9 reports the heatmaps of the precision of the red class over all the runs for increasing j, for the training and validation sets. Notice that the redder is the area, the more runs achieved that performance. The black curves report



(a) Logistic Regression (b) Random Forest (c) XGBoost

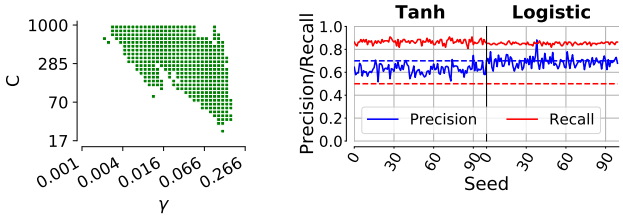
(d) SVM (e) MLP

Figure 8: Grid Search results. Each dot represents on classifier setup. Red lines are the minimum precision and recall performance to meet.



(a) Precision SVM (b) Precision MLP

Figure 9: Training Stability.



(a) SVM Focus (b) MLP Focus

Figure 10: Classification Parameters.

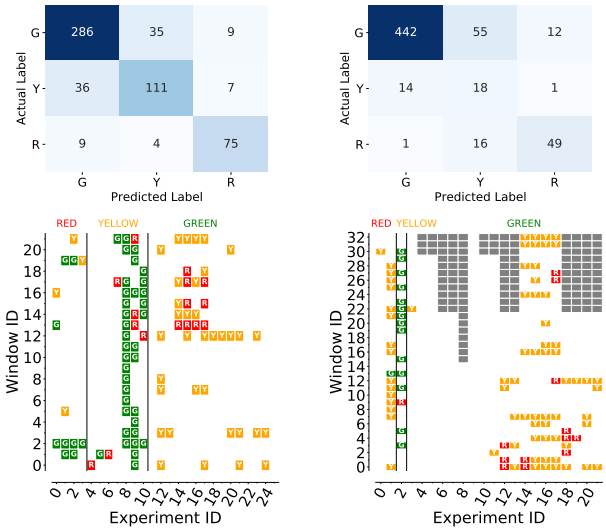
the average performance with the training sets (dashed line), and the validation set (solid line).

Focus on SVM first. Fig. 9a shows that the average precision follows a decreased trend when tested on the training set, and an increasing trend when tested on the validation set. This suggests that the model requires a large amount of data to be generic enough. Considering the spread of the precision, for a given j, the more the number of experiments, the smaller the spread is. A symptom that we can create a general model with a suitable variety of data. The colored error bars highlight how more stable the model becomes for larger training set size.

Considering the MLP learning curve (Fig. 9b), the average precision on the validation set always remains below the minimum performance threshold. It only suddenly rises when all available experiments are used to train the model. Hence, the model is very sensitive to the training data and it clearly suffers from an overfitting phenomenon. In conclusion, by varying the input training set size, the SVM results are more stable with respect to MLP.

Hyperparameter stability

We now investigate the impact of small variations on hyperparameters on classification performance. For SVM, Fig. 10a shows those γ and C combinations for which precision and recall are above the target thresholds. We have 440 configurations that meet the performance requirements. Those are densely compact in the hyperpa-



(a) *Validation set*: Precision 0.824, Recall 0.852

(b) *Test set*: Precision 0.790, Recall 0.742

Figure 11: Classification Performance

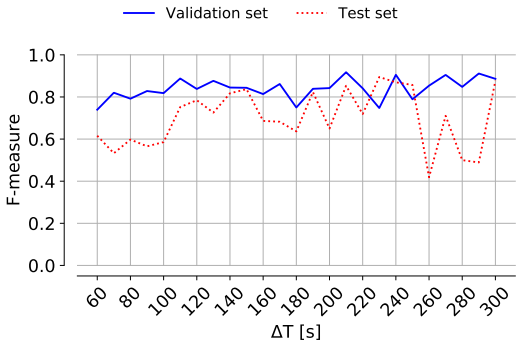


Figure 12: Impact of the Time windows duration on the classification performance

rameter subspace. Intuitively, little hyperparameter perturbations do not harm SVM performance.

For MLP, we have more than two hyperparameters. We start investigating the two main ones i.e., the number of neurons in the 1*st* and 2*nd* layer. The result (not reported for the sake of brevity) shows that, given any combination of neurons, it is possible to find at least one model meeting the minimum performance. In total, we find 3260 good configurations. However, for each configuration of neurons and activation function, we perform 100 experiments just changing the initial random seeds. One would expect that the initial random seed shall not play any role. However, this is not the case. Fig. 10b reports, for the neurons combination with the highest performance, precision and recall of the red class varying the seed and the activation function. The dotted blue and red lines represent the precision and the recall minimum thresholds. Almost all configurations exceed the recall threshold, but only a few are above the precision threshold. And this depends on the random seed only. This result is generic, only some "random" configurations of MLP reach the desired performance for any neuron layer configuration. Therefore, MLP tuning is not stable.

Given the result of these analyses, we select SVM as the *final model*.

Testing with new data

As last, to evaluate if the model would achieve acceptable performance in deployment, we evaluate the performance with a set of data never used before, i.e., the test set.

Fig. 11 reports performance on the validation set (left) and on the test set (right) using the confusion and the mismatch matrices. Firstly, looking at the overall performance, we can see how the selected model passes the minimum performance thresholds on the test set as well. Looking at the confusion matrix of the Validation set (Fig. 11a), we can see how only a few green and yellow windows are misclassified as red. Recalling that our goal is to send to the service only malfunctioning cars, this represents an important milestone. Focusing on the test set (Fig. 11b), we can see very similar results, demonstrating the generality of the model. Next analyze the outcome of the mismatch matrices. This helps us understanding whether the misclassified windows are concentrated in only a single cycle, i.e., increasing the probability to wrongly recall a car to the service, or spread across multiple cycles. Looking at the mismatch matrix of the validation set (Fig. 11a), we can see how in a few cases the engine behaviour differs from the applied label. For instance, experiment 8 has more green windows with respect to yellow ones. Focusing on the red experiments, we can see how our solution well models the engine behaviour with only a few misclassified windows, i.e., we can easily identify which car needs to go to the service. Fig. 11b reports how the model performs similarly in the test set. Here, only experiment 1 shows several misclassified windows with most of them correctly classified as red, i.e., 17 out of 33. The others are classified as yellow or green as the engine partially behaves like a car drifting from the normal behaviour. This confirms that a malfunctioning HPF can still behave normally under some part of the cycle. To handle these cases, in Sec. 7.2 we discuss the decision-making process to decide when send a car to the service.

Finally, to assess the easy reproducibility of the classification pipeline, we evaluate the performance of the entire pipeline on data coming from a new engine. As for the previous case, we divide the data into three distinct sets. Then, we train an SVM model with the training set, where the data is described only by the features selected with the first engine. To find the best hyperparameters we run a grid search optimizing parameters with the validation set. The results show that also with this engine the SVM overcomes the minimum performance thresholds by yielding stable behavior.

6.4. Impact of Time windows size

Once we finish setting all the parameters, we determine how frequently we should compute our predictions. This translates into tuning how frequently we should compute the features given a time window of new data. On the one hand, a large time window allow us to collect more data, hence having a stable picture of the engine behaviour. On the other hand, smaller windows allow us to capture more frequently the engine status, hence making the final decision based on more observations.

We consider window size in $[60, 300]$ s with a pace of 10 seconds. For each ΔT, we create the model by finding the best SVM hyperparameters via a grid search. We select the best performing model on the validation set and use it to predict the labels of the testing set as well. Fig 12 reports the F-measure versus ΔT. Short time windows do not allow to fully capture the engine status - with too few samples to correctly measure the percentiles. A window size in $[110, 160]$ s shows the most balanced performance with all datasets having similar performance. Increasing the ΔT reduces the size of training set, which in turn causes more unstable results for the test set. As such, we confirm that the choice of $\Delta T = 120$ s offers a good trade-off between the number of decisions and performance.

7. Discussion

After evaluating how to perform our prognostic pipeline, here we briefly discuss practical aspects of the implementation, namely the computational complexity to implement it on-board, and the decision-making process to identify which car should go to the service.

7.1. Complexity

Along with the paper, we employed different methodologies to limit the amount of data to be stored on-board to cope with the limited hardware capabilities and to reduce the bandwidth required to transmit the data to the cloud.

Bandwidth Requirements

To quantify the bandwidth required along the pipeline, we compute the number of samples that the ECU should collect and transfer every second in case different data transformation is employed. This amount changes based on the subset of signals considered. In particular, we consider three different scenarios: (i) the ECU transfers all the signals, i.e., 614 signals; (ii) the ECU transfers the subset of signals after the domain knowledge and data analysis signal selection, i.e., 285 signals; and (iii) the ECU transfers the subset of signals left after the full signal selection process, i.e., 43 signals. Next, for each signal subset, we estimate the bandwidth requirements by considering that each sample is encoded as a 4-bytes floating-point number. Finally, we compare these estimations with the bandwidth required to send only the features, i.e., 25 features encoded as 4-bytes floating number each, sent once every 120 seconds.

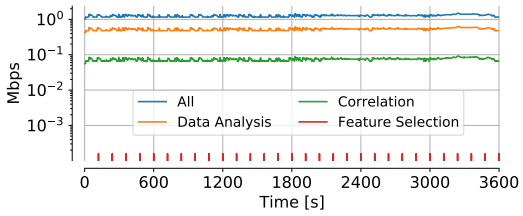


Figure 13: Bandwidth required to transmit signals and features to the cloud

Fig. 13 reports the number of bits per second to transmit, evaluated for 1 hour. When all signals are transmitted (blue line), the ECU should constantly transmit more than 1 Mbps. This definitely represents an infeasible scenario. Considering the subset of signals after the domain knowledge and data analysis (orange line), the required bandwidth only halves. As expected, the best improvements are achieved at the end of the signal selection (green line) where 43 signals ask for a constant bandwidth of about 100 kbps yet, this calls for significant cost both for on-board and cloud connectivity. Considering the bandwidth required to transmit just features (red dots), we require only 100 bytes every 120 seconds. This solution makes the transmission easily affordable also in the automotive scenario when an unstable connection may be present.

Memory Requirements.

Before the data transmission, we require to compute the percentiles by the on-board ECU. As highlighted in Sec. 6, after our feature selection stage we monitor only 6 signals. Despite this low number, collecting all the samples for 2 minutes can be memory consuming in a scenario where little memory is present, e.g., in the order of a few MB for all the applications running in the ECU. Indeed, considering again that each signal is encoded with a 4-byte floating-point number, that each signal may generate a new sample every 6.25 ms, in total, in 2 minutes we may have to store up to 20 thousand samples per signal for a total of 120 thousand samples equal to 480 kB. To reduce the amount of data required to compute percentiles, several algorithms are present in the literature. Jain & Chlamtac (1985) deployed the heuristic P^2 algorithm to estimate the percentiles on the fly, without storing observations. Greenwald & Khanna (2001) propose a solution to accurately compute the percentile with a memory footprint of $O\left(\frac{1}{\epsilon}log(\epsilon N)\right)$ in function of the precision ϵN and the number of samples N. This is recently been confirmed as the optimal space-bound by Cormode & Veselỳ (2020). These solutions can be effectively put in place to reduce the amount of memory required making the ECU implementation feasible.

Cloud Computational Requirements

Regarding the computational cost, this evaluation is not critical as, after the training phase, testing is done in

the cloud and asks very little time. In the case of limited resources, the carmaker can easily scale-up the required back-end. For instance, considering a medium-level server equipped with an Intel Xeon Gold 6140 CPU at 2.30 GHz and 32 GB of RAM, and by testing with our Python prototype, we can execute about 9000 classifications per second, potentially handling more than 1 million vehicles every time window.

7.2. Decision Making

While the proposed pipeline assesses the state of the HPF for each time window, the carmaker is ultimately interested in finding a decision-making policy to recall cars to the service.

We conducted a preliminary study based on the validation set for the assessment of a voting strategy and time span for making the decision. For instance, a majority voting strategy over all time windows in an hour span shows satisfactory results both on the validation and on the test set. Indeed, in both cases, all and only the red cars would be correctly recalled to go to service.

However, tuning these thresholds for decision-making requires a proper calibration with additional data. An initial *soft release* of the proposed pipeline is required to collect more data and to validate the performance. During this phase, our pipeline should be implemented in cars, and when cars go to the service, both statistics about the pipeline and the HPF should be collected to investigate the best decision strategy. This analysis is required to allow the car-maker to run a data-driven calibration of the criteria. In a second stage, a *hard release* of the system should take place in which the carmaker actually recalls the cars as the system can be used for the prognosis of the state of the HPF system.

8. Related Work

The topic of predictive maintenance has been of particular interest in recent years. The enabling technologies at the core of the Internet of Things paradigm (more specifically, connected devices and cloud computing) have brought predictive maintenance within reach. As such many fields studied approaches to predict maintenance operations. For instance, authors in (Baptista et al., 2018; Ferreiro et al., 2012) studied how to predict maintenance in aircraft, authors in (Rabatel et al., 2011) detect anomalies in railway to predict potential failures, authors in (Renga et al., 2020) study the problem of prognostic vs diagnosis to study an electric distribution network, authors in (Rohani et al., 2011) predict repair and maintenance costs of a fleet tractors, while authors in (Proto et al., 2019, July; Apiletti et al., 2018, December) proposed data-driven methodology to support predictive maintenance in the era of Industry 4.0. The two main approaches to predictive maintenance found in literature are *model-based* and *data-driven*. The former is based on the introduction of physics-based models of the system under study, along with its possible interactions with other components. The latter approach is characterized by the collection of data and the development of agnostic models based only on empirical observations. We focus on the second approach only.

Data-driven approaches can be applied to multiple problems. Intuitively low domain expertise is required for the definition of the output model since the bulk of the relevant domain knowledge is automatically extracted from the data. This requires collecting significant amounts of data for the learning process. Some domain validation is still needed to define goals and validate different steps. Many examples of data-driven works can be found in the literature. In (Kargupta et al., 2004, April), a data mining approach to a vehicle's health is proposed: through a PCA, the authors identify low-dimensional clusters of nominal behaviors. When the vehicle drifts away from these clusters, a faulty situation is identified. In (Jagannathan & Raju, 2000, June), the authors collect samples of oil engine and label them based on their Remaining Useful Life (RUL). Then, an artificial neural network is trained to predict RUL from data collected by various on board sensors.

Many predictive maintenance problems need to process time series since the data comes from sensors which collect signals as they evolve in time. Possible approaches to time series data are wavelets, recurrent neural networks, and convolutional neural networks (with 1-dimensional convolutions on the time axis). All these approaches are explored in (Munikoti et al., 2019) to detect on an early stage DC motors faults. Convolutional neural networks obtain the best performance, but similar results are achieved with the other techniques.

When the phenomenon under study is cumulative, time-series data can be converted into a collection of summary statistics (e.g. mean, maximum, minimum, variance). Authors of (Giobergia et al., 2018 October) present a predictive maintenance pipeline that adopts this data transformation: here, to predict a fault in the oxygen sensor of diesel engines, the signals collected by the engine are converted into summary statistics, used for training multiple classifiers. A similar approach is used in (Susto et al., 2015), which is instead concerned with semiconductors manufacturing, more specifically, with the changing of filaments in ion implantation tools.

In this paper we discuss a complete predictive maintenance pipeline, exploiting the possible alternatives that can be pursued at each step of the process and discussing the rationale behind the decisions taken. In particular, we adopt a pipeline similar to the one presented in (Giobergia et al., 2018 October). However, we put additional focus on the signal and feature selection steps, which are particularly relevant in a constrained scenario such as the on-board data processing in the automotive setup. More specifically, we introduce an additional feature selection step that helps reduce the redundancy in the data that needs to be transferred. On top of that, we study the classification models trained in terms of robustness in terms of

hyperparameters and stability over time and with new, different data. These aspects are particularly relevant when deploying a model in a production environment, where reliability should be the main concern. By contrast, the majority of the literature does not explore alternatives and suitability of various techniques in different scenarios, thus hindering the applicability of the presented methodologies in new, likely different, scenarios. Additionally, the proposed case study (the HPF system) is a prognostics problem that we have not found to have been approached before from a data-driven perspective.

9. Conclusion

In this work, we dissected a full prognostic pipeline to study challenges and possible solutions for each step. We applied our pipeline in the context of the automotive field to identify when the HPF system drifts from nominal behavior. Given the limited computational resources on board, we showed how a thorough preprocessing step is fundamental to select only the most important signals and then features.

To study which classification algorithms could be more promising in a deployment scenario, other than evaluating classification performance we extensively analyzed the stability of the algorithms under different perspectives. The results showed that a careful evaluation of each step, and with the aid of domain experts, we successfully create a data-driven prognostic pipeline. Performed experiments on real data showed that the designed pipeline yielded accurate performance (above the required thresholds) with data coming from different driving situations and different engines.

As future work, we aim to assess the performance achievable in a deployment scenario with data coming from a non-controlled environment. Furthermore, we plan to quantify the economical benefits both for the carmaker and the car owner.

Acknowledgements

The research leading to these results has been funded by the General Motors (GM) through a research project and the SmartData@PoliTO center for Big Data technologies. We thank Michelangelo Matina for assistance performing part of the experiments.

References

Andr, M. (2004). The artemis european driving cycles for measuring car pollutant emissions. *Science of The Total Environment*, *334-335*, 73 – 84. https://doi.org/10.1016/j.scitotenv.2004.04.070.

Apiletti, D., Barberis, C., Cerquitelli, T., Macii, A., Macii, E., Poncino, M., & Ventura, F. (2018, December). istep, an integrated self-tuning engine for predictive maintenance in industry 4.0. Paper session presentation at the IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, ISPA/IUCC/BDCloud/SocialCom/SustainCom , Melbourne, Australia.

Baptista, M., Sankararaman, S., de Medeiros, I. P., Nascimento Jr, C., Prendinger, H., & Henriques, E. M. (2018). Forecasting fault events for predictive maintenance using data-driven techniques and arma modeling. *Computers & Industrial Engineering*, *115*, 41–53. https://doi.org/10.1016/j.cie.2017.10.033.

Blum, A. L., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial intelligence*, *97*, 245–271. https://doi.org/10.1016/S0004-3702(97)00063-5.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32. https://doi.org/10.1023/A:1010933404324.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794).

Cormode, G., & Veselỳ, P. (2020). A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (pp. 81–93).

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297. https://doi.org/10.1007/BF00994018.

DieselNet. Diesel fuel injection. (2009). https://dieselnet.com/tech/diesel_fi.php. Accessed 9 March 2020.

Ding, C., & Peng, H. (2005). Minimum redundancy feature selection from microarray gene expression data. *Journal of bioinformatics and computational biology*, *3*, 185–205. https://doi.org/10.1142/S0219720005001004.

Ferreiro, S., Arnaiz, A., Sierra, B., & Irigoien, I. (2012). Application of bayesian networks in prognostics for a new integrated vehicle health management concept. *Expert Systems with Applications*, *39*, 6402 – 6418. https://doi.org/10.1016/j.eswa.2011.12.027.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, (pp. 1189–1232).

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 October). Mining sensor data for predictive maintenance in the automotive industry. Conference session presentation at the IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA), Turin, Italy.

Greenwald, M., & Khanna, S. (2001). Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, *30*, 58–66.

Han, J., Kamber, M., & Pei, J. (2012). Classification: Basic concepts. In J. Han, M. Kamber, & J. Pei (Eds.), *Data Mining (Third Edition)* The Morgan Kaufmann Series in Data Management Systems (pp. 327 – 391). Boston: Morgan Kaufmann.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

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.

Jagannathan, S., & Raju, G. V. S. (2000, June). Remaining useful life prediction of automotive engine oils using mems technologies. Conference session presentation at the American Control Conference. ACC (IEEE Cat. No.00CH36334), Chicago, IL.

Jain, R., & Chlamtac, I. (1985). The p2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Communications of the ACM*, *28*, 1076–1085.

Kargupta, H., Bhargava, R., Liu, K., Powers, M., Blair, P., Bushra, S., Dull, J., Sarkar, K., Klein, M., Vasa, M. et al. (2004, April). Vedas: A mobile and distributed data stream mining system for real-time vehicle monitoring. SIAM. Conference session presentation at the SIAM International Conference on Data Mining, Lake Buena Vista, FL.

Munikoti, S., Das, L., Natarajan, B., & Srinivasan, B. (2019). Data driven approaches for diagnosis of incipient faults in dc motors.

IEEE Transactions on Industrial Informatics, *15*, 5299–5308. https://doi.org/10.1109/TII.2019.2895132.

Proto, S., Ventura, F., Apiletti, D., Cerquitelli, T., Baralis, E., Macii, E., & Macii, A. (2019, July). Premises, a scalable data-driven service to predict alarms in slowly-degrading multi-cycle industrial processes. Conference session presentation at the IEEE International Congress on Big Data, BigData Congress, Milan, Italy.

Provost, F., Jensen, D., & Oates, T. (1999, August). Efficient progressive sampling. Conference session presentation at the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, San Diego, CA.

Rabatel, J., Bringay, S., & Poncelet, P. (2011). Anomaly detection in monitoring sensor data for preventive maintenance. *Expert Systems with Applications*, *38*, 7003 – 7015. https://doi.org/10.1016/j.eswa.2010.12.014.

Renga, D., Apiletti, D., Giordano, D., Nisi, M., Huang, T., Zhang, Y., Mellia, M., & Baralis, E. (2020). Data-driven exploratory models of an electric distribution network for fault prediction and diagnosis. *Computing*, *1*, 1–13. https://doi.org/10.1007/s00607-019-00781-w.

Rohani, A., Abbaspour-Fard, M. H., & Abdolahpour, S. (2011). Prediction of tractor repair and maintenance costs using artificial neural network. *Expert Systems with Applications*, *38*, 8999 – 9007. https://doi.org/10.1016/j.eswa.2011.01.118.

Sammut, C., & Webb, G. I. (2011). *Encyclopedia of machine learning*. New York: Springer Science & Business Media.

Satopaa, V., Albrecht, J., Irwin, D., & Raghavan, B. (2011, June). Finding a" kneedle" in a haystack: Detecting knee points in system behavior. IEEE. Conference session presentation at the 31st international conference on distributed computing systems workshops. Minneapolis, MN.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003.

Suarez-Bertoa, R., Valverde, V., Clairotte, M., Pavlovic, J., Giechaskiel, B., Franco, V., Kregar, Z., & Astorga, C. (2019). On-road emissions of passenger cars beyond the boundary conditions of the real-driving emissions test. *Environmental research*, *176*, 108572. https://doi.org/10.1016/j.envres.2019.108572.

Susto, G. A., Schirru, A., Pampuri, S., McLoone, S., & Beghi, A. (2015). Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, *11*, 812–820. https://doi.org/10.1109/TII.2014.2349359.

Tutuianu, M., Bonnel, P., Ciuffo, B., Haniu, T., Ichikawa, N., Marotta, A., Pavlovic, J., & Steven, H. (2015). Development of the world-wide harmonized light duty test cycle (wltc) and a possible pathway for its introduction in the european legislation. *Transportation Research Part D: Transport and Environment*, *40*, 61 – 75. https://doi.org/10.1016/j.trd.2015.07.011.