A Machine Learning-based Distributed System for Fault Diagnosis with Scalable Detection Quality in Industrial IoT

Rodrigo Marino, *Student Member, IEEE,* Cristian Wisultschew, Andres Otero, *Member, IEEE,* Jose M. Lanza-Gutierrez, Jorge Portilla, *Senior Member, IEEE,* and Eduardo de la Torre

***Abstract*—In this paper, a methodology based on machine learning for fault detection in continuous processes is presented. It aims to monitor fully distributed scenarios, such as the Tennessee Eastman Process, selected as the use case of this work, where sensors are distributed throughout an industrial plant. A hybrid feature selection approach based on filters and wrappers, called Hybrid Fisher Wrapper method, is proposed to select the most representative sensors to get the highest detection quality for fault identification. The proposed methodology provides a complete design space of solutions differing in the sensing effort, the processing complexity, and the obtained detection quality. It constitutes an alternative to the typical scheme in Industry 4.0, where multiple distributed sensor systems collect and send data to a centralised cloud. Differently, the proposed technique follows a distributed approach, in which processing can be done eventually close to the sensors where data is generated, i.e., at the edge of the Internet of Things. This approach overcomes the bandwidth, privacy, and latency limitations that centralised approaches may suffer. The experimental results show that the proposed methodology provides Tennessee Eastman Process fault detection solutions with state-of-the-art detection quality figures. In terms of latency, solutions obtained outperform in 37.5 times the implementation with the highest detection quality, using 1.99 times fewer features, on average. Also, the scalability of the framework provides a design space where the optimal implementation can be chosen according to the application needs.**

***Index Terms*—Industry 4.0; Fault Detection; Feature Selection; Edge Computing; Machine Learning; Tennessee Eastman Process.**

I. Introduction

THE uptake of the Internet of Things (IoT) and Artificial Intelligence (AI) technologies in the industry is enabling a new industrial revolution, known as Industry 4.0 [1], [2]. This transformation aims at raising the productivity, autonomy, and efficiency of manufacturing systems. To this end, it is necessary to count on with fully autonomous real-time decision-making systems, distributed throughout the factory [1] and capable of making intelligent decisions without human intervention. Data is the fuel that powers this revolution since smart manufacturing systems use measurements gathered from the plant to monitor autonomously, control, and optimise industrial processes. IoT constitutes the appropriate technology to collect these data from networked embedded systems distributed throughout the factory, while autonomy is achieved by integrating AI techniques to drive decision-making.

One of the essential needs in industrial automation is process monitoring [3], [4]. Traditionally, process monitoring involved all the methods to track quantities directly measured from simple sensors in the plant, such as temperature or pressure [5]. However, AI techniques make it possible to extend the properties to be detected and tracked in real-time, including those depending on data provided by multiple sensors, or by a single but advanced sensor. The properties to be monitored may, therefore, depend on data analysis techniques applied to multidimensional data, either in the spatial, temporal, or frequency domains. Machine Learning (ML), considered as one of the ways to achieve genuine AI, plays a significant role when dealing with high-dimensional data.

Considering the substantial impact that faults have on the efficiency and productivity of industrial systems, fault diagnosis is one of the specific targets of process monitoring [6], [7], mainly in Industry 4.0. The detection quality of fault diagnosis systems may be improved by applying classifiers based on ML algorithms. They can be trained to know whether there is a fault or even to determine the type of fault affecting the plant, using the data provided by a set of distributed sensors [8]. Note that ML techniques are usually composed of two stages, training and inference [9], [10], being training the one demanding more intensive computing [11].

As described in [12], six design principles should be taken into account when implementing solutions for Industry 4.0: virtualisation, interoperability, decentralisation, real-time, service orientation, and modularity. ML techniques usually require high intensive computations and time. Thus, traditionally they have been executed in the cloud [13]. From an Industry 4.0 perspective, allocating all the computation in the cloud domain might not be compliant with the requirements related to the decentralisation and real-time principles. Therefore, it is necessary to explore alternatives to distribute them throughout the lower layers in industrial IoT infrastructures. A well-known approach is to place the training stage in the cloud and to move the inference stage to the network edge [14]. Near-sensor decision-making offers essential benefits, such as the reduction in communication cost and response time, together with an increase in system security and privacy [15], [16]. All these features that represent the paradigm of edge computing

R. Marino, C. Wisultschew, A. Otero, J. Portilla and Eduardo de la Torre are with the Centro de Electrónica Industrial, Universidad Politécnica de Madrid, 28006 Madrid, Spain (email: {rodrigo.marino, cristian.wisultschew, joseandres.otero, jorge.portilla, eduardo.delatorre}@upm.es).

J.M. Lanza is with the Department of Computer Science, University of Alcalá, 28871, Alcalá de Henares, Spain (e-mail: jm.lanza@uah.es).

Manuscript received February XX, 2019; revised YYYYY XX, 20XX.

are convenient for the industrial domain. In particular, reducing the response time to detect abnormal events might avoid hazardous situations, otherwise ineluctable.

Edge computing capabilities have increased significantly in the last years with the emergence of specific devices for local AI, such as Google Coral [17] or Intel Movidius [18] specialised in Deep Learning. For this reason, the integration of ML directly in the IoT infrastructure emerges now as a real possibility [19]. In the IoT scenario, the edge nodes might be defined as a networked embedded system which incorporates sensing and processing elements, and also integrates communication capabilities, primarily wireless and through the Internet [16]. Processing in the edge allows distributing the decision-making capabilities throughout the factory, in such a way that groups of neighbouring sensors can be gathered to infer a property locally using a trained ML model. Distribution would also reduce the in-factory communication costs and their associated latency when local but centralised solutions are implemented.

In this regard, an original framework, called Hybrid Fisher Wrapper (HFW) feature selection method, is proposed in this paper to distribute fault diagnosis systems in Industry 4.0, based on deploying multiple machine learning models throughout the plant, one for each node. For the detection of each particular type of fault, the proposal combines different feature selection techniques for deciding which sensors (features) are more relevant from the whole set of sensors. The relevance of a feature is given by its correlation with the predicted property compared to the entire collection of features. The purpose of feature selection techniques is to reduce the dimensionality of the measured data needed for the application while maintaining a high level of detection quality [20]. Particular emphasis is put on studying the scalability of the distributed machine learning-based systems problem as a trade-off: the more sensors are used to detect a given fault, the higher is the detection quality but also the complexity and resource utilisation of the solution. Besides the number of sensors, near-sensor applications demand low complexity ML techniques because of the constrained computing capacities in the sensing nodes, as occurs in an embedded system. Thus, the implementation of lightweight ML algorithms permits to reduce both the response time and the energy consumption of the prediction [21], [22]. Specifically, K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) with linear kernel, also called Linear SVM (LSVM), algorithms, are used to perform the fault classification, from the features (sensors) selected for each fault.

The proposed distributed agnostic ML model-based framework is applied to the Tennessee Eastman Process (TEP), which is widely used in the literature for benchmarking fault detection algorithms on chemical industrial environments [23]. Note that this method could be applied to other industrial applications without loss of generality because it does not make any assumption about the industrial process underneath.

Experimental results show that this framework provides reduced-feature ML solutions based on lightweight ML models, achieving a detection quality that meets the best solutions in the state-of-the-art. This framework can create a scalable fault-detection infrastructure where distributed sets of sensors are specialised in detecting different faults. The decentralisation and the use of a reduced set of sensors alleviate the hardware requirements for the processing nodes in the edge layer. As a result, implementations, given by this framework, stand out in terms of latency, needing only one sample to provide a prediction. Compared to the state-of-the-art, solutions provided in this paper also reduce in-factory communications since they need a lower number of sensors to predict a fault, but maintaining their prediction performance. The main contributions of this work are summarised as follows:

- A novel approach for fault-detection in distributed and scalable Industry 4.0 scenarios based on ML techniques.
- An agnostic ML model-based framework for selecting features (i.e., sensors) according to the application needs targeting resource-constrained IoT edge nodes.
- The implementation of the achieved solutions, comprising the trained ML models for the chosen sensors, in an IoT edge platform.

The rest of this paper is structured as follows. In Section II, state-of-the-art in fault detection in Industry 4.0 is reviewed. In Section III, a technical background about the ML techniques considered is provided. In turn, the specific method for feature selection and fault classification originally proposed in this work are detailed in Section IV. The TEP dataset and the performance metrics used for the evaluation of the proposed scheme are described in Section V. Experimental results applying the proposed method to TEP are provided in Section VI. Finally, concluding remarks are provided in Section VII.

II. State-of-the-Art

TEP provides a real-world scenario where traditional sensors (e.g., temperature or pressure) and actuators control an industrial process. Moreover, it does not only have simulations that can model the process, but these simulations can generate a pre-defined set of system faults [24]. Therefore, TEP is extensively used as a fault detection benchmark [25].

Researchers have studied different techniques for fault detection. Data-driven techniques, including statistics and ML, are the solutions that achieve better performance [26]–[28]. In the statistical approaches, the application of time series provides a high detection quality. This type of statistical solutions requires a centralised analysis of the evolution of sensor data over time so that they might produce high communication loads or high system time response in comparison to other strategies, such as ML approaches. Moreover, it is also required a significant number of previous samples to produce a prediction. These facts make unfeasible applying them under real-time constraints when there is a considerable delay between samples, as happens in TEP [24]. Differently, ML techniques do not require a high volume of data from each sensor during inference, meaning that the current sensor values could be enough to identify the faults without waiting for future data, so improving the time response of the system [29].

In ML-based fault detection methods, the main goal is to infer whether there is a fault inside the system. However, in

many industrial applications, it is necessary to detect more than one fault. A common strategy is to use a single multi-class classifier to identify the whole set of faults. There are two main types of multi-class classifiers: model-based, such as SVM ([30], [31]), and neural network-based [32]. Comparing both approaches, authors in [32] reported similar performance using neural networks than [31], achieving a detection quality above 90%. This multi-class conception has an important shortcoming, which makes it not suitable to be accomplished in Industry 4.0. This shortcoming is related to the fact that all the sensors in the plant power a unique classifier. This fact creates a centralised scheme where data is gathered to the same point in the network, conflicting with the decentralisation requirement [33].

One possibility to decentralise the system is to consider single-class classifiers for identifying a particular fault, instead of having a global multi-class classifier. Moreover, fault-specific classifiers usually provide a higher detection quality compared to the multi-class ones, as reported in [34], [35]. Nevertheless, most of the fault-specific models considered in the literature adopted a centralised strategy powered by all the sensors in the system, conflicting again with the decentralisation requirement [36], [37]. As a response, different strategies for selecting which are the most suitable sensors for a specific fault-detection model were proposed in the literature. For instance, the authors in [38]–[40] integrated SVM techniques to select the sensors in the TEP use case, resulting in a high detection quality. However, some SVM approaches, as in [39], [40], were not able to provide any solution for three particular faults from the whole set of faults, which were also reported in the literature as the most complicated TEP faults to identify. Moreover, the selected SVM techniques also demand high computational capabilities because of using non-linear kernels, such as gradient radian basis kernel [39]. On the other hand, the authors in [41] selected the sensors using an ML technique based on Neural Networks (NNs) called Extreme Learning Machine (ELM), requiring even more computational resources than SVM.

On this basis, the challenge in this paper is to create a decentralised fault diagnosis system for the TEP use case based on single-fault lightweight classifiers and selecting a reduced set of sensors. To this end, a distributed agnostic ML model-based framework is created. The method proposed should be able to determine which set of sensors in the TEP use case fulfils the decentralisation requirement, also attending to requirements in detection quality and computational capacity. A comparative analysis of state-of-the-art solutions is performed. As a result, the solutions found by the proposal outperforms in 37.5 times the best solution in detection quality within the state-of-the-art, using 1.99 times fewer sensors, on average. Furthermore, the scalability of the framework gives a space of solutions attending the number of sensors and detection quality as a trade-off. Thus, a designer, the network architecture developer, could choose the most suitable solution according to the application needs.

III. Technical Background

ML is a scientific discipline that proposes the use of algorithms and mathematical models to make computers capable of performing specific tasks without being explicitly programmed for it. Programming is substituted by the construction of mathematical models that, after a data-driven learning stage, can infer patterns from input data [42]. Data-driven permitted ML to be widely applied in multiple domains, such as medicine, biology, economy or engineering [43].

A generic ML system consists of two stages: training and inference. Training tunes the parameters of the ML model through a data-driven optimisation algorithm. Inference performs a prediction from an input sample based on the previously trained model. From the data point of view, there are two main strategies for learning: supervised learning and unsupervised learning. In supervised learning, the data used for training the model include the value of the property (or label) to be estimated for each input sample. In unsupervised learning, the value of the properties is unknown. Unsupervised learning offers some practical advantages, such as it does not require a human operator to label input samples and allows the model to evolve when the properties of the inputs vary with time. However, unsupervised learning cannot distinguish between specific classes for a property required in an application. Therefore, the authors of the present paper only focus on supervised methods because of the requirement of detecting particular faults.

Supervised learning can be divided into classification and regression methods. In classification, the inferred property is a categorical unordered variable (from a discrete and finite set) to which input data belong, for instance, a fault. If a fault is detected, it must be classified from a list of potential failure situations, fitting with the usage of classification methods in supervised learning.

Usually occurs that input samples are multidimensional, being each independent component a feature. Then, a critical step when designing the classifier is to determine which features from the input sample are significant to discriminate between classes. Relevant features must be extracted from process observations guaranteeing they are directly correlated with classification categories. The relevance of a feature can be qualified as weak or strong, depending on how it correlates with other features and the impact it has on the classifier performance. Irrelevant or redundant features, which do not contribute to distinguishing between classes, must be removed since they introduce noise in the prediction [44], [45]. This preprocessing stage results in a useful dimensionality reduction of the feature space.

There are two main approaches for feature preprocessing: feature extraction and feature selection [46], [47]. In feature extraction, input features are combined and projected into a reduced feature space, as in Principal Component Analysis (PCA), which is a widely used technique in the literature [31], [48], [49]. As combining all the features implies a high response time, energy consumption, or communication overheads, it prevents its implementation in the edge [50]. On the contrary, feature selection consists in selecting a subset

of features from the initial feature space to feed the ML model. This fact results in a lower computational complexity when compared with feature extraction, also having other significant benefits, such as scalability, generalization, and understandability. These properties are essential to generate cost-effective ML models for the edge layer [45], [51]. The rest of this section discusses some technical background about both feature selection and classification techniques.

A. Feature Selection Techniques

Feature selection requires assigning an identifier of relevance to every feature subset under evaluation, within the selection process. Depending on the strategies followed for the assessment, different feature selection techniques are defined. Techniques of relevance for this work are summarised next.

Filter methods consider ML-independent statistical tests to measure the correlation of the features with the properties to predict. Wrapper methods evaluate the performance achieved by the ML technique for all the possible subsets of features drawn from the initial list [52], [53]. In this paper, a hybrid approach is proposed mixing filter and wrapper strategies, trying to keep the lower computation requirements of the filter strategies and the higher performance of the wrapper algorithms. Traditionally, the hybrid approaches based on filter and wrapper methods, first, remove the irrelevant features from the filter point of view, and then a wrapper looks for the subset of features which produces the best performance metrics [45]. A sequential forward selection wrapper method, used in this paper, usually starts with an empty dataset and adds a feature in each iteration. Traditionally, its stop criterion consists in comparing the performance metrics of the current iteration to the best combination of the previous iteration. So, this general approach focuses on the performance metrics rather than the number of features selected. Differently, the proposal in this paper involves taking the number of features into consideration besides the performance metrics. Hence, instead of applying a stop criterion based on the performance metrics, the stop criterion is based on the maximum number of features (sensors) the system should have, previously defined by a designer in charge of the industrial network. Thus, the proposal might obtain the combination of features which provides the best performance metrics for a particular number of features. Furthermore, to enhance the design process, this proposal also provides the exploration of solutions (where a solution consists in a subset of features which has the best performance metrics for a particular number of features) improving the decision-making during the system design process.

In particular, the Fisher-Score (FS) and the Sequential Forward Selection (SFS), with a modification of the stop criterion, have been selected as the filter and wrapper algorithms, respectively. Both methods are described below.

1) Fisher-Score: It evaluates the relevance of each feature analysing (i) the similarity of the sample values for a particular property, and (ii) the divergence for the other property values [44].

This method evaluates each feature individually based on its mean and its variance values according to different rules, for instance, the variance of a feature for a particular class value or the variance for the whole set of classes. As a result, the FS filter method creates the vector FSV, which expresses the relevance value for each feature as given by

$$FSV = [FS_1, FS_2, \ldots, FS_n]^T, \tag{1}$$

where $FS_j \in FSV$ is the fisher-score value of the j-th feature, with $j \in 1, \ldots, n$, being n the number of features in the dataset. The higher the FS value of a feature is, the higher the relevance of that feature. It is worth noting that the position in the FSV corresponds to the ID of each feature. Then, features can subsequently be sorted attending to their significance.

2) Sequential-Forward Selection: It is a search-based method, which selects and includes features in the ML model from a pool of candidate features over iterations. The procedure starts with an empty set of selected features and a pool full of candidates. Then, the algorithm trains and evaluates an ML model for each of the features in the pool, selecting and adding to the set of features chosen the one that meets a given criterion, e.g., providing the highest detection quality in the ML model trained. All the already selected features are used when training the model in subsequent iterations. This iterative method ends when a stopping condition is met, e.g., a detection quality level or a number of features selected [45].

B. Classification Techniques

As introduced before, this paper focus on applying two lightweight supervised classifiers: KNN and LSVM. Both algorithms are used as binary classifiers instead of multi-class to keep complexity low while meeting the decentralisation and scalability requirements. Moreover, the binary approaches improve detection quality compared to multi-class ones, as will be discussed in Section VI-D. KNN is a classification technique which is widely used in the IoT domain due to its complexity and computational efficiency [54]. The trade-off between low execution times and detection quality makes it suitable for our use case. SVM usually has a higher detection quality, but its computational cost is also higher. In this use case with energy-constrained edge nodes, it is recommendable to implement its linear kernel approach (LSVM) for maintaining high detection quality and high computational efficiency [55]. Further details on the two classification algorithms are provided below.

1) Support Vector Machine: SVM identifies decision boundaries in the data space, which are hyper-planes that represent the relationship between features and properties [56]. Different mathematical models (kernels) can be used for generating decision boundaries, such as linear, quadratic, cubic, or Gaussian.

Once the kernel is chosen, the training phase is executed, typically offline, to obtain the hyperparameters of the model. The inference stage involves the calculus of the position of a new sample in the hyper-plane given by the trained kernel equation. As a result, the situation regarding the hyper-plane will indicate the class value of the sample. As discussed before, the authors consider LSVM with a binary classification approach. Thus, there is only one hyper-plane which identifies whether a sample is within a particular fault or not.

2) K-Nearest Neighbors: KNN relies on a learning approach, known as lazy learning, where there is no real training stage. Instead, KNN searches in a ground truth dataset the k nearest neighbours to a given sample based on a distance metric, such as Euclidean, Minkowski, chi-square, Mahalanobis, or cosine. Following the lightweight approach in this paper, the authors consider the Euclidean distance to reduce the computational load.

The inference stage takes a new sample and calculates the Euclidean distance between the sample and each of the samples in the ground-truth dataset. The resulting distances are sorted in ascending distance value, selecting the first k samples. This set of k samples is passed to a voting algorithm, which infers the class for the new sample based on the classes to which the k samples belong from the ground truth dataset. As discussed before, the authors consider a binary classification approach. Thus, the voting algorithm establishes whether a sample is within a specific fault or not.

IV. Scalable and Distributed Fault Detection

The proposed methodology for implementing scalable and distributed fault detection systems relies on two foundations. The first one is to consider binary classifiers trained individually to detect the presence of each type of fault, instead of a single multi-class detector for all potential faults in the plant. This strategy usually results in that each binary classifier does not require the measurements provided by all the sensors. In this way, a reduced number of sensor signals are selected to feed each fault-specific classifier. This strategy allows distributing the fault detectors throughout the plant, near to the sensors associated with each fault. This approach reduces the latency, increases the robustness of the system, and reduces the communication overhead. The second foundation is the use of the originally proposed Hybrid Fisher Wrapper (HFW) feature selection method to identify the set of sensors needed to detect each type of fault efficiently. Thus, the information provided by each sensor is considered as a feature from an ML-based classification perspective. Then, HFW identifies the subset of sensors required for each specific classification problem.

The remainder of this section describes the methodology, the HFW selection method proposed and discusses some critical aspects related to the parameterisation in HFW.

A. Methodology Overview

As introduced before, ML requires two stages: training and inference, which are performed offline and online, respectively. In this specific use case for fault detection, training is conducted in a regular workstation and consists in obtaining the different individual fault detectors by applying the HFW method. Thus, training is considered as an offline designing stage, providing the fault detection systems to be deployed in the plant. On the other hand, the inference is performed online in a constrained IoT edge device. It consists in running the online fault detection systems, which were previously designed during the training stage. That means that the computation in the edge node is significantly lighter than the one done in the workstation, running HFW.

Before starting the training stage, it is required to select or create a labelled dataset with information about all sensors in the system and possible faults. Each sample in the dataset contains the single information provided by all the sensors and information about existing faults at this time. Next, the parameters of HFW should be fixed, as will be discussed in Section IV-C. These parameters include the ML models to be considered or the maximum number of features to be explored. Then, HFW is executed, generating an exploration of different subsets of features regarding the parameters selected in the previous parameterisation. As a result, HFW provides a space of solutions, where a solution consists in a trained machine learning model to be executed with a subset of features.

The resulting space of solutions obtained during the training can be plotted as a trade-off. For instance, showing the number of features versus fault detection quality for each solution found. A designer will consider this trade-off to decide on the selection that fits better to the needs of the application. Once one solution is selected, the inference part of the trained ML model should be implemented in one of the IoT nodes. Also, this node will collect data from the sensors selected and infer the occurrence of a specific fault.

B. Hybrid Fisher Wrapper Description

The HFW selection method is a hybrid approach combining both FS and SFS techniques. The algorithm requires a dataset in the form of a matrix $X \in \mathbb{R}^{m \times n}$ with m samples for the n sensors in the plant and a vector $L \in \{0,1\}^m$ with the binary labels defined for each of the samples. As Algorithm 1 shows, HFW is composed of two stages.

Based on the FS metric, the first stage computes the relevance of each sensor (a feature from an ML point of view), as given by Equation (1). It consists in sorting X in descending order of relevance, resulting in $R \in \mathbb{R}^{n \times 1}$ (lines $1-2$), which represents the ranked set of features by their IDs. The second stage explores the search space following an SFS approach (lines $3-16$). To this end, it considers the previously defined ranked R, as well as two parameters p and h. Let p be defined as the maximum number of sensors to be included in the final model. Let h be defined as the maximum number of features to be explored in the ranking defined before. Further discussion about assigning values to p and h is included in Section IV-C.

In the second stage, the algorithm looks for the best combination of z sensors over iterations, with z varying from 1 to p. Let $M_z \subseteq X \in \mathbb{R}^{z \times m}$ be the dataset generated based on R for the z sensors selected during the z-th iteration. In the z-th iteration, there are available $n - z - 1$ sensors for choosing one of them. Traditional wrapper methods will explore the whole set of sensors to obtain the proper one. However, the proposed algorithm, enhanced by the ranked R from FS method, studies the detection quality of the system (see Section V-B) while adding a sensor to be selected among the first h sensors in R, using j to index the sensors in R, with $j \in 1 \ldots \min(h, size(R,1))$. Note that $size(R,1) \in 1 \ldots n$ refers to the size of the first dimension of R, which varies during the execution of the algorithm. Thus, when $z = 1$, the algorithm selects the best sensor among the first h sensors

Algorithm 1 Hybrid Fisher Wrapper.

Require: X, L, p, h
Ensure: $M_z \ \forall z \in 1 \dots p$
1: $FSV \leftarrow fisherScore(X, L)$
2: $R \leftarrow SortFisher(FSV, descending)$
3: $M_0 \leftarrow \emptyset$
4: **for** $z = 1;\ z \leqslant p;\ z++$ **do**
5: $q_{th} \leftarrow 0$
6: **for** $j = 1;\ j \leqslant min(h, size(R))\ ;\ j++$ **do**
7: $X_{wf} \leftarrow M_{z-1} \cup \{R_j\}$
8: $q_{metric} \leftarrow evaluateModel(X_{wf}, L)$
9: **if** $q_{metric} > q_{th}$ **then**
10: $q_{th} \leftarrow q_{metric}$
11: $mark \leftarrow R_j$
12: **end if**
13: **end for**
14: $M_z \leftarrow M_{z-1} \cup mark$
15: $R \leftarrow R - R_{mark}$
16: **end for**

based on an evaluation process. When, $z = 2$, the algorithm considers the sensor selected before, in $z = 1$, and searches another sensor among the h sensors to be combined with.

For a specific value of z and j, the evaluation process is as follows (lines $6 - 13$). The algorithm generates a dataset (X_{wf}) combining the data from the previous selection process (M_{z-1}) with the data $R_j \in \mathbb{R}^{1\times m}$ for the j-th sensor in R (line 7). That means that to select the z-th feature (sensor) of the model, the model is trained and tested $\min(h, size(R, 1))$ times. Thus, in each iteration of j, the dataset X_{wf} is considered to train and test a model by 10-fold cross-validation, obtaining a detection quality metric q_{metric} (line 8). To choose which feature fits better to the application, a detection quality threshold (q_{th}) is updated during the execution of the algorithm, starting by zero for each exploration of z. Thus, if the detection quality of the system increases in the j-th iteration, then the q_{th} is updated, and R_j is marked as a candidate sensor to be selected in this iteration (lines $9 - 12$). After the $\min(h, size(R, 1))$ iterations, HFW adds the selected feature to M_z and remove it from R (lines $14 - 15$). Next, HFW starts the search for the next value of z, if it is lower or equal than p. Otherwise, HFW ends and the resulting space of solutions are in $M_z, \forall z \in 1, \dots, p$.

C. Hybrid Fisher Wrapper Parameterisation

As discussed before, there are some design parameters (p, h, and the type of ML classification algorithm) to be configured when applying the HFW algorithm. These parameters affect the detection quality of the final system, as well as the training and inference computing costs. Note that Table I shows the parameters which shape the HFW method.

In terms of feature selection, the p value indicates the maximum number of features explored for obtaining an ML model. This value also affects the maximum number of features (sensors) that will feed the final ML model. This parameter impacts on the HFW computing time and the detection quality of the system. In high-dimensional applications, this parameter might define whether the feature selection algorithm, particularly HFW, can provide a final solution due to the training time of the models explored. So, the system designer should select this parameter according to the application needs. It also has practical effects on the distribution of the measuring nodes throughout the factory.

The type of ML classification method considered is also relevant, although it was not directly included in the parameters of Algorithm 1. The type of classification model should be the same that it will be considered in the final system to avoid biasing the conclusions obtained during the search. Thus, it is crucial first to identify the classification models which perform better for the application, while considering computing cost limitations. As discussed before, the authors opted for lightweight binary classification algorithms because they are ready for near-sensor implementation.

The value for h determines how the search process is done over the ranked set of sensors (R), affecting both the training computing time and the detection quality of the system. Thus, if h equals 1, then the best sensor, according to the Fisher ranking, is always selected for each iteration of z. That means that HFW is running a pure Fisher algorithm, which does not ensure a high-quality solution to the problem. On the contrary, if h equals n, then HFW evaluates all the available sensors for each iteration of z. That means that HFW is running a classic SFS algorithm, which provides better quality solutions that the Fisher algorithm, but it is highly time-consuming. Thus, it is essential to find a trade-off for this value. To this end, the authors propose a general methodology as follows.

This general methodology consists in running HFW configured as a Fisher algorithm for all the available sensors, i.e., $h = 1, p = n$. As a result, the algorithm will provide n solutions generated during the n iterations of z, i.e., $M_z, \forall z \in 1 \dots n$. Next, it should be plotted the detection quality for the n solutions obtained. The position (regarding the number of sensors needed) of the local maximum detection quality in the trend will provide an indicative value for h, which later the designer could redefine based on experience. This methodology will be considered during the experimental analysis in Section VI.

The result of this parameterisation affects directly to the performance of the HFW algorithm. This performance has been studied in terms of computational complexity. The HFW complexity is bounded by $O(p \times \min(h, \text{size}(R)) \times t)$, where t refers to the computational effort required to train and evaluate the ML model. Thus, the t effort will have a hard dependence on the ML model considered and the data used to perform the training task. In the literature, the t effort for LSVM and KNN were defined as $O(N_{sv}^3 + N_{sv}^2 + N_{sv} \times s \times f)$ and $O(s \times f)$, respectively, where f is the number of features, s is the number of instances, and N_{sv} is the number of support vectors [57], [58].

V. Experimental Setup

This section describes the TEP benchmark and the ML performance metrics for assessing the proposed methodology.

A. The Tennessee Eastman Process

TEP is a well-established simulation benchmark of a real industrial chemical process designed by the Eastman Chemical

Table I: HFW Parameters summary.

Parameters	Description
p	Number of features chosen
h	Number of features evaluated during the j-th iteration
$model$	ML model used
X	Dataset input data
L	Dataset labels
R	ID vector of the ranked set of features
M_z	Dataset generated after the z-th iteration
X_{wf}	Dataset created to test the features provided by h
n	Number of features of the dataset
m	Number of instances of the dataset

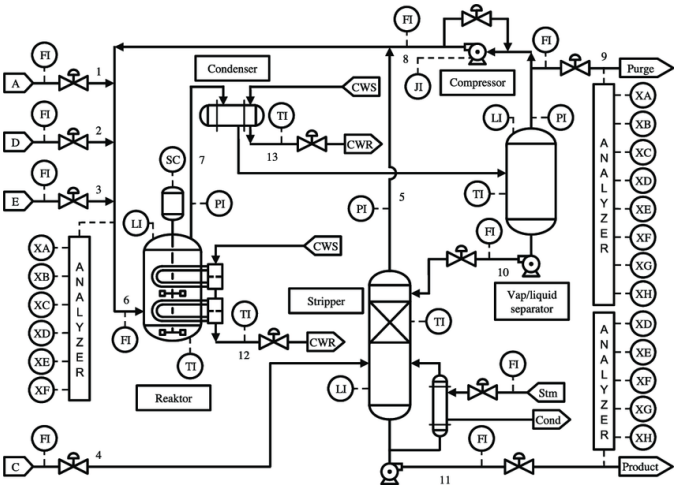

Figure 1: Tennessee Eastman Process flowchart [62].

Company [59]. To adapt this chemical process to the ML scenario, authors in [60] developed a dataset[1] which is used in this work. The challenge behind TEP is to identify the faults that occur in the system. Due to its complexity, its goal is to promote research in fault detection and process monitoring techniques [31], [61].

The chemical process emulated in TEP is fed by several gaseous reactants (A, C, D, E) and an inert component B. The outcomes of the process are three products (F, G, H) presented in the liquid state (see Fig. 1). Forty-one sensors and twelve actuators control this chemical process. Unlike the state-of-the-art, our work only considers data from the sensors, discarding the information about the state of the actuators. It must be noted that in real industrial scenarios, the actuators do not produce measurements. Instead, they receive signals from the control system, which is usually centralised and not accessible from the distributed processing nodes, where the fault-detection algorithm is executed. Table II shows an IDentification (ID) and a description for each sensor in TEP.

The TEP database in [59] was generated through simulation in two different experiments, one for 24 hours and the other for 48 hours, where different types of faults were introduced. The detailed description of each fault was extracted from [24] and is shown in Table III. The failure types or process disturbances are, on the one side, variations on the temperature, pressure, and composition of the different reactant elements (e.g., faults 1-3, 6-10). On the other side, there are failures produced in the reactors and condensers caused by variations in the temperature of the cooling water or failures, which affect water valves that feed them (faults 4,5,11-15). Moreover, five unknown source failures can be found (faults 16-20). During the simulation, each sensor in the system acquires a sample every 3 minutes, resulting in 480 and 980 samples for each sensor in the 24 and 48-hour experiments, respectively. The strategies followed in each experiment are different. In the 24-hour version, a fault is introduced after each hour of operation. In the 48-hour version, a fault is introduced after every eight hours of operation.

[1]https://github.com/camaramm/tennessee-eastman-profBraatz

Table II: Sensors considered in TEP [63].

ID	Description
F1	Feed flow component A (stream 1) in kscmh
F2	Feed flow component D (stream 2) in kg/h
F3	Feed flow component E (stream 3) in kg/h
F4	Feed flow component A/B/C (stream 4) in kscmh
F5	Recycle flow to reactor from separator (stream 8) in kscmh
F6	Reactor feed rate (stream 6) in kscmh
P7	Reactor pressure in kPa gauge
L8	Reactor level
T9	Reactor temperature in $^\circ$C
F10	Purge flow rate (stream 9) in kscmh
T11	Separator temperature in $^\circ$C
L12	Separator level
P13	Separator pressure in kPa gauge
F14	Separator underflow in liquid phase (stream 10) in m^3/h
L15	Stripper level
P16	Stripper pressure in kPa gauge
F17	Stripper underflow (stream 11) in m^3/h
T18	Stripper temperature in $^\circ$C
F19	Stripper steam flow in kg/h
J20	Compressor work in kW
T21	Reactor cooling water outlet temperature in $^\circ$C
T22	Condenser cooling water outlet temperature in $^\circ$C
XA	Concentration of A in reactor feed (stream 6) in mol %
XB	Concentration of B in reactor feed (stream 6) in mol %
XC	Concentration of C in reactor feed (stream 6) in mol %
XD	Concentration of D in reactor feed (stream 6) in mol %
XE	Concentration of E in reactor feed (stream 6) in mol %
XF	Concentration of F in reactor feed (stream 6) in mol %
YA	Concentration of A in purge (stream 9) in mol %
YB	Concentration of B in purge (stream 9) in mol %
YC	Concentration of C in purge (stream 9) in mol %
YD	Concentration of D in purge (stream 9) in mol %
YE	Concentration of E in purge (stream 9) in mol %
YF	Concentration of F in purge (stream 9) in mol %
YG	Concentration of G in purge (stream 9) in mol %
YH	Concentration of H in purge (stream 9) in mol %
ZD	Concentration of D in stripper underflow (stream 11) in mol %
ZE	Concentration of E in stripper underflow (stream 11) in mol %
ZF	Concentration of F in stripper underflow (stream 11) in mol %
ZG	Concentration of G in stripper underflow (stream 11) in mol %
ZH	Concentration of H in stripper underflow (stream 11) in mol %

B. Machine Learning Performance Metrics

Performance metrics are required to evaluate the detection quality of the ML models for a classification problem, which corresponds in this case with the fault detection problem. To this end, the authors focus on three usual metrics within the field, sensitivity (svt), specificity (spc), and geometric mean

Table III: Faults considered in TEP [63].

Fault N°	Description	Fault type
1	A/C feed ratio	Step
2	B composition	Step
3	D feed temperature	Step
4	Reactor cooling water inlet temperature	Step
5	Condenser cooling water inlet temperature	Step
6	A feed loss	Step
7	C header pressure loss	Step
8	A,B,C feed composition	Random variation
9	D feed temperature	Random variation
10	C feed temperature	Random variation
11	Reactor cooling water inlet temperature	Random variation
12	Condenser cooling water inlet temperature	Random variation
13	Reaction kinetics	Slow drift
14	Reactor cooling water valve	Sticking
15	Condenser cooling water valve	Sticking
16	Unknown	N/A
17	Unknown	N/A
18	Unknown	N/A
19	Unknown	N/A
20	Unknown	N/A
21	The valve for Stream 4	Constant position

($gmean$), which are expressed as

$$svt = \frac{tp}{tp + fn}, \tag{2}$$

$$spc = \frac{tn}{fp + tn}, \tag{3}$$

$$gmean = \sqrt{svt * spc}, \tag{4}$$

where tp denotes the number of true positives, tn the number of true negatives, fp the number of false positives, and fn the number of false negatives.

According to these expressions, sensitivity quantifies the proportion of positive detections which are correctly identified regarding the total. A high sensitivity value indicates that the system has a low ratio of false negatives. In turn, specificity quantifies the proportion of negative detections which are correctly identified regarding the total. A high specificity value shows that the system has a low proportion of false positives. The geometric mean provides a trade-off metric between sensitivity and specificity, being a pertinent metric to evaluate the quality of the ML system [64], that is, the detection quality.

These performance metrics are usually calculated during the testing stage, after performing the training, to quantify the quality of the model. The authors of this paper consider the accepted k-fold cross-validation strategy [42], [65], [66] to perform the testing stage. This cross-validation strategy consists in splitting the dataset into training and testing subsets in a systematic way. That is, the dataset is randomly split into into k subsets $Q = \{q_1, \ldots, q_k\}$ of similar size, being k defined by the designer. In the i-th iteration, the training stage is computed using all the subsets in Q, but q_i, with $i \in 1, \ldots, k$. The testing stage is performed by inferring over q_i and calculating the performance metrics, on average. In this proposal, the authors opted for using an accepted value in the literature for k equalling 10.

VI. Experimental Results

The fault detection methodology proposed in this paper was applied to TEP, using the previously described setup. Experimental results are provided and analysed in the following subsections. First, a discussion on the HFW parameterisation is provided. Then, the proposed methodology is applied to the TEP use case. The same experiments are repeated using a multi-class classifier, instead of a binary one. The best solutions obtained are implemented in an IoT edge platform, measuring their execution performance. Finally, the results obtained with both approaches are compared among them and with the state-of-the-art.

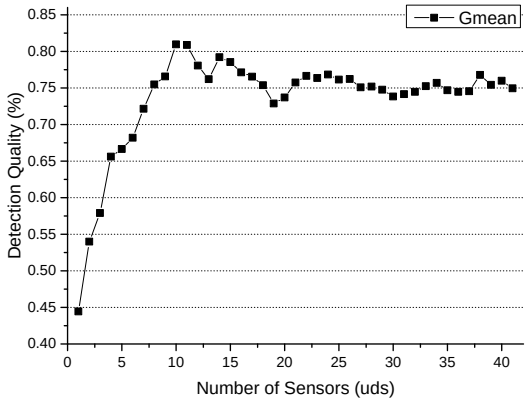

Figure 2: h parameter exploration for the single-class KNN and fault 3.

A. HFW Parameterisation in TEP

This subsection reports an example of the selection of the p and h parameters in HFW for the TEP use case.

The network architecture designer defines the parameter p according to the application needs. In this use case, the maximum number of features corresponds to the number of sensors used for identifying the faults. Aiming at minimising the model complexity for the inference, a maximum number of sensors of 10 ($p = 10$) is assumed. The criterion was keeping p below the 25% from the total number of sensors in the dataset, which is 41. The h parameter is set following the procedure described in Section IV-C. For instance, suppose the h parameter should be obtained for the KNN model and fault 3 of TEP. Then, an exploration is performed by HFW for $h = 1$, $p = n = 41$, and $model = KNN$ (and $K = 3$). This exploration results in Fig. 2, which shows the quality of the prediction (according to $gmean$) for the different numbers of sensors. Analysing this figure, the maximum quality is obtained for ten sensors. Therefore, h is fixed to ten for the single-class KNN and fault of type 3.

B. Single-class HFW in TEP

This subsection describes the exploration performed by the single-class HFW approach proposed. Thus, the authors of this paper create a model per fault, being each model able to identify whether there is a particular fault in the TEP use case or not. In this regard, the TEP database is transformed into 21 different databases, as explained in Section V-A. As reported

Table IV: h parameter for each single-class ML model and fault.

	h value of each fault																				
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
LSVM	11	20	8	18	15	10	15	12	15	14	14	16	9	22	17	17	18	17	23	8	20
KNN	15	21	10	17	10	18	11	13	14	13	10	9	10	22	20	14	9	16	16	13	17

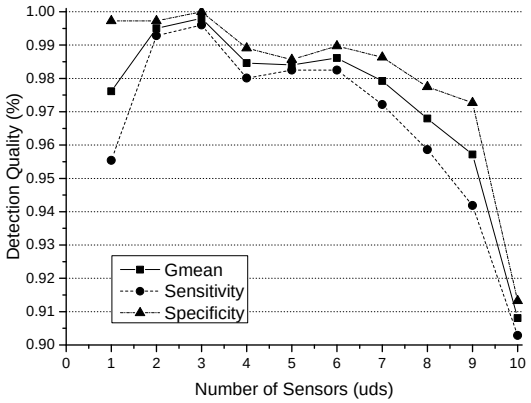

Figure 3: Feature space exploration for fault of type 3, with $p = 10$, $model = KNN$, and $h = 10$.

before in Section VI-A, the p parameter was set to 10, the ML models were KNN and LSVM, and the h parameter was established individually for each fault and ML model (resulting in 42 different explorations), as Table IV shows.

Once the parameterisation was done, the next step is to execute the single-class HFW for each ML model and fault in TEP. Thus, for instance, Fig. 3 shows the feature space exploration during the selection of a range of sensors (from 1 to h) to detect the fault of type 3 with KNN. Note that as was established in Section IV-B, HFW considers an accumulative strategy, where past decisions will affect future ones. That means that if the sensor selected during the first iteration was ZF, then the set of sensors selected during the second iteration of the algorithm will always include the sensor previously selected, i.e., ZF in this example.

As the algorithm will report h sets of sensors to detect a given fault using an ML model, it is required to define a strategy to select the best configuration among the others. Thus, the authors propose to use a threshold based on detection quality (in terms of $gmean$) with a value of 99%. This threshold should be adjusted by the industrial network designers to meet other application-specific requirements, for instance, 90%. In the example in Fig. 3, the best performance was obtained using two and three sensors. The authors also include a criterion based on penalising more the false negatives rather than the false positives. This fact is because the ML system might accomplish to detect, as much as it can, the faults occurred in TEP. This second criterion results in that the solution with the highest sensitivity will be selected, i.e., the solution with three sensors in Fig 3.

Table V summarises the implementations selected, based

Table V: Selected implementations for the single-class KNN model.

Fault	**h**	**Sensors**	**Sensors ID**	**Gmean(%)**	**Svt(%)**	**Spc(%)**	**Time(s)**
1	15	4	F1, P16, F19, YC	99.65	99.36	99.93	8.07
2	21	4	ZF, F10, T18, F19	99.35	99.04	99.66	10.93
3	10	3	ZF, ZH, ZG	99.80	99.60	100.00	3.83
4	17	2	ZF, ZG	99.80	99.60	100.00	4.12
5	10	2	ZD, ZF	99.96	99.92	100.00	2.39
6	18	1	ZE	99.92	99.60	100.00	2.32
7	11	2	ZF, ZH	99.82	99.92	99.73	2.66
8	13	3	ZD, ZF, ZH	99.82	99.92	99.73	4.81
9	14	5	ZH, T18, J20, F19, ZF	93.35	92.52	94.19	8.81
10	13	1	ZF	98.03	96.10	100.00	1.50
11	10	2	ZF, ZG	99.60	99.92	100.00	2.54
12	9	4	ZG, P7, F19, ZD	98.81	97.69	99.93	4.59
13	10	2	ZG, ZH	99.96	99.20	100.00	2.40
14	22	2	ZF,ZD	99.96	99.92	100.00	5.31
15	20	2	ZF, ZG	99.66	99.60	99.73	4.79
16	14	3	ZF, ZG, ZH	99.55	99.92	99.18	5.15
17	9	2	ZD, ZH	99.82	99.92	99.73	2.14
18	16	4	ZE, J20, F19, P7	96.97	96.34	97.61	7.99
19	16	3	ZF, ZH, ZG	99.82	99.92	99.73	5.68
20	13	5	ZH, J20, T18, F19, ZF	97.71	96.26	99.18	7.99
21	17	8	L8, F19, P16,J20, T11, ZH, T18, ZD	92.17	91.32	93.03	18.05

on the previously discussed strategy, for each fault using the binary KNN classifier. This table shows the performance metrics ($gmean$, svt, and spc), the number of sensors used, as well as their corresponding names according to the TEP nomenclature stated before in Table II. Analysing this table, it is obtained that the binary KNN classifier has a high-detection quality, surpassing the 90% in the three metrics. The lowest detection quality, with a 92.17% of $gmean$, was in fault 21, which also has the highest number of sensors (8). The rest of the faults usually needs less than 4-sensors to be detected. Furthermore, it should be noticed that ZF and ZH sensors are especially relevant because they appear in most solutions.

In turn, Table VI summarises the implementations selected using the binary LSVM classifier, showing a similar type of information as before. Results are significantly different from the ones obtained by KNN, even leading to have no solution for some faults. This latter fact is due to a lack of convergence during the training of LSVM. Thus, there are only three solutions whose results are higher than 90% (faults 1, 2, and 6). These solutions have a difference in their quality value and sensitivity of less than 1% compared to the same binary KNN solutions. Since the LSVM classifier is lighter than the KNN classifier in terms of computation, it is preferred for a similar quality of detection.

Tables V and VI also show the times required for the execution of the HFW algorithm. Experimental results indicate that there is a strong dependence on the ML model selected. The LSVM model needs more time than the KNN model even for lower p and h values (see faults 1 and 2, as an example). Besides, h and p also impact the training time, as the complexity analysis introduced in Section IV-C indicates. In Table V, for $p = 2$, it is shown that a higher h value implies an increment of the HFW time. This situation occurs for the p parameter as well (e.g., in Table VI, for any fault with $p = 5$). Note that these values are obtained using an Intel i7 processor with 8 GB of RAM, running in Matlab over Windows 10 OS.

According to the result obtained for KNN and LSVM, the final fault detection system for TEP will use the LSVM solutions for faults 1, 2, and 6, and the KNN for the others. Thus, this methodology provides a framework to observe how

Table VI: Selected implementations for the single-class LSVM model.

Fault	h	Sensors	Sensors ID	Gmean(%)	Svt(%)	Spc(%)	Time(s)
1	11	3	F1, P16, YF	99.11	99.89	99.32	25.39
2	20	3	F10, YB, XB	98.90	98.57	99.32	46.57
3	8	3	T21, ZG, F19	58.53	39.09	75.20	35.73
4	18	-	-	-	-	-	-
5	15	5	XA, YD, F6, L15, F4	53.82	0.00	100.00	109.59
6	10	1	F1	99.42	99.32	99.52	6.79
7	15	3	F19, P16, L8	59.59	12.50	100.00	60.92
8	12	3	XA, XB, ZD	71.10	37.42	100.00	50.92
9	15	-	-	-	-	-	-
10	14	3	YG, XB, YE	54.78	18.79	85.65	55.96
11	14	-	-	-	-	-	-
12	16	2	F19, YA	72.17	40.84	99.04	42.85
13	9	6	F19, F2, YD, P13, ZH, T11	78.89	58.35	96.51	75.68
14	22	-	-	-	-	-	-
15	17	-	-	-	-	-	-
16	17	-	-	-	-	-	-
17	18	5	T21, T11, F3, YC, YD	95.14	91.56	98.22	94.90
18	17	2	F6, F4	85.88	70.22	99.31	38.39
19	23	-	-	-	-	-	-
20	8	5	J20, T22, T11, P13, YC	80.73	64.01	95.08	47.10
21	20	5	P7, F19, XB, YD, T11	71.03	44.26	93.99	137.19

Table VII: Selected multi-class implementations.

Model	h	Sensors	Sensors ID	Gmean (%)	Svt (%)	Spc (%)	Time (min)
KNN	8	6	F1, XF, F19, T18, ZF, T21	76.67	59.64	98.58	0.03
SVM	-	-	-	-	-	-	timeout

the detection quality scales according to the number and type of features (i.e. sensors) and the ML model selected. In this particular case, the maximum geometric mean and sensitivity values are used as the conditions to select the approaches. However, different conditions, such as the specificity threshold, can be applied to this framework to meet the needs of other application. Therefore, this scalable detection quality framework might be applied to different domains.

C. Multi-class HFW in TEP

This subsection describes the exploration performed using a multi-class HFW approach, instead of the single-class one. This multi-class approach means that a single model will identify all the faults in TEP, instead of creating an individual model for each potential fault. In this experiment, the p parameter was also set to 10, the ML models were KNN and LSVM, and the h parameter is defined as proposed in Section IV-C. Given the complexity of the global search, a timeout of 2 days (2880 minutes) was set during the procedure. As a result, the process could only finish for the KNN model. Note that this timeout was not required in the single-class HFW, due to its lower computational complexity.

Fig. 4 shows the analysis performed for selecting the h parameter in the multi-class KNN model. In this figure, the first local maximum appears with eight sensors, providing a detection quality of 74.31%. The global peak corresponds to using 38 sensors, for which a detection quality of 76.62% is achieved. Therefore, adding 30 extra sensors (from 8 to 38) would only increase the detection quality a 2.3%, which is below the threshold fixed in a 5%. For this reason, the h parameter has been defined as 8.

After setting all the parameters, the next step is to execute the multi-class HFW, in this case, only using the KNN ML model because of the timeout issue described before. The results obtained can be observed in Fig. 5, where the geometric mean ranges from 46.63% to 81.01% when using from 1 to 10 sensors, respectively. The maximum detection quality is the 81.01% reached by the 10-sensor solution. To reduce the network complexity, it is chosen the minimum number of sensors that differ less than 5% from the maximum detection quality value. Thus, the optimal amount of sensors to be used during inference is therefore set to 6. In this regard, Table VII summarises the solution selected according to this multi-class HFW. It should be noticed that this solution has a substantial impact on the false-negative rating, implying an important limitation for real-world process monitoring applications. Note that although the solution with the highest sensitivity was selected in Fig. 5, this metric is below the 66.39%.

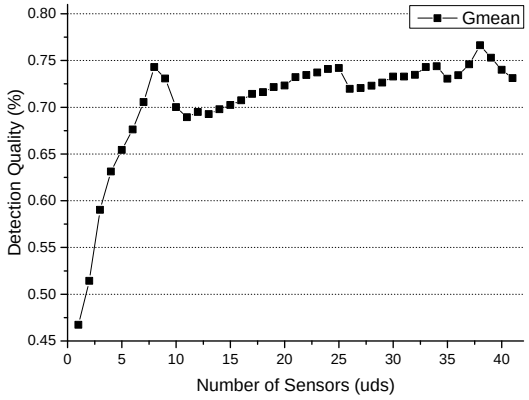

Figure 4: h parameter exploration for the multi-class KNN.

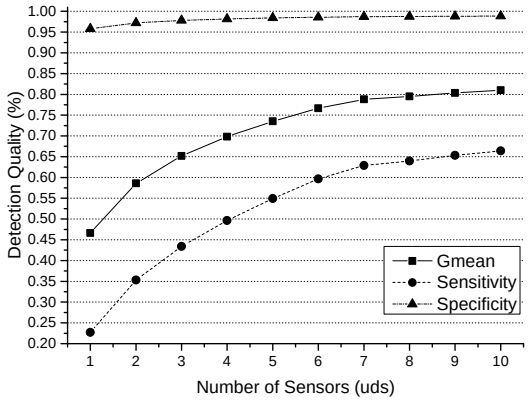

Figure 5: Feature space exploration for the multi-class approach with $model = KNN$, $p = 10$, and $h = 8$.

D. Comparing Single-class to Multi-class HFW in TEP

As stated before, the multi-class LSVM classifier did not converge to a solution. In turn, the 6-sensor multi-class KNN solution obtained has a lower detection quality than the single-class classifiers (see Tables V, VI, and VII), achieving an abso-

Table VIII: Comparison of TEP fault detection approaches within the state-of-the-art.

	[31]			[67]			[34]			[41]			[39]			[38]			Single-class HFW		
Fault	**Features**	**Gmean**	**Latency**	**Features**	**Gmean**	**Latency**	**Features**	**Gmean**	**Latency**	**Features**	**Gmean**	**Latency**	**Features**	**Gmean**	**Latency**	**Features**	**Gmean**	**Latency**	**Features**	**Gmean**	**Latency**
1	52(12)	91.87*	-	52(12)	98.77	24	33(12)	99.75	500	30	99.86	-	2(1)	99.90	37.5*	3(1)	99.44	-	4	99.65	1
2	52(12)	91.87*	-	52(12)	99.01	24	33(12)	99.56	500	21	98.75	-	5(1)	99.10	37.5*	3	97.81	-	4	99.35	1
3	52(12)	91.87*	-	52(12)	69.83	24	33(12)	99.27	500	12	60.23	-	-	-	-	26(1)	49.01	-	3	99.80	1
4	52(12)	91.87*	-	52(12)	69.02	24	33(12)	100.00	500	1(1)	99.97	-	1(1)	100.00	37.5*	1(1)	100.00	-	2	99.80	1
5	-	-	-	52(12)	100.00	24	33(12)	99.87	500	6(1)	99.89	-	3(1)	100.00	37.5*	25(6)	96.56	-	2	99.96	1
6	-	-	-	52(12)	98.73	24	33(12)	99.56	500	1(1)	99.87	-	2(1)	100.00	37.5*	1(1)	99.63	-	1	99.92	1
7	-	-	-	52(12)	98.22	24	33(12)	99.76	500	3(1)	100.00	-	3(1)	100.00	37.5*	2(1)	100.00	-	2	99.82	1
8	-	-	-	52(12)	95.94	24	33(12)	99.88	500	12(2)	82.18	-	4(1)	100.00	37.5*	5	97.46	-	3	99.82	1
9	-	-	-	52(12)	45.48	24	33(12)	99.53	500	6(1)	71.81	-	-	-	-	25(2)	47.45	-	5	93.35	1
10	-	-	-	52(12)	74.86	24	33(12)	99.69	500	7(1)	79.10	-	14(1)	99.40	37.5*	13(2)	71.64	-	1	98.03	1
11	-	-	-	52(12)	90.72	24	33(12)	99.88	500	2(1)	88.84	-	2(1)	100.00	37.5*	9(4)	84.80	-	2	99.60	1
12	-	-	-	52(12)	99.75	24	33(12)	99.48	500	10(1)	92.70	-	5	100.00	37.5*	10(2)	98.89	-	4	98.81	1
13	-	-	-	52(12)	99.75	24	33(12)	99.50	500	11(1)	83.66	-	7	100.00	37.5*	4(1)	90.16	-	2	99.96	1
14	-	-	-	52(12)	56.64	24	33(12)	98.59	500	3	99.94	-	2(1)	100.00	37.5*	2	100.00	-	2	99.96	1
15	-	-	-	52(12)	98.26	24	33(12)	99.20	500	6(1)	67.53	-	-	-	-	23(2)	55.94	-	2	99.66	1
16	-	-	-	52(12)	95.84	24	33(12)	99.76	500	6(1)	82.98	-	2(1)	100.00	37.5*	14(3)	73.28	-	3	99.55	1
17	-	-	-	52(12)	97.71	24	33(12)	91.59	500	2	95.57	-	27(1)	98.20	37.5*	8(1)	95.09	-	2	99.82	1
18	-	-	-	52(12)	93.91	24	33(12)	99.42	500	3	93.59	-	2	95.30	37.5*	3	85.66	-	3	96.97	1
19	-	-	-	52(12)	88.99	24	33(12)	91.43	500	5(1)	91.95	-	3(1)	100.00	37.5*	12(3)	85.68	-	2	99.82	1
20	-	-	-	52(12)	84.93	24	33(12)	99.37	500	4(1)	88.75	-	13	100.00	37.5*	20(4)	76.29	-	4	97.71	1
21	-	-	-	52(12)	87.74	24	-	-	500	2(1)	91.33	-	1(1)	100.00	37.5*	1(1)	99.24	-	3	92.17	1

lute difference higher than 15%. The multi-class solution also has a low sensitivity, much less than 90%, provoking a high rate of non-identified faults. Even with the best combination reached in the multi-class KNN exploration (i.e., using ten sensors instead of six), the detection quality and the sensitivity are still much lower than the single-class classifiers.

Moreover, the solutions provided by the single-class classifiers require a low number of sensors, being less than four sensors in most of the cases. This characteristic allows a distributed deployment of the system throughout the factory, using a processing platform near to the set of sensors required to detect each fault.

Apart from the detection quality, the single-class provides more robustness than the multi-class solution. Multi-class uses only one classifier to identify the whole set of faults. Therefore, when a single sensor stops producing data, the diagnosis of all the faults is interrupted. Differently, in the distributed single-class scheme, only the classifiers associated with the damaged sensor will be affected. Thus, the authors state that the single-class HFW approach proposed here clearly outperforms the multi-class version.

E. Evaluating Single-class HFW solutions at the Edge

Once shown that the single-class HFW outperforms the multi-class one, it should be checked that the solutions proposed for each fault by the single-class HFW are implementable at the edge. To this end, the optimal KNN and LSVM solutions shown in Tables V and VI have been implemented in a resource-constrained IoT edge node. In particular, the *cookie* platform has been selected for this evaluation. This is a modular platform developed at the Universidad Politécnica de Madrid [68]. It is comprised of four hardware layers: power supply, sensing/actuation, processing, and communications. The four layers are connected by a standard vertical connector, which creates a bridge for all the layers. The processing layer consists of a SAMA5D3 processor featured with 256 MB external RAM. This processor is an ultra-low-power, 32-bit medium performance ARM hard-float Cortex-A5, which consumes less than 150 mW in active mode.

Table IX shows the inference execution time in the *cookie* platform for each single-class HFW solution, trained ML model for a particular number of sensors. Focusing on KNN, the execution time ranges from 0.91 ms to 2.88 ms. Focusing on LSVM, the inference times range from 0.01 ms to 1.18 ms. Despite that, the KNN inference time might have a value up to 90 times higher than LSVM, either type of ML solutions can be implemented in the resource-aware IoT edge node. Note that in the TEP use case, the measured inference times are lower than the data acquisition time, established in three minutes.

Table IX: The inference execution time of the single-class HFW solutions in the embedded IoT cookie platform.

	KNN		LSVM	
Fault	**Sensors(uds)**	**Time(ms)**	**Sensors(uds)**	**Time(ms)**
1	4	1.86	3	0.01
2	4	1.86	3	0.02
3	3	1.50	3	0.70
4	2	1.25	-	-
5	2	1.25	5	1.19
6	1	0.91	1	0.01
7	2	1.25	3	0.71
8	3	1.50	3	0.69
9	5	2.00	-	-
10	1	0.91	3	0.01
11	2	1.25	-	-
12	4	1.86	2	0.48
13	2	1.25	6	0.97
14	2	1.25	-	-
15	2	1.25	-	-
16	3	1.50	-	-
17	2	1.25	5	0.18
18	4	1.86	2	0.01
19	3	1.50	-	-
20	5	2.00	5	0.69
21	8	2.88	5	0.01

From this analysis, it can be said that the best solutions found for the TEP use case are implementable at the edge. 21 different ML models will be distributed along the factory, running two types of ML techniques (KNN and LSVM), getting up to 2.88 ms for inference, and with an average quality detection of 98.74%.

F. Comparing Single-class HFW to the State-of-the-art

Table VIII compares the single-class HFW methodology compared in this paper to the most significant works in the literature. This table shows detection quality, latency and num-

ber of features (i.e., sensors) used. The number in parenthesis in the features column indicates the actuators required if any.

In the implementations in [31], [67], all the sensors and actuators were considered. This fact prevents the distribution of the fault-detection system. Hence, it can be only implemented in a centralised system. In [31], PCA was used to extract compressed features from which the most relevant ones are next selected. Therefore, they did not reduce the number of required sensors and actuators. Moreover, in [31], the authors studied only four faults, which limits the comparative analysis, also they applied a multi-class strategy. Besides, the detection quality reported for these four faults (91.87%) were surpassed by the methodology proposed in this paper. The implementation in [67] also provides average performance metrics below the results achieved with the proposed method. Besides, it requires a latency of 24 samples to produce the alarm, which means 72 minutes for the TEP use case. This is not adequate for a real-time fault detection system.

The method presented in [34] provided higher performance when compared to [31], [67], but also requiring 33 features and a latency of 500 samples, thus, 1500 minutes in the TEP model. The approach in [41] considered a feature selection method in conjunction with a classifier based on neural networks. When compared to the method proposed in this paper, it provides still a lower quality of detection, requiring more features. In [38], the authors offered multiple implementations whose performance values are near to [41], but using more data from the actuators and also more features per fault.

Only a few implementations achieve the same quality metrics as our proposed method. This is the case of non-linear SVM implementations reported in [38], [39]. Particularly, authors in [39] achieved the highest performance metric values, but it only increases an average of 0.81% the geometric mean compared to our solutions. Furthermore, it also requires 37.5 times more samples on average for each fault when compared to our implementation. Moreover, this method did not provide solutions for faults 3, 9, and 15, reported in the literature as the most difficult to detect. These implementations included a feature selection algorithm, but the method proposed in this paper still requires 1.99 times fewer features. Moreover, all the solutions provided in [39] required the data from the actuators.

The single-class HFW proposed method is the only that considers data from the sensors, discarding the actuators. The number of features is limited to a maximum of ten, providing still a high accuracy, that only differs on a 0.81% compared to the best results in the literature. Regarding latency, the solution proposed in this paper is the only that guarantees fault detection with a single sample. This results from the fact that the system does not require time-dependent features, only the sensor values at the given time instant. From a qualitative perspective, it must be noticed that the methodology proposed in this paper is fully scalable. That means that the quality of detection can be traded-off with the number of features finally used to detect each fault.

VII. Conclusion

In this paper, a novel scalable and distributed methodology to diagnose faults in the Industry 4.0 environment is proposed. It is based on the use of a hybrid feature selection procedure and lightweight binary classifiers. The HFW feature selection allows determining which sensors are more significant to detect each type of fault. By changing a parameter in the algorithm, the maximum number of sensors used in the exploration is changed, trading-off accuracy with computing complexity. In turn, the use of lightweight classifiers allows distributing computation near to the sensors throughout the plant. TEP is used as the industrial use case to validate the proposed algorithms. Obtained results are comparable with the best references in the state-of-the-art, which are neither distributed nor scalable while using data only coming from sensors. Selected solutions were implemented in an edge platform, verifying that the algorithm can be implemented and executed inside a resource-constraint edge embedded system, but also that it achieves the time execution required for the TEP use case. In the future, it is planned to implement the proposed algorithm in different computing platforms, ranging from specific AI devices to low-cost microprocessors and flash-based FPGAs, in order to extend the design space exploration from algorithms to the platform. Moreover, other use cases, such as smart grids, will be used to evaluate the effectiveness of the proposal.

Acknowledgment

This work has been partially funded by Spanish Ministry (Ministerio de Economía y Competitividad) under project PLATINO (Ref. TEC2017-86722-C4-2-R).

References

[1] L. D. Xu, E. L. Xu, and L. Li, "Industry 4.0: state of the art and future trends," *International Journal of Production Research*, vol. 56, no. 8, pp. 2941–2962, 2018.

[2] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE industrial electronics magazine*, vol. 11, no. 1, pp. 17–27, 2017.

[3] S. Ahmad, A. Badwelan, A. M. Ghaleb, A. Qamhan, M. Sharaf, M. Alatefi, and A. Moohialdin, "Analyzing critical failures in a production process: Is industrial iot the solution?" *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[4] E. Oztemel and S. Gursev, "Literature review of industry 4.0 and related technologies," *Journal of Intelligent Manufacturing*, pp. 1–56, 2018.

[5] L. Wang, C. Yang, Y. S. andHaifeng Zhang, and M. Li, "Effective variable selection and moving window hmm-based approach for ironmaking process monitoring," *Journal of Process Control*, vol. 68, pp. 86–95, 2018.

[6] M. Reis and G. Gins, "Industrial process monitoring in the big data/industry 4.0 era: From detection, to diagnosis, to prognosis," *Processes*, vol. 5, no. 3, p. 35, 2017.

[7] K. Severson, P. Chaiwatanodom, and R. D. Braatz, "Perspectives on process monitoring of industrial systems," *Annual Reviews in Control*, vol. 42, pp. 190–200, 2016.

[8] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1996–2018, 2014.

[9] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[10] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[11] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3039–3071, 2019.

[12] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *2016 49th Hawaii international conference on system sciences (HICSS)*. IEEE, 2016, pp. 3928–3937.

[13] D. Wu, S. Liu, L. Zhang, J. Terpenny, R. X. Gao, T. Kurfess, and J. A. Guzzo, "A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing," *Journal of Manufacturing Systems*, vol. 43, pp. 25–34, 2017.

[14] M. Nazemi, A. E. Eshratifar, and M. Pedram, "A hardware-friendly algorithm for scalable training and deployment of dimensionality reduction models on fpga," *arXiv preprint arXiv:1801.04014*, 2018.

[15] P. G. Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. I. Iamnitchi, M. Barcellos, P. Felber, and E. Rivière, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015. [Online]. Available: http://doi.acm.org/10.1145/2831347.2831354

[16] J. Portilla, G. Mujica, J.-S. Lee, and T. Riesgo, "The extreme edge at the bottom of the internet of things: A review," *IEEE Sensors Journal*, vol. 19, no. 9, pp. 3179–3190, 2019.

[17] S. Cass, "Taking ai to the edge: Google's tpu now comes in a maker-friendly package," *IEEE Spectrum*, vol. 56, no. 5, pp. 16–17, May 2019.

[18] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on vision processing unit for mobile applications," *IEEE Micro*, vol. 35, no. 2, pp. 56–66, Mar 2015.

[19] W. Saad, M. Bennis, and M. Chen, "A vision of 6g wireless systems: Applications, trends, technologies, and open research problems," *IEEE network*, vol. 34, no. 3, pp. 134–142, 2019.

[20] J. Miao and L. Niu, "A survey on feature selection," *Procedia Computer Science*, vol. 91, pp. 919–926, 2016.

[21] B. Zhou, A. Yu, J. Menke, and A. Yang, "Real-time hand model estimation from depth images for wearable augmented reality glasses," in *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE, 2019, pp. 269–273.

[22] R. F. Bikmukhamedov and A. F. Nadeev, "Lightweight machine learning classifiers of iot traffic flows," in *2019 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*. IEEE, 2019, pp. 1–5.

[23] A. N. Sokolov, I. A. Pyatnitsky, and S. K. Alabugin, "Research of classical machine learning methods and deep learning models effectiveness in detecting anomalies of industrial control system," in *2018 Global Smart Industry Conference (GloSIC)*. IEEE, 2018, pp. 1–6.

[24] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Computers & Chemical Engineering*, vol. 17, no. 3, pp. 245 – 255, 1993, industrial challenge problems in process control. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0098135493800181

[25] S. Yin, S. X. Ding, A. Haghani, H. Hao, and P. Zhang, "A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process," *Journal of Process Control*, vol. 22, no. 9, pp. 1567 – 1581, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0959152412001503

[26] Z. Chen, "Data-driven fault detection for industrial processes," *Journal of Process Control*, 2017.

[27] G. Luo, J. E. D. Hurwitz, and T. G. Habetler, "A survey of multi-sensor systems for online fault detection of electric machines," in *2019 IEEE 12th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED)*. IEEE, 2019, pp. 338–343.

[28] S. J. Qin, "Survey on data-driven industrial process monitoring and diagnosis," *Annual reviews in control*, vol. 36, no. 2, pp. 220–234, 2012.

[29] S. Zidi, T. Moulahi, and B. Alaya, "Fault detection in wireless sensor networks through svm classifier," *IEEE Sensors Journal*, vol. 18, no. 1, pp. 340–347, 2017.

[30] B. Chebel-Morello, S. Malinowski, and H. Senoussi, "Feature selection for fault detection systems: application to the tennessee eastman process," *Applied Intelligence*, vol. 44, no. 1, pp. 111–122, 2016.

[31] X. Gao and J. Hou, "An improved svm integrated gs-pca fault diagnosis approach of tennessee eastman process," *Neurocomputing*, vol. 174, pp. 906–911, 2016.

[32] M. Adeli and A. H. Mazinan, "High efficiency fault-detection and fault-tolerant control approach in tennessee eastman process via fuzzy-based neural network representation," *Complex & Intelligent Systems*, pp. 1–14, 2019.

[33] J. Sengupta, S. Ruj, and S. D. Bit, "A comprehensive survey on attacks, security issues and blockchain solutions for iot and iiot," *Journal of Network and Computer Applications*, vol. 149, p. 102481, 2020.

[34] P. Hajihosseini, M. M. Anzehaee, and B. Behnam, "Fault detection and isolation in the challenging tennessee eastman process by using image processing techniques," *ISA transactions*, vol. 79, pp. 137–146, 2018.

[35] W. Zou, Y. Xia, and H. Li, "Fault diagnosis of tennessee-eastman process using orthogonal incremental extreme learning machine based on driving amount," *IEEE transactions on cybernetics*, vol. 48, no. 12, pp. 3403–3410, 2018.

[36] M. Ammiche, A. Kouadri, and A. Bakdi, "A combined monitoring scheme with fuzzy logic filter for plant-wide tennessee eastman process fault detection," *Chemical Engineering Science*, vol. 187, pp. 269–279, 2018.

[37] J. Yuan and Y. Tian, "An intelligent fault diagnosis method using gru neural network towards sequential data in dynamic processes," *Processes*, vol. 7, no. 3, 2019. [Online]. Available: https://www.mdpi.com/2227-9717/7/3/152

[38] Y. Xue, L. Zhang, B. Wang, Z. Zhang, and F. Li, "Nonlinear feature selection using gaussian kernel svm-rfe for fault diagnosis," *Applied Intelligence*, vol. 48, no. 10, pp. 3306–3331, 2018.

[39] M. Onel, C. A. Kieslich, and E. N. Pistikopoulos, "A nonlinear support vector machine-based feature selection approach for fault detection and diagnosis: Application to the tennessee eastman process," *AIChE Journal*, vol. 65, no. 3, pp. 992–1005, 2019.

[40] M. Onel, C. A. Kieslich, Y. A. Guzman, and E. N. Pistikopoulos, "Simultaneous fault detection and identification in continuous processes via nonlinear support vector machine based feature selection," in *Computer Aided Chemical Engineering*. Elsevier, 2018, vol. 44, pp. 2077–2082.

[41] F. de Assis Boldt, T. W. Rauber, and F. M. Varejao, "Cascade feature selection and elm for automatic fault diagnosis of the tennessee eastman process," *Neurocomputing*, vol. 239, pp. 238–248, 2017.

[42] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[43] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, p. 67, 2016.

[44] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 94, 2018.

[45] J. C. Ang, A. Mirzal, H. Haron, and H. N. A. Hamed, "Supervised, unsupervised, and semi-supervised feature selection: a review on gene selection," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 13, no. 5, pp. 971–989, 2015.

[46] X. Zeng, S.-B. Yin, Y. Guo, J.-R. Lin, and J.-G. Zhu, "A novel semi-supervised feature extraction method and its application in automotive assembly fault diagnosis based on vision sensor data," *Sensors*, vol. 18, no. 8, p. 2545, 2018.

[47] R. Zhang, F. Nie, X. Li, and X. Wei, "Feature selection with multi-view data: A survey," *Information Fusion*, vol. 50, pp. 158–167, 2019.

[48] P. Peets, I. Leito, J. Pelt, and S. Vahur, "Identification and classification of textile fibres using atr-ft-ir spectroscopy with chemometric methods," *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy*, vol. 173, pp. 175–181, 2017.

[49] H. Zhan, S. Wu, R. Bao, L. Ge, and K. Zhao, "Qualitative identification of crude oils from different oil fields using terahertz time-domain spectroscopy," *Fuel*, vol. 143, pp. 189–193, 2015.

[50] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[51] R. Sheikhpour, M. A. Sarram, S. Gharaghani, and M. A. Z. Chahooki, "A survey on semi-supervised feature selection methods," *Pattern Recognition*, vol. 64, pp. 141–158, 2017.

[52] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[53] A. Jović, K. Brkić, and N. Bogunović, "A review of feature selection methods with applications," in *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*. Ieee, 2015, pp. 1200–1205.

[54] F. Alam, R. Mehmood, I. Katib, and A. Albeshri, "Analysis of eight data mining algorithms for smarter internet of things (iot)," *Procedia Computer Science*, vol. 98, pp. 437–442, 2016.

[55] G. Surrel, A. Aminifar, F. Rincón, S. Murali, and D. Atienza, "Online obstructive sleep apnea detection on medical wearable sensors," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 4, pp. 762–773, 2018.

[56] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep 1995. [Online]. Available: https://doi.org/10.1023/A:1022627411411

[57] A. Palaniappan, R. Bhargavi, and V. Vaidehi, "Abnormal human activity recognition using svm based approach," in *2012 International Conference on Recent Trends in Information Technology*. IEEE, 2012, pp. 97–102.
[58] Q. Kuang and L. Zhao, "A practical gpu based knn algorithm," in *Proceedings. The 2009 International Symposium on Computer Science and Computational Technology (ISCSCI 2009)*. Citeseer, 2009, p. 151.
[59] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Computers & Chemical Engineering*, vol. 17, no. 3, pp. 245 – 255, 1993, industrial challenge problems in process control. [Online]. Available: http://www.sciencedirect.com/science/article/pii/009813549380018I
[60] E. L. Russell, L. H. Chiang, and R. D. Braatz, *Data-Driven Methods for Fault Detection and Diagnosis in Chemical Processes*, 01 2000.
[61] K. Tidriri, N. Chatti, S. Verron, and T. Tiplica, "Model-based fault detection and diagnosis of complex chemical processes: A case study of the tennessee eastman process," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 232, no. 6, pp. 742–760, 2018.
[62] A. Kroll and H. Schulte, "Benchmark problems for nonlinear system identification and control using soft computing methods: Need and overview," *Applied Soft Computing*, vol. 25, p. 496–513, 12 2014.
[63] A. Ragab, M. El-koujok, M. Amazouz, and S. Yacout, "Fault detection and diagnosis in the tennessee eastman process using interpretable knowledge discovery," in *2017 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE, 2017, pp. 1–7.
[64] F. Forooghifar, A. Aminifar, and D. Atienza, "Resource-aware distributed epilepsy monitoring using self-awareness from edge to cloud," *IEEE transactions on biomedical circuits and systems*, 2019.
[65] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.
[66] T. Wong, "Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation," *Pattern Recognition*, vol. 48, no. 9, pp. 2839–2846, 2015.
[67] M. F. S. V. D'Angelo, R. M. Palhares, M. C. O. C. Filho, R. D. Maia, J. B. Mendes, and P. Y. Ekel, "A new fault classification approach applied to tennessee eastman benchmark process," *Applied Soft Computing*, vol. 49, pp. 676–686, 2016.
[68] P. Merino, G. Mujica, J. Señor, and J. Portilla, "A modular iot hardware platform for distributed and secured extreme edge computing," *Electronics*, vol. 9, no. 3, p. 538, 2020.

Rodrigo Marino received the BSc degree in Industrial Electronics Engineering and Automation from Universidade de Vigo, Spain, in 2015, and the MSc in Industrial Engineering in 2017 from the same University. He is currently working toward the Ph.D. degree in Industrial Electronics at Universidad Politécnica de Madrid. His current research area is in the field of machine learning applied to embedded systems, also known as expert embedded computing.

He is participating in a national research project, PLATINO, related to the enhace the acquisition systems, combining machine learning techniques in embedded systems, in order to develop expert sensors for the agro-food industry. He has also participated in an industrial project, REMO, with Indra and Repsol companies, so as to create a framework for chemical detection.

Cristian Wisultschew received the M.Sc. degree in Industrial electronics from the Universidad Politécnica de Madrid, Madrid, Spain, on 2018. He is currently a Ph.D. student at Universidad Politécnica de Madrid. He carries out his research activity within the Centro de Electrónica Industrial, belonging to the UPM. His research interests are focused on Embedded Machine Learning, Deep Learning HW accelerators, Internet of Things and Digital Embedded Systems.

He is participating in a H2020 project, SCOTT, related with the object detection and tracking system used in railway level crossing surveillance systems. He is also participating in a national research project, PLATINO, related to machine learning techniques in embedded systems using specific DL HW accelerators applied to the agro-food industry.

Andrés Otero received his M.Sc. degree in Telecommunication Engineering from the University of Vigo, where he graduated with honors in 2007. He received his Master of Research and Ph.D. degrees in Industrial Electronics from Universidad Politécnica de Madrid (UPM), in 2009 and 2014, respectively. He is currently an Assistant Professor of electronics with the UPM, as well as a researcher in the Centro de Electrónica Industrial (CEI). His current research interests are focused on Embedded System Design, Reconfigurable Systems on FPGAs, Evolvable Hardware and Embedded Machine Learning . During the last years, he has been involved in different research projects in these areas, and he is the author of more than 30 papers published in international conferences and journals. He has served as the Program Committee member of different international conferences in the field of reconfigurable systems, such as SPL, ERSA, ReConFig, DASIP and ReCoSoC.

Jose M. Lanza-Gutierrez received the B.S and M.S degrees in Computer Science from the University of Extremadura, Caceres, Spain, in 2008 and 2009 respectively. In 2010, he obtained the master's degree in grid computing and parallelism at the same University. In 2015, he received the Ph.D. degree in Computer Science from the University of Extremadura under the guidance of Prof. Dr. Juan A. Gomez-Pulido. Currently, he is an assistant professor at University of Alcalá, Spain. He has authored or co-authored more than 40 publications, including Journal Citation Report (JCR) papers in journals, such as Applied Soft Computing, Expert Systems with Application, BMC Bioinformatics, Soft Computing, Reliability Engineering and System Safety, and Journal of Heuristics. His main research interests include metaheuristics, digital embedded systems, cognitive computing, and machine learning.

Jorge Portilla (M'09-SM'18) received the M.Sc. degree in Physics from the Universidad Complutense de Madrid, Madrid, Spain, on 2003, and the Ph.D. degree in Electronic Engineering from Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2010.

He is currently an Assistant Professor Tenured at Universidad Politécnica de Madrid. He carries out his research activity within the Centro de Electrónica Industrial, belonging to the UPM. His research interests are focused on Wireless Sensor Networks, Internet of Things, Digital Embedded Systems and Reconfigurable FPGA-based embedded systems. He has participated in more than 30 funded research projects, including European Union FP7 and H2020 projects and Spain Government funded projects, as well as private industry funded projects, mainly related to Wireless Sensor Networks and Internet of Things. He was a visiting researcher with the Industrial Technology Research Institute (ITRI), Hsinchu Taiwan, in 2008 and with the National Taipei University of Technology (Taipei Tech), Taipei, Taiwan, in 2018, working on Wireless Sensor Networks hardware platforms and network clustering techniques. He has numerous publications in prestigious international conferences as well as in journals with impact factor.

Eduardo de la Torre is Associate Professor in Electronics at Universidad Politécnica de Madrid (UPM), Spain, doing his research at the Centre of Industrial Electronics. He obtained his PhD in Electrical Engineering from UPM in 2000. His main expertise is in FPGA design, embedded systems design, HW acceleration, signal processing and partial and dynamic reconfiguration of digital systems. He has participated in more than 40 projects, eleven of them being EU funded projects and, overall, in nine funded projects related with reconfigurable systems. He has been program chair of ReConFig and general chair of ReCoSoC, two conferences with strong interest in hardware reconfiguration.