Identification of sensors in smart manufacturing via mutually exclusive multiple time series classification

Alberto Ceselli[1], Giuseppe De Martino[1,2], Marco Premoli[1*]

[1]Department of Computer Science, Università Degli Studi di Milano, 18, via Celoria, Milano, 20133, Italy.
[2]Par-Tec SpA, 11, via Panfilo Castaldi, Milano, 20124, Italy.

*Corresponding author(s). E-mail(s): marco.premoli@unimi.it;
Contributing authors: alberto.ceselli@unimi.it;
giuseppe.demartino@unimi.it;

Abstract

Industry 4.0 envisions full integration of manufacturing machines with decision support tools and ultimately with business processes. Our research focuses on methodologies to achieve such integration: production devices are equipped with sensors that provide streams of data; decision support systems collect and manipulate these data. Seamless integration requires digital interfaces in between them, mapping each stream a correct semantics. This may be untrivial in scenarios like retrofitting. Currently, the lack of consensus on standardization requires that such a mapping is performed manually. We propose a methodology to automate it.
Such task requires the simultaneous classification of multiple time series, with the specific constraint that each of them must be assigned a different class. This latter feature makes existing methods unsuitable.
We propose a pipeline of models, including: ensemble methods for single time series classification (supervised learning), exact matching optimization (mathematical programming), and adaptive aggregation to improve accuracy over time.
We experiment on real data from the textile industry. Our pipeline shows to clearly outperform approaches from the literature. Our ensemble methods show to be competitive with benchmarks from the literature also on the single time-series classification task. The combined use of machine learning and mathematical programming allows to simultaneously exploit a data driven approach and obtain a-priori guarantees of compliance with the specific mutual exclusion constraint.

Furthermore, the computing load of the single components of our pipeline keeps very limited, enabling the deployment in computing environments with low resources.

Keywords: Industry 4.0 enablers, Retrofitting, Time Series Classification, Mathematical Programming, Sensors

1 Introduction

The industrial sector is facing an increasing push to fully move to Industry 4.0 (I4.0) standards (Ai, Wang, Han, & Ye, 2024; Olsen & Tomlin, 2020; Oztemel & Gursev, 2020; Xu, Xu, & Li, 2018). In manufacturing, such a transition involves not only monitoring and automation of machines, but also better integration with decision support tools and in general with business processes.

In this paper we focus on a challenge arising in production monitoring. We consider a set of devices, each equipped with a set of sensors. Each sensor produces a stream of data. The providers of external decision support tools need to retrieve these streams of data for analytics, and finally to plan actions on the corresponding devices. A key step in this retrieval operation is the mapping of the stream which is read from each sensor to the corresponding *semantics*; by semantics we mean the class of operations that the sensor is monitoring (*e.g.*, the temperature or pressure of a specific machine component, a start-stop trigger, a counter for a specific event). Despite efforts on standardization, in a wide range of applications such a mapping is far from obvious. For instance, the devices might be legacy machines, which are not yet compliant to I4.0 standards, and require therefore retrofitting (Alqoud, Schaefer, & Milisavljevic-Syed, 2022; Sanchez-Londono, Barbieri, & Fumagalli, 2023). They can be as well generic Industrial Internet of Things (IIoT) devices, which are installed in the production plant, and need to hot-plug into the company information system, but still require manual customization (Haskamp, Orth, Wermann, & Colombo, 2018; Nayernia, Bahemia, & Papagiannidis, 2022). That is, the heterogeneity of implementations and settings makes the automated identification of sensors an open challenge (Sun et al., 2023).

Our research study stems in fact from a real-world application in textile manufacturing. Similar machines from different providers, or even different versions of the same machine from the same provider, may expose a different registry interface for their sensors. The lack of standardization requires a tedious and potentially long process of manual configuration to identify the matching between each registry and the type of data which is read in it. This ultimately undermines the configuration flexibility of decision support tools, and consequently their adoption rate.

At the same time, fully automating this task with methods from the literature is not possible. In fact, while mapping a single stream to its semantics translates into a time series classification problem, a further feature complicates the task: each device produces at most *one* stream of data for each semantic. Mapping two streams to the

same class would produce a conflict in the connection to the Information System, and must therefore be avoided.

We propose a methodology to automatically obtain this mapping with *a-priori guarantee of mutual exclusion*. We design a pipeline of models, combining machine learning and mathematical programming techniques. Our methodology proves more effective than benchmarks from the literature and baseline approaches, and simple enough to foresee direct implementations in a real-world system. The manuscript is organized as follows. In Section 2, we detail our industrial context, and review background literature. In Section 3, we formalize our approach in terms of both architecture and mathematical models. In Section 4, we discuss the structure and manipulation of data sources. In Sections 5 and 6, we undertake an experimental campaign, benchmarking with existing methods from the literature. In Section 7, we collect some conclusions.

2 Industrial Context

2.1 Smart manufacturing architecture

As discussed in the Introduction, our models cover a range of applications in production monitoring for I4.0. For the ease of presentation, in this Section we illustrate the particular case of retrofitting from which our research interest arises.

In our scenario we consider a three-layer hierarchical smart manufacturing architecture, commonly considered in literature (Josbert, Wei, Ping, & Rafiq, 2024; Li et al., 2024; Nain, Pattanaik, & Sharma, 2022; Qi & Tao, 2019). We summarize such architecture in Fig. 1.

The first layer is represented by physical assets ('Physical Layer' box in Fig. 1), which in our case are manufacturing machines. These machines expose an interface for their sensors producing numerical data: in our case such interface is represented by a Programmable Logic Controllers (PLC) (Sehr et al., 2020). As a practical example, sensors may generate real-valued data for the machine's temperature, speed, engine revolutions, liquid levels and pressure, as well as binary data encoding specific states (such as active, inactive, or error conditions). In case of retrofitting, the computational and networking capabilities of the machine are extended by enhancement technologies, for example by attaching Single Board Computer (SBC) (Matt, Modrák, & Zsifkovits, 2020). The SBC retrieves sensor data written by PLCs, typically through serial communication protocol, and provides limited computational capabilities.

The second layer acts as a middleware (second box in Fig. 1), and might be represented by the edge/fog layer (Shinde, Hemanth, & Elhoseny, 2023). In our scenario, it is composed by multiple SBCs: they collect data provided by all machines in the physical layer and provide additional computational and storage capabilities, close to the manufacturing devices.

On top, we have an application layer (third box in Fig. 1), for example a cloud-computing layer (Khan, Ali, & Nazir, 2024), holding huge computation capabilities and storage space, with limitations in terms of communication latency. It is the destination for the data from all sensors in the physical layer, delivered by middleware. The

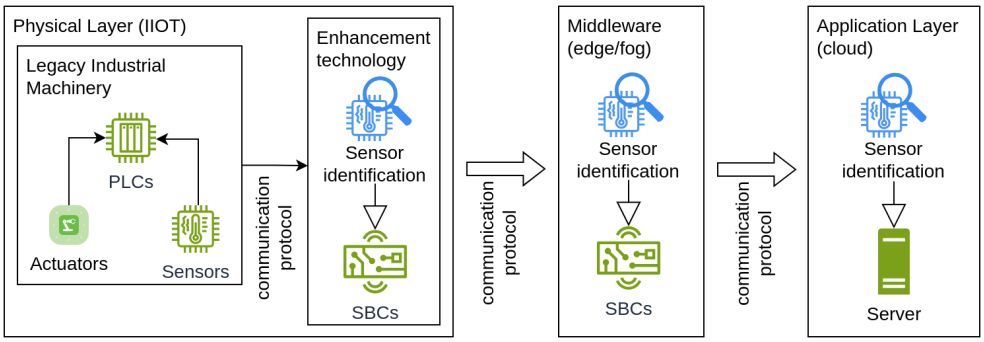


Fig. 1: High-level design of the three-layer architecture of our smart manufacturing scenario: sensor identification algorithm may run in any layer, with limitations on the available computational power.

collected data may undergo transformations through dedicated services to populate a data storage system. In this layer, off-the-shelf application-level softwares, such as decision support tools, are fed with such transformed data.

2.2 Mapping raw data from sensors to their semantics

The production of large quantities of raw data is a common setting in I4.0 and specifically in the manufacturing sector. The automatic management of these data is the basis of numerous applications, most of which use machine learning (ML) techniques, and in particular supervised learning (Rai, Tiwari, Ivanov, & Dolgui, 2021; Usuga Cadavid, Lamouri, Grabot, Pellerin, & Fortin, 2020). Different machines are involved in the same production process, each generating large volumes of heterogeneous, rapidly changing data. Handling them is a challenge on its own (Escobar, McGovern, & Morales-Menendez, 2021; Rosati et al., 2023)

While low-level data transmission between industrial machine and external devices is standardized, there is a lack of consensus on the standardization of the mapping between such data and their semantics. Different providers of the same machine may supply their own proprietary formats. A manufacturing company holding machines from multiple providers must integrate data coming from them to a common semantics to enable high-level application such as decision support tools. Current industrial practice is to resort to time consuming manual configuration. We refer to Sun et al. (2023) for a recent survey on current standardization effort for data access in IIoT.

Our proposal is to automate the mapping between raw data and their semantics by means of a software tool for the identification of the *topic* of the data stream produced by each sensor. We use the term *topic* to make it clear that the type of semantics we try to identify relates only to the data source (which keeps the same over time), and not to the specific activity on the machine (which could change over time). For brevity, we informally use the term *sensor* to indicate also the stream of data coming from the sensor. The assumptions we make concerning our mapping tool are the following.

- we assume the initial availability of a known mapping for a set of data streams, each correctly marked with a label indicating its topic. This initialization can benefit from semi-manual configuration; that is, the topics of some registries could actually be documented; for the remaining ones, a domain expert may initially perform manual configuration.
- Once an appropriate set of data is collected and labeled, it is stored in a central location, where a supervised training of models can take place. It can be, for instance, the server in Fig. 1.
- The models produced by training will have as input a proper manipulation of the streams coming from the registries, and as output the guess of the topic related to each stream, which is either a known topic or the special label *unknown topic* if the stream can not be classified.
- The model, which has been centrally trained, could then be distributed either on the intermediate layer's SBCs, on SBCs directly attached to the PLCs of the industrial machinery (that is, in the middleware or in the physical layer of the architecture of Fig. 1, respectively), or on new, previously unseen machines connecting to the system. Such deployment imposes limitations on the resources required to run the model.
- An additional manual configuration step is triggered *only* if some of the new streams is labeled as *unknown* by the models. After further manual checking (and possibly relabeling), also the new data becomes part of the training base, and a re-training operation might be triggered.

We remark that the setting would be the same if generic IIoT devices appear in place of the legacy machines, and different technological choices are made for network communications and information system implementation.

2.3 Background on time series classification

The main link enabling our proposal is methodological, and consists in the task of training models with predictive power on the semantics of sensor data streams produced by sensors over time. Such prediction must comply two additional constraints. First, the mapping of two data streams to the same class would produce a conflict in the connection to the Information System, and must therefore be avoided with a-priori guarantee of mutual exclusion of the classification. Second, as depicted in Fig. 1, while training could in principle be performed also in the cloud, often the classification model needs to be used directly in the physical layer, such as on an SBC (or any equivalent enhancement technology) attached directly to the manufacturing machine; this latter has strict limits on the amount of computational resources available, therefore restricting the scope of suited classification methodologies. For instance, models obtained by computational-demanding deep learning approaches (Fawaz, Forestier, Weber, Idoumghar, & Muller, 2019) are inapplicable in our case.

A natural approach for the classification of data stream is to exploit time series analysis (Fawaz et al., 2019; Zheng, Liu, Chen, Ge, & Zhao, 2014), and more specifically time series classification, which has been subject to deep investigations over the last

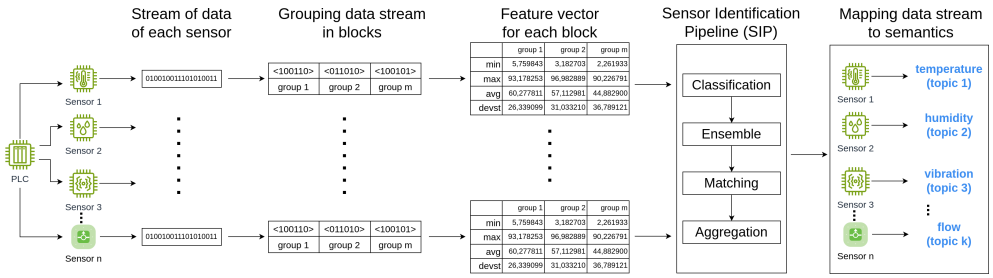


Fig. 2: Architecture of our method for the identification of sensors

decades. For a recent and wide survey on the subject we refer to Farahani et al. (2023), in which a taxonomy of algorithms and applications in manufacturing is also provided.

Several effective methodologies have been proposed, some of which led to software libraries which are amenable for industrial use. In terms of recent general-purpose tools, we mention `tslearn` (Tavenard et al., 2020), which provides a set of classification models, as well as preprocessing and feature extraction tools. More specific tools for classification include `pyts` (Faouzi, 2022), which collects effective implementations of several state-of-the-art methods like the Bag-of-SFA-Symbols transformer and the one-nearest neighbor classifier with special metrics (Schäfer, 2015), and dictionary-based methods like (Schäfer & Leser, 2023), as well as other techniques. We refer to Faouzi (2022) for an overall comparison between existing packages. These methods are not suited to directly tackle our problem, as they consider the classification of one univariate time series at a time, while our application requires the classification of several (univariate) time series at once with a-priori guarantee of mutual exclusion of classification.

The classification of several (univariate) time series at once clearly requires different approaches than the classification of a single multivariate time series (Hsu & Liu, 2021; Ruiz, Flynn, Large, Middlehurst, & Bagnall, 2021; Xi, Wang, Chen, & Wu, 2023), where multiple streams are classified as a single entity, or than a single multiclass classification for a univariate time series (Bradde, Fracastoro, & Calafiore, 2021), where a single stream is mapped to an entity chosen among a set of classes, hence not considering the mutual exclusion among streams.

3 Models

To face our mapping task, we design a pipeline of models. Its architecture is summarized in Fig. 2, while its pseudo-code is presented in Algorithm 1. A set I of sensors from a manufacturing machine is represented on the left: each is producing a stream of data. The pipeline output is represented on the right. We assume that a set of known topics K is given. The output is a label for each sensor, representing either a topic $k \in K$ or an additional label *unknown topic*. In the following, we refer to our algorithm as Sensor Identification Pipeline (SIP); its key components are detailed below.

3.1 Classification

3.1.1 Grouping

The first step in our pipeline is the creation of *blocks*. For each stream, a block consists in the sequence of values appearing in a certain time slot, whose length τ is fixed. In our application, not all sensors may output values at constant rates, and therefore, not all blocks may contain the same number of values. We denote by T the set of time slots in a certain running time horizon, and by V_{it} the set of values produced by sensor $i \in I$ in time slot $t \in T$.

3.1.2 Feature Extraction

In the second step we compute features of each block independently. The main idea is to represent the distribution of values in each block in a compact way. After preliminary experiment, we found that, in this step, recording the following five simple statistical features suffices (Nanopoulos, Alcock, & Manolopoulos, 2001): minimum, maximum, average and standard deviation and number of values in the block. In terms of notation:

$$\begin{aligned}
\phi_{it}^{M} &= \max(V_{it}) \\
\phi_{it}^{m} &= \min(V_{it}) \\
\phi_{it}^{a} &= \frac{\sum_{v \in V_{it}} v}{|V_{it}|} \\
\phi_{it}^{s} &= \sqrt{\frac{\sum_{v \in V_{it}} (v - \phi_{it}^{a})^2}{|V_{it}|}} \\
\phi_{it}^{c} &= |V_{it}|
\end{aligned}$$

We remark that the feature extraction is just a modular component of the proposed pipeline: its implementation can be customized and changed if necessary, without further changes to the other components of the pipeline.

At the same time, the specific option that we take into account to design and experiment the full framework, is peculiar, contributing to the good performance of the full SIP. In fact, splitting into blocks and then using simple statistical features allows to achieve good accuracy with a very limited computational cost, thereby making the deployment of SIP well suited also on a resource constrained SBC.

3.1.3 Ensemble of classification algorithms

The third step is a classification one. That is, we assume a set of functions $f_k(\phi^M, \phi^m, \phi^a, \phi^s, \phi^c)$ to exist for each topic $k \in K$. Each function maps the features of a into a boolean label: the output of the function $f_k()$ is *true* if the topic of the sensor producing the block is k, *false* otherwise.

These functions $f_k()$ are obtained with a data-driven approach: a set of blocks is collected in a training phase, coming from sensors whose topics are known, and their features are measured. As supervised learning we consider binary classification, where each $f_k()$ is trained independently.

In working conditions, therefore, the pipeline applies these pre-trained functions to each block. For each sensor $i \in I$ we define an *observation time* Q_i, which is a set of time slots in which the pipeline is allowed to measure the output of sensor i before providing its final output.

A *score* π_{ik} is computed by first considering

$$\rho_{ik} = \frac{|\{t \in Q_i : f_k(\phi_{it}^M, \phi_{it}^m, \phi_{it}^a, \phi_{it}^s) = \text{true}\}|}{|Q_i|}$$

that is the fraction of slots in Q_i in which blocks from $i \in I$ have been classified as topic $k \in K$, and then normalizing:

$$\pi_{ik} = \frac{\rho_{ik}}{\sum_{j \in K} \rho_{ij}}.$$

Intuitively, the π_{ik} values represent a guess on the probability that sensor $i \in I$ is producing data of topic $k \in K$.

We consider a generalization of this principle. Instead of choosing only one classification method, we train a *set* C of classifiers, each classifier $j \in C$ producing a model $f_k^j()$ for function $f_k()$. Each of them may yield a different score π_{ik}^j. The final aggregated score π_{ik} is computed as:

$$\pi_{ik} = g\left(\pi_{ik}^1 \dots \pi_{ik}^{|C|}\right) \tag{1}$$

for a suitable consensus function $g() : [0,1]^{|C|} \to [0,1]$. Such a practice share the rationale with stacking in ensemble methods, and therefore we refer to $g()$ as the *ensemble function*. An intuitive choice of $g()$ is the arithmetic average; specific choices of $g()$ for our application are instead presented in Section 6, and motivated by experimental evidence.

3.2 Matching

The fourth step exploits the prior knowledge that two sensors cannot produce data for the same topic. We search for a mapping $\mu() : I \to K$ of sensors to topics maximizing the likelihood function

$$\prod_{i \in I} \pi_{i\mu(i)}$$

We formalize the problem of finding such an optimal mapping in terms of mathematical programming. First, we express the likelihood in logarithmic terms:

$$\max \log \prod_{i \in I} \pi_{i\mu(i)} = \max \sum_{i \in I} \log \pi_{i\mu(i)}$$

Then, we introduce binary decision variables x_{ik} which take value 1 if sensor i is mapped to topic k, 0 otherwise. The maximum likelihood mapping problem can be

formulated as follows:

$$\max \sum_{i \in I, k \in K \cup \{u\}} \log(\pi_{ik})\, x_{ik} \tag{2}$$

$$\text{s.t.} \sum_{k \in K \cup \{u\}} x_{ik} = 1 \qquad \forall i \in I \tag{3}$$

$$\sum_{i \in I} x_{ik} \leq N_k \qquad \forall k \in K \cup \{u\} \tag{4}$$

$$x_{ik} \in \{0,1\}^{I \times K \cup \{u\}} \tag{5}$$

where u is an additional element encoding the *unknown* topic. Constraints (3) impose that exactly one topic $k \in K$, or the unknown topic, is assigned to each sensor i. Constraints (4) impose that at maximum N_k sensors are assigned to topic k. Under practical uses of the model, we expect $N_k = 1$ for the real topics, and $N_k = |I|$ for the *unknown* topic, but other values might be pertinent in specific instances. The objective (2) maximizes the total likelihood in logarithmic terms. After optimization we consider $\mu(i) = k$ if and only if $x_{ik} = 1$.

We recall that values π_{ik} are computed in the previous step of the pipeline, and are therefore data in the model (and so is the corresponding logarithm). The values π_{iu} corresponding to the unknown status are set in a parametric way: in Section 6 we detail and motivate our final implementation.

Model (2)–(5) is therefore a linear assignment problem, which possesses the integrality property. That is, integrality conditions (5) can be dropped and the problem can be solved as a Linear Program (Wolsey, 2020), obtaining an optimal solution which automatically satisfies conditions (5). It can be even be further reformulated as a minimum cost flow problem on a compact graph: in both cases it allows for fast polynomial time exact resolution algorithms (Ahuja, Magnanti, & Orlin, 1993).

3.3 Adaptive Aggregation

Summarizing: (i) we assume the block length τ to be defined by the application designer, (ii) the functions $f_k()$ to be found in a training phase ahead of deployment, (iii) the parameters N_k, the function $g()$ and the observation times Q_i to be chosen by the application user. During its usage, the pipeline can start giving an output as soon as Q_i time slots have been observed for each $i \in I$, and the first matching problem is solved. However, each sensor will likely go on producing data *after* this first prediction. Intuitively, the more the observation time, the better the expected accuracy of the mapping produced by the pipeline. It is therefore appealing to make such a prediction adaptive, possibly refining over time. A trivial option to try to adapt it when more time is available would be to enlarge the Q_i observation times. We indeed evaluated such an option, but found the following policy to be more effective.

Let us assume that the system has been measured for a certain time period $[0, t)$ (such that at least Q_i slots of measurements are available for each sensor $i \in I$), and we aim at updating our mapping at time t. We proceed as follows: for $p = 1 \ldots P$ rounds

Data: For each sensor in set I a stream of data of T time-slots of length τ ($|T| = M \cdot Q_i, M \geq 1$)
Result: mapping $\mu() : I \to K$
for $[t, t+1) \in T$ **do**
  compute and store $f^j_{k,i,[t,t+1)}(), \forall j \in C, i \in I, k \in K$, using values of all features $\phi_{i,[t,t+1)}$ `/* Classification */`
end
for $p \leftarrow 0$ **to** P **do**
  for all $i \in I$, draw an observation time Q^p_i (at random or with ad-hoc policy);
  compute $\pi^j_{i,k}, \forall j \in C, k \in K, i \in I$ using classifier outputs $f^j_{k,i,t}()$ for all time-slots t in Q^p_i;
  compute $\pi_{i,k} = g\left(\left\{j \in C \colon \pi^j_{i,k}\right\}\right), \forall i \in I, k \in K$ `/* Ensemble */`
  compute and store mapping $\mu^p()$ solving maximum likelihood matching model (2)–(5) `/* Matching */`
end
compute final mapping $\mu()$ from aggregation of all mappings in set $\{p \in P \colon \mu^p()\}$ `/* Aggregation */`

Algorithm 1: Pseudo-code of our algorithm for sensor identification (SIP)

(1) we draw an observation time Q^p_i for each $i \in I$
(2) we compute π^p_{ik} values, as described before, setting $Q_i = Q^p_i$ for each $i \in I$
(3) we compute a maximum likelihood mapping $\mu^p()$

the updated mapping $\mu()$ at time t is then produced by a proper aggregation of the mappings $\mu^p()$. The choice of Q^p_i in step (1) can either be random or application-driven; for instance, using a sliding window approach, they can be the last P disjoint consecutive slots of Q_i elements for each $i \in I$. The aggregation of mappings also allow some freedom. We considered a few options including majority voting (set $\mu(i) = k$ if k is the most frequent value for which $\mu^p(i) = k$) and its variants, maximum likelihood (set $\mu() = \mu^{\bar{p}}()$ if $\bar{p}$ is the round producing the highest overall likelihood value in the matching model), or last step (simply keep $\mu() = \mu^P()$). A computational comparison is carried out in Section 6.

4 Data Sources

We had access to approximately one year of production data from three manufacturing companies. Their names cannot be disclosed due to industrial agreements, but they all operate in the high-quality textile sector in Italy. Each company has a different set of machines, each providing data from a set of sensors. Each record in their databases contains: a timestamp, the unique identifier of the machine, the unique identifier of the sensor and the measured value.

We built a dataset for each company as follows: for each sensor, we merged in a single time series all data produced by each machine, sorted by timestamp. As a result, we obtained three datasets, one per company, named D1, D2 and D3 in the remainder,

Table 1: Number of occurrences (and number of total hours) of data per sensor and dataset

	Sensors						
Dataset	I1	I2	I3	I4	I5	I6	I7
D1	36674 (41)	33329 (42)	276541 (118)	273634 (119)	36494 (53)	32047 (53)	–
D2	–	–	–	–	–	–	148761 (215)
D3	10597 (53)	8264 (42)	1386461 (2056)	1281156 (2002)	277903 (1774)	261081 (1728)	–

and seven sensors, labeled from I1 to I7 in the remainder. The topics corresponding to sensors I1 to I7 on the original machines are known, and all distinct.

In Table 1, we show the number of samples and the length in hours of the resulting time series for each dataset and each sensor. We remark that, while D1 and D3 share the same set of sensors, D2 has data from a single sensor that is not present in the other datasets. We also remark that the stream of data of the same sensor from different companies may differ (even if the topic is the same) since they reflect a specific industrial process and working behavior of operators. In fact, they may show different patterns over time even within the same company.

We have considered four experimental scenarios:

- E1 – in a best-case scenario, both training and testing data come from the same company. We split dataset D1 in two sub-sets, so that each sub-set has same number of occurrences for each sensor. The first half of occurrences of each sensor is used for training, while the second half is used for testing. While training and testing sets contain different data, they should reflect the same working processes and working behavior of the operators of the machines that generate them;
- E2 – in a scenario when data from a new company are analyzed, the training is performed on dataset D1 while testing is performed on dataset D3, coming from a different company; that is, we test the potential of transferring an existing model to a new company. In this case, data of a new company may reflect a different working processes and behavior from that of data used to train the classification algorithms: we study the changes of performances in such case;
- E3 – extending experiments E2, training is performed on data of two companies, namely D1 and D2, while testing is performed on data of a third company, namely D3. D2 contains one additional sensor, not present neither in D1 nor in D3: we analyze the changes of performance of classifiers when new data are used to update their training;
- E4 – finally, we check the robustness of the classifiers when tested against data of unknown sensors. Training is performed on dataset D1 and testing on dataset D2, which is composed by a single sensor not present in D1.

5 Evaluating the performances of the classification step

All steps or our SIP pipeline were implemented in Python 3.12.3 and executed on a workstation equipped with a 1.80GHz Intel(R) Core(TM) i7-8550U CPU and 16GB of RAM. The tests were conducted on the operating system Windows 10 Pro 22H2.

On the same workstation, additional tests were conducted on a simulated Single Board Computer (sSBC). A virtual machine on Oracle VirtualBox 7 was configured as follows to mimic the actual SBC provided to the customers: 2 virtual CPU cores with 50% Execution Cap and 4GB of RAM, running the operating system Ubuntu LTS 24.04.

In the first phase of our approach we use supervised classification algorithms to assign a single topic to each sensor. In this section we compare the performance of several classification algorithms to identify which best suits our goal.

5.1 Features and classifiers

The set T of time slots is defined by partitioning the stream of each sensor in non-overlapping sub-sequences of ten-minutes each (*i.e.*, parameter $\tau = 10$ minutes). On each such a block we compute the features described in Section 3.1.2.

We compare the following classification algorithms: the C-Support Vector Classification (**SVM** in the remainder), the k-nearest neighbors (**KNN** in the remainder), the decision tree (**TREE** in the remainder) and the Gaussian naïve Bayes (**BAYES** in the remainder). We used the implementation of `scikit-learn` Python library (Pedregosa et al., 2011) of these classifiers. We present results of the use of these classifiers for binary classification. We have also experimented the same classifiers for multiclass classification: this variant resulted in worse performance than binary classification and hence we hereby omit their discussion, which can be found in Supplementary Material.

We include in the comparison also two ad-hoc classification algorithms for time series: k-nearest neighbors with dynamic time warping metric (Sakoe & Chiba, 1978) using the implementation of `pyts` library (Faouzi & Janati, 2020) (**DTW** in the remainder) and the support vector machine with the global alignment kernel (Cuturi, 2011) using the implementation of `tslearn` library (Tavenard et al., 2020) (**GAK** in the remainder). According to Bradde et al. (2021), **DTW** proved to be a suitable tradeoff between accuracy and computing effort; **GAK** instead was taken as representative of methods targeting higher accuracy at the price of higher computing efforts. They have the further advantage of being available in industry-ready implementations.

It is important to stress that, while **SVM**, **KNN**, **TREE** and **BAYES** work only on the five features of each single block, **DTW** and **GAK** work on the whole data of the block as a time series (and therefore have a chance to be more accurate, at the price of higher training and prediction complexity). The complete settings of hyperparameters for all classifiers are described in Appendix A (Table A.1).

We evaluate the performance of the classifiers with traditional key performance indicators (KPI): besides accuracy, precision, and F_1-score, also weighted versions of precision and recall are considered, in which metrics are computed for each label

separately and then average weighted by the support of the label (that is, the number of actual occurrences of the label in the dataset). A formal description of our KPIs is given in Appendix B. Complete results, in terms of each KPIs of all classifiers for each experiment are presented in Supplementary Materials, while in the following we discuss aggregated results.

5.2 Experiment E1: baseline scenario

Dataset D1 is used as base for both train and test sets. D1 is split into two equal parts so that each part has the same number of occurrences for each topic: the first half is used as training set, while the second half is used as test set.

Fig. 3a shows spider-charts of KPI values of each classifier, averaged over all topics. **TREE** has best performance among all classifiers (see Fig. 3a), with an accuracy of 0.94 and a F_1-score of 0.85. Table 2 reports the detailed numerical results. Moreover, it contains the values of the KPIs for two ad-hoc classifiers for time series from literature, **DTW** and **GAK** (last two rows of the table). These latter differ from traditional classifiers either in required execution time or in the performance of prediction.

The last four columns of Table 2 show the overall CPU time required to train the models (training) and to perform the stream prediction analysis (analysis), for the implementation on both a workstation and a sSBC. The execution of prediction tasks during analysis is almost instantaneous (at most half a second of cumulative CPU time) for traditional classifiers and for **DTW**, both on workstation and on sSBC. Instead, we had to arbitrarily stop the prediction process of **GAK**, because after 5 minutes no prediction was produced. That is, the CPU load required for stream prediction by **GAK** are unsuitable for our task, both on SBCs and on workstation. Therefore, we are not considering **GAK** in the remainder of our analyses.

DTW, similarly to **TREE**, has very high accuracy and low CPU effort for stream prediction analysis, but much lower F_1-score than the other traditional classifiers. Moreover, it requires higher CPU effort for training: one order of magnitude higher than traditional methods on the workstation, and 9 seconds on sSBC. That is, **DTW** is dominated in terms of quality by other methods, which also require less computational load (*e.g.*, **TREE**). Therefore, also **DTW** is excluded from the remainder of our analyses.

5.3 Experiments E2 and E3: classification of data from a new company

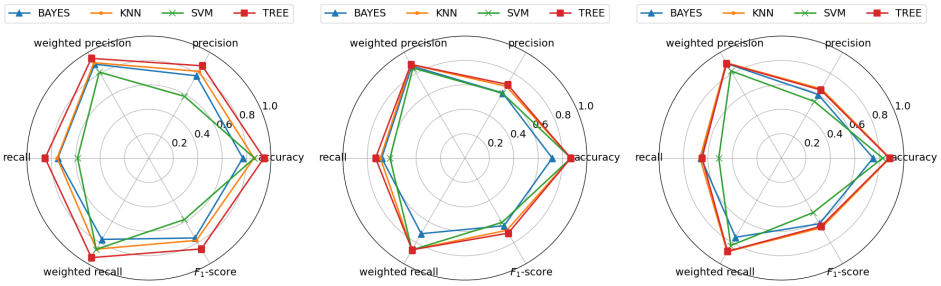
Experiments E2 and E3 compare classifiers trained on data of a company and tested on stream of data from a different company.

In experiment E2, dataset D1 is used for training, while testing is performed on dataset D3. Experiment E3 extends E2: while testing is still performed against dataset D3, the training is performed using both datasets D1 and D2, each of which comes from a distinct company. Figures 3b and 3c show spider-charts of KPI values of each classifier, averaged over all sensors, for experiments E2 and E3 respectively. Detailed numerical results are available in Supplementary Materials.

Table 2: Experiments E1: KPI values per classifier averaged over all sensors

		workstation KPIs						workstation exe. time		sBC exe. time	
library	classifier	accuracy	precision	weigh. precision	recall	weigh. recall	F_1-score	training CPU (s)	analysis CPU (s)	training CPU (s)	analysis CPU (s)
`scikit-learn`	BAYES	0.76	0.77	0.88	0.74	0.76	0.75	0.03	0.01	0.10	0.01
	KNN	0.86	0.82	0.90	0.75	0.86	0.77	0.04	0.06	0.11	0.18
	SVM	0.86	0.5	0.81	0.58	0.86	0.58	0.04	0.13	0.01	0.51
	TREE	0.94	0.87	0.94	0.85	0.94	0.85	0.05	0.01	0.12	0.01
`pyts`	DTW	0.93	0.70	0.92	0.69	0.93	0.69	0.75	0.02	9.02	0.02
`tslearn`	GAK	–	–	–	–	–	–	0.74	*	7.95	*

* stream prediction analysis stopped after 5 minutes.



(a) Experiments E1 (b) Experiments E2 (c) Experiments E3

Fig. 3: Spider-charts of KPI values per classifiers, averaged over all sensors

As we are interested in the change of performance of the classifiers when data from a new company is used, we focus on the difference between the values of KPI resulting from successive experiments, that is we compare E2 against E1 and E3 against E2. Such differences can be noticed visually from the comparison between Figures 3b and 3a and between Figures 3c and 3b, respectively: all figures have same scale from 0 to 1; the farther a point from the center of the chart, the better.

Comparing E2 with E1, we notice that all classifiers (Fig. 3b) worsen their performance, with a maximum decrease of -0.20. Comparing E3 with E2 we notice that worsenings have a maximum between -0.1 and -0.15.

classifier	accuracy	precision	recall	F_1-score
BAYES	0.39	0.58	0.28	0.29
KNN	0.83	0.83	0.75	0.75
SVM	0.83	0.91	0.83	0.83
TREE	0.61	0.50	0.38	0.40

(a) Per classifier, averaged over all sensors

topic	accuracy	precision	recall	F_1-score
I1	0.67	0.75	0.58	0.60
I2	0.74	0.62	0.49	0.50
I3	0.75	0.75	0.62	0.63
I4	0.003	0.37	0.001	0.003
I5	1.00	1.00	1.00	1.00
I6	0.83	0.75	0.66	0.68

(b) Per sensor, averaged over all classifiers

Table 3: Experiment E4: KPI values

5.4 Experiment E4: testing on unknown topic

In experiment E4 classifiers are trained with dataset D1 and tested on D2. This latter contains only a single sensor, I7, whose topic is not present in D1. This experiment is aimed at verifying how classifiers behave towards an unknown topic. For binary classification, the desired result is for a classifier to always answer with a `false`.

Table 3a shows KPI values of each classifier, averaged over all sensors. **KNN** and **SVM** have good performance in correctly identifying the data as not belonging to the training class, with both accuracy and F_1-score around 0.8. The classifier with worst performance is **BAYES** with only 0.3 of F_1-score and 0.4 of accuracy. Table 3b shows KPI values of each sensor used for training phase, averaged over all binary classifiers. While I5 is never misclassified as I7, I4 has very poor performance, being almost always misclassified.

5.5 Discussion

We use the results of these experiments as support for the choice of the classification algorithms to use in the next phase of our approach.

We consider only traditional classifiers with standard features, that, while granting good values of KPI, also need shorter execution time than ad-hoc classification algorithm for time series from the literature (see results of experiment E1 in Table 2).

The more promising classifiers are **KNN** and **TREE**: while the former has best overall performance when considering all experiments from E1 to E4, **TREE** has best performance for experiments from from E1 to E3, while performing poor for experiment E4. We discard **BAYES** being the worst classifier when considering unknown sensors (experiment E4) and **SVM** being the worst classifier when considering known sensors (experiments E1 to E3). Therefore, in the next phase, we consider only **KNN** and **TREE** for binary classification.

6 Evaluating the performances of SIP full pipeline

In this section we consider the SIP algorithm in full. That is, using as input the data stream of all sensors simultaneously, we generate a mapping between each sensor and either a specific (known) topic or the *unknown* topic.

Experimental setup

We solve each instance of the optimization problem (2)–(5) as a minimum cost flow on a compact graph, using the open-source Python library OR-Tools (Perron & Furnon, 2023). The computing load was negligible. The remaining parts of SIP (*e.g.*, the sampling and aggregation steps in the adaptive improvement) are implemented with custom Python code.

We run experiments with the same setting of experiment E3 of Section 5.3, where training is performed on datasets D1 and D2 and testing is performed on dataset D3 that contains streams of 6 sensors. To obtain results which are more statistically relevant, we created 50 random test instances from the real one, by randomly permuting the blocks of time slots of each sensor in D3, independently one another. This simulates instances in which the same machines perform the same operations over time as in D3, but in random order.

The set I of sensors is composed by the 6 sensors found in D3, while the set K of topics is composed by the union of sensors found in D1 and D2, which correspond to 7 known topics (see Table 1). We consider three choices of the ensemble function $g()$, which computes probabilities π_{ik} as in equation (1). The first two implementations use only one binary classifier to compute π_{ik}, namely **KNN** or **TREE**, that is,

$$g^1(\rho_{ik}^{\text{KNN}}, \rho_{ik}^{\text{TREE}}, \rho_{ik}^{\text{BAYES}}, \rho_{ik}^{\text{SVM}}) = \rho_{ik}^{\text{KNN}}$$

and

$$g^2(\rho_{ik}^{\text{KNN}}, \rho_{ik}^{\text{TREE}}, \rho_{ik}^{\text{BAYES}}, \rho_{ik}^{\text{SVM}}) = \rho_{ik}^{\text{TREE}}$$

The third implementation uses a convex combination of outcomes of **KNN** and **TREE**, that is

$$g^3(\rho_{ik}^{\text{KNN}}, \rho_{ik}^{\text{TREE}}, \rho_{ik}^{\text{BAYES}}, \rho_{ik}^{\text{SVM}}) = \alpha^{\text{KNN}} \rho_{ik}^{\text{KNN}} + \alpha^{\text{TREE}} \rho_{ik}^{\text{TREE}}.$$

We experimented by setting each α to the value of one of the KPIs obtained from training. The rationale for this latter scenario is to use an ensemble of binary classifiers, weighing more those with better performances for each sensor. In our final implementation, we set each α as the F_1-score of the corresponding classifier, divided by the sum of the two F_1-score values. The value of parameter N_k, that is, the expected maximum number of sensors whose topic is k, is set to 1 for the real topics, and to $|I|$ for the unknown topic. That is, only one sensor can be classified as corresponding to each real topic, but possibly all sensors may be classified as unknown. The parameters $\pi_{i,\text{u}}$ are set to a constant value for each sensor, which we arbitrarily set as 0.025 in our experiments. Intuitively, these values act as a threshold, under which the likelihood of real topics is too low to be used to reliably classify a sensor in the matching (and therefore the unknown topic is used as a fallback).

Given that the maximum amount of data available among all sensors in dataset D3 (see Table 1) is 40 hours, we select Q_i to be 5, 10, 20, and 30 hours of data, constant for all $i \in I$.

We set the number of rounds P to 50, and we select observation time Q_i^p as follows: for each round $p \in P$, we draw at random a starting time-slot $t \in T$ followed by at least Q_i slots, independently for each stream $i \in I$. Finally, we compare two aggregation policies: first, to select the assignment provided by the resolution of model (2)–(5) with highest objective function (*i.e.*, with highest likelihood) among all time-slots; second, to select the assignment that appears more frequently as result of the matching over all time-slots (*i.e.*, the mode).

KPI

As KPI, we consider the accuracy of the assignment $\mu()$ resulting from the aggregation policy over all mappings $\mu^p()$, which in turn result from model (2)–(5). The accuracy is measured as the fraction of sensors correctly classified. As comparison, we report that the average accuracy of an assignment resulting from choosing values of the score π_{ik} uniformly at random is about $1/|I|$, that is $1/6 = 0.1\bar{6}$ in our case.

6.1 Best set-up of the full pipeline

Fig. 4a shows the boxplots of accuracy provided by the best set-up of the full pipeline of our algorithm, which consider as aggregation policy the assignment that appears most often as result of the matching over all time-slots (*i.e.*, the mode). Extended results, which include the assignment with maximum likelihood as aggregation policy, are presented in Supplementary Materials.

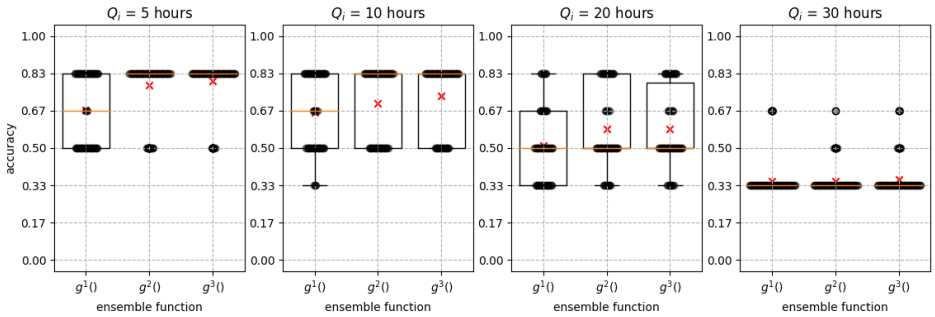
Fig. 4a is composed by four sub-figures, column-wise, each being related to a value of observation time Q_i (as indicated by the label on top). Each of these sub-figures contain three boxplots, one per implementation of the ensemble function $g()$ (as indicated by labels the x-axis).

Each boxplot is built with the values of accuracy of our algorithm on the 50 random test instances obtained from D3: all single values are represented as points within the boxplot, stacking up in horizontal bars (the longer the bar, the higher the number of runs yielding that accuracy). The orange line in the middle of each boxplot identifies the median value, the cross mark identifies the average value.

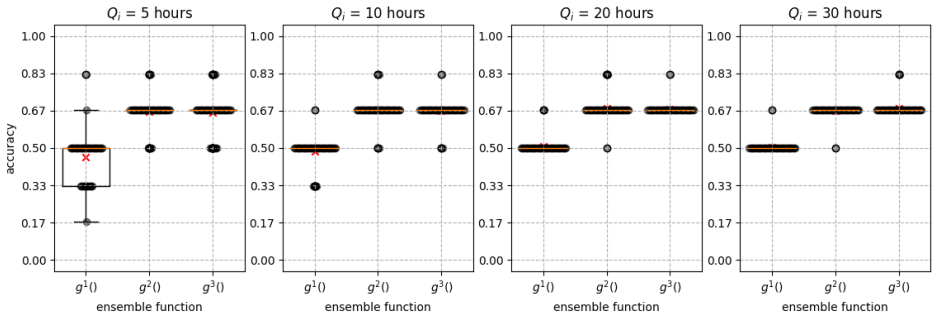
Median values of boxplots are either 0.67 or 0.83: we recall that in our experiments $|I| = 6$, and hence these median values are considerably higher than the accuracy of random likelihood $1/|I| = 1/6 = 0.1\bar{6}$. The best solution is given by the following setting: using 5 hours of time-range Q_i and $g^3()$ as ensemble function, that is, the convex combination of **KNN** and **TREE**; the corresponding boxplot is the the third in the first group of Fig. 4a. In a very large set of instances, it was possible to find the correct topic for 5 sensors over 6.

Accuracy per stream

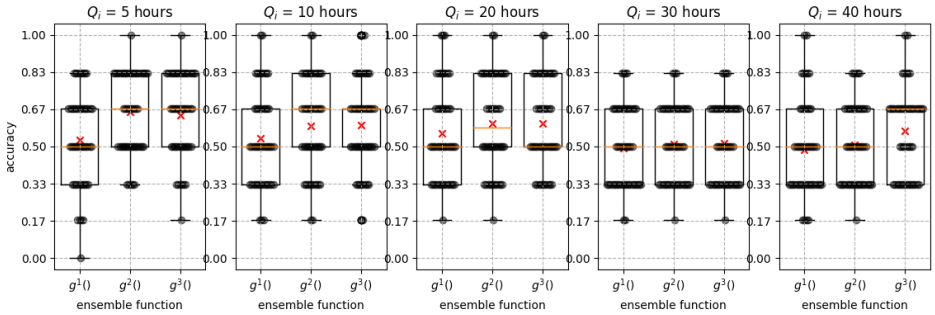
For the best setting found, we analyze the accuracy of classification of each sensor and each topic. That is, we compute the relative frequency of assignments for each sensor $i \in I$ to each topic $k \in K$ among all P rounds of the algorithm and all replications.



(a) Best set-up of SIP full pipeline



(b) Variant without matching algorithm



(c) Variant without adaptive aggregation

Fig. 4: Boxplots of accuracy of SIP

Table 4 contains such values: each row is related to a sensor, each column is related to a topic and each cell in row i column k contains the relative frequency of assignments of a sensor i to topic k, considering all rounds of our algorithm and all replications (that is $50 \cdot 50 = 2500$ assignments). The correct outcome corresponds to the cells in the diagonal of the table: we highlight such cells with gray background.

Table 4: Relative frequency of assignment of sensors (rows) to topics (columns) among all assignments of all experiments: the highest value per sensor is underlined; diagonal cells (highlighted with gray background) are correct mappings.

		Topics (set K)							
		κ1	κ2	κ3	κ4	κ5	κ6	κ7	unknown
Sensor (set I)	ι1	0.8988	0	0	0.0004	0.0796	0	0.0028	0.0184
	ι2	0	0.5872	0	0.0108	0	0.3848	0.008	0.0092
	ι3	0.0444	0	0.6552	0.0012	0.1728	0	0.0124	0.114
	ι4	0	0.0024	0	0.9876	0	0.0092	0	0.0008
	ι5	0.0232	0	0.3292	0	0.0948	0	0	0.5528
	ι6	0	0.3996	0	0	0	0.5996	0	0.0008

The highest value for each stream (*i.e.*, for each row) is underlined: in 5 cases out of 6 the highest relative frequency lies in the diagonal; while in three cases the mapping is clearly correct (for ι1, ι3 and ι4), a mismatch occurs often, but to a minority extent, between sensors ι6 and ι2. Finally, the only bad performance is observed for ι5 that is classified mostly as *unknown* or κ3.

6.2 Assessing necessity for pipeline components

We compare our results with two benchmarks. The first one (Fig. 4b) is a greedy method, not using the mathematical programming step for matching. The second one (Fig. 4c) is a monolithic approach, which is using the whole set of time slots at once, without using the adaptive mapping stage. They are detailed and discussed in the following. We kept the same experimental setting and implementation presented at the beginning of this Section.

6.2.1 Matching algorithm

First, we experiment on the variant of our algorithm that does not make use of the matching algorithm to build the mapping $\mu^p()$ at the end of each round p. As replacement for the matching algorithm, we use a simple heuristic that assign every sensor $i \in I$ to the topic with maximum score $\pi^p_{i,k}$, *i.e.*, $\mu^p(i) = \arg\max_{k \in K} \pi^p_{i,k}$.

We remark that our full algorithm provides a theoretical a-priori guarantee to comply constraint (4), while such heuristic may assign the same topic to more than one stream of data and hence does not guarantee to comply constraint (4).

Nevertheless, the results of such heuristic are presented in Fig. 4b. We show the best results provided by the heuristic, which are related to the aggregation policy that consider the assignment with highest likelihood. Results for other configurations can be found in the Supplementary Materials.

The best setting has a median accuracy of 0.67, and is obtained by using $g^3()$ as the ensemble function and either 20 or 30 hours for Q_i. We conclude that for the experimental instances, solely considering the highest values of $\pi_{i,k}$ is enough to correctly mapping 4 streams out of 6, but the full SIP has better performance, correctly mapping 5 streams over 6 (as can be seen in Fig. 4a).

That is, the SIP is superior both in terms of theoretical guarantees, and in terms of experimental performance, requiring a similar computing effort.

6.2.2 Rounds and aggregation policy

Second, we experiment on the variant of our algorithm that does not make use of the P rounds of draw of time-slots. Consequently, it does not need an aggregation policy either: the mapping resulting from the matching algorithm of the first Q_i time-slots is also the final mapping.

Results of this variant are presented in Fig. 4c. The median value of accuracy is never higher than 0.67, while the full pipeline of SIP reaches a median of 0.83. Average accuracy is also systematically lower than that of the full algorithm. Moreover, boxplots show higher variability of the values of accuracy if compared to results of the full pipeline in Fig. 4a: the former results from a mapping of a single matching, while the latter results from the aggregation of the P matching from all rounds, whose purpose is precisely to reduce variability. It is also interesting to note that the results for this method tend to get *worse* as the length of Q_i increases (even if longer Q_i means more data which is allowed to be used for the final prediction).

6.3 Discussion

Our experiments allow to identify which settings yield to maximum median accuracy of classification, in terms of: (i) time range of data stream to consider, (ii) classification algorithms and ensemble function to choose and (iii) choice of the aggregation of the mappings as final solution, among all those generated in the observation time range.

Indeed, with the use of the best settings, the topic of 5 out of 6 sensors is correctly classified in median. Our results highlight that misclassifications are related always to a specific subset of sensors. We tend to ascribe such misclassifications to changes in working processes and operators' behavior in different companies. Such information are missing in our dataset, and further analyses may consider them to improve performances of classification.

Moreover, we highlight that all components of SIP pipeline are needed to achieve highest classification accuracy: (i) discarding matching algorithm leads to the loss in overall accuracy both theoretically and experimentally and (ii) discarding P rounds of sampling and adaptive aggregation leads to a loss in overall accuracy together with higher variability in results.

7 Conclusions

The main target of our research is to ease the adoption of Industry 4.0 standards for companies in the manufacturing sector, which is characterized by the extensive use of industrial machinery. In fact, our SIP algorithm helps to achieve seamless integration between new machinery and the information system of the manufacturing company: when companies introduce new machinery, either I4.0-ready or requiring retrofitting, SIP automatically configures the mapping between the data stream produced by sensors on the machine and their semantics, reducing the costly, tedious and long process

of manual configuration. Such integration enables fast and reliable retrieval of information from the manufacturing machines, and their feeding to decision-support tools.

From a methodological point of view, despite having a clear time series structure at its core, our analyses reveal that standard approaches from the literature do not apply nicely to our industrial application. The reasons are many, but the most intuitive one is the following: in our context, many series (the data stream of each sensor) need to be classified *at once*, being the outcomes *mutually exclusive*. Current methods are unable to handle such a peculiarity. Additionally, the trained models are meant to be used in resource-constrained environments, such as SBCs, and therefore a suite of traditional approaches like deep learning are inapplicable.

These observations led us to design our SIP algorithm. It is a pipeline of models, combining (a) extraction, splitting and feature evaluation on stream blocks; (b) supervised learning models, to classify each block independently; (c) ensemble functions, merging classifications on several blocks from different classifiers; (d) an exact matching algorithm, to manage the mutually exclusive mapping requirement; (e) a sampling and aggregation method, to improve accuracy when longer streams of data are available during the usage of the pipeline.

The SIP shows to be effective also in a challenging setting, in which training and testing are performed on data from different companies. Our experiments highlighted that each part of the SIP contributes to the final outcome. We mention three quantitative results showing how the advantage of the SIP is achieved: the ensemble classification process of step (c) allows to get F_1-scores as high as 0.85, against scores of 0.69 of competitors from the literature; using exact matching algorithms in step (d) allows both to have a *theoretical guarantee* of mutually exclusive classification outcomes, and to increase the average accuracy by about 10%; the use of dedicated sampling and aggregation in step (e) similarly grants average accuracy increases of about 13% on longer data streams.

That is, the improvements of our pipeline do not limit to the handling of mutual exclusion: the ensemble method used in step (c) improves with respect to state-of-the-art time series classification models even in the *single* time series classification task. The structure of data itself might contribute to the poor relative performance of state of the art algorithms: instead of a set of well diversified series, our streams are very few (although long) sequences of data.

The integration of such a set of diversified models is not without drawbacks: SIP shows to work well *provided* some careful choices are made for its components. Among the most important algorithmic details we list the choice of specific supervised learning models in the classification and ensemble steps (namely, combining KNN and Decision Trees), the use of an exact mathematical programming algorithm in the matching step, the use of sampling with a particular choice of the final representative in the adaptive aggregation step.

We also stress that the aspect of computational effort is promising: the CPU time required to use our models (after training) is always fractions of seconds. This is certainly helpful if the pipeline runs as a service in a company server. More in perspective, it opens up also an even more ambitious industrial scenario, in which pipeline models

are deployed as micro-services, directly running when needed in an SBC acting as a smart, self-configuring, interface.

We finally report a more prospective outcome: our research provides a clear example on how machine learning and mathematical programming gracefully combine in solving complex industrial problems. Our methods allow to exploit the flexibility of data-driven models, to improve their accuracy by optimization, and simultaneously to obtain formal a-priori guarantees that constraints on the solutions structure are respected.

Statements and Declarations

Competing interests

The authors report there are no competing interests to declare.

Data availability

Restrictions apply to the availability of these data, which were used under license for this study. The data that support the findings of this study are available upon request to Par-Tec SpA.

References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B. (1993). *Network flows: Theory, algorithms, and applications*. USA: Prentice-Hall, Inc.

Ai, Y., Wang, Y., Han, S., Ye, C. (2024). Investigation of cladding layer formation in uphill wire laser additive manufacturing on inclined substrate. *Applied Thermal Engineering*, *247*, 122919,

Alqoud, A., Schaefer, D., Milisavljevic-Syed, J. (2022). Industry 4.0: a systematic review of legacy manufacturing system digital retrofitting. *Manufacturing Review*, *9*, 32,

Bradde, T., Fracastoro, G., Calafiore, G.C. (2021). Multiclass sparse centroids with application to fast time series classification. *IEEE Transactions on Neural Networks and Learning Systems*, ,

Cuturi, M. (2011). Fast global alignment kernels. *Proceedings of the 28th international conference on machine learning (icml-11)* (pp. 929–936).

Escobar, C.A., McGovern, M.E., Morales-Menendez, R. (2021). Quality 4.0: a review of big data challenges in manufacturing. *Journal of Intelligent Manufacturing*, *32*, 2319–2334,

Faouzi, J. (2022). Time series classification: A review of algorithms and implementations. *Machine Learning (Emerging Trends and Applications)*, ,

Faouzi, J., & Janati, H. (2020). pyts: A python package for time series classification. *Journal of Machine Learning Research*, *21*(46), 1-6, Retrieved from http://jmlr.org/papers/v21/19-763.html

Farahani, M.A., McCormick, M., Gianinny, R., Hudacheck, F., Harik, R., Liu, Z., Wuest, T. (2023). Time-series pattern recognition in smart manufacturing systems: A literature review and ontology. *Journal of Manufacturing Systems*, *69*, 208-241, https://doi.org/https://doi.org/10.1016/j.jmsy.2023.05.025 Retrieved from https://www.sciencedirect.com/science/article/pii/S0278612523000997

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

Haskamp, H., Orth, F., Wermann, J., Colombo, A.W. (2018). Implementing an opc ua interface for legacy plc-based automation systems using the azure cloud: An icps-architecture with a retrofitted rfid system. *2018 ieee industrial cyber-physical systems (icps)* (pp. 115–121).

Hsu, C.-Y., & Liu, W.-C. (2021). Multiple time-series convolutional neural network for fault detection and diagnosis and empirical study in semiconductor manufacturing. *Journal of Intelligent Manufacturing*, *32*, 823–836,

Josbert, N.N., Wei, M., Ping, W., Rafiq, A. (2024). A look into smart factory for Industrial IoT driven by SDN technology: A comprehensive survey of taxonomy, architectures, issues and future research orientations. *Journal of King Saud University-Computer and Information Sciences*, 102069,

Khan, H.U., Ali, F., Nazir, S. (2024). Systematic analysis of software development in cloud computing perceptions. *Journal of Software: Evolution and Process*, *36*(2), e2485,

Li, Z., Mei, X., Sun, Z., Xu, J., Zhang, J., Zhang, D., Zhu, J. (2024). A reference framework for the digital twin smart factory based on cloud-fog-edge computing collaboration. *Journal of Intelligent Manufacturing*, 1–21,

Matt, D.T., Modrák, V., Zsifkovits, H. (2020). *Industry 4.0 for smes: Challenges, opportunities and requirements*. Palgrave Macmillan.

Nain, G., Pattanaik, K., Sharma, G. (2022). Towards edge computing in intelligent manufacturing: Past, present and future. *Journal of Manufacturing Systems*, *62*, 588–611,

Nanopoulos, A., Alcock, R., Manolopoulos, Y. (2001). Feature-based classification of time-series data. *International Journal of Computer Research*, *10*(3), 49–61,

Nayernia, H., Bahemia, H., Papagiannidis, S. (2022). A systematic review of the implementation of industry 4.0 from the organisational perspective. *International Journal of Production Research*, *60*(14), 4365–4396,

Olsen, T.L., & Tomlin, B. (2020). Industry 4.0: Opportunities and challenges for operations management. *Manufacturing & Service Operations Management*, *22*(1), 113–122,

Oztemel, E., & Gursev, S. (2020). Literature review of industry 4.0 and related technologies. *Journal of intelligent manufacturing*, *31*, 127–182,

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830,

Perron, L., & Furnon, V. (2023). *OR-Tools*. https://developers.google.com/optimization/. Google.

Qi, Q., & Tao, F. (2019). A smart manufacturing service system based on edge computing, fog computing, and cloud computing. *IEEE access*, *7*, 86769–86777,

Rai, R., Tiwari, M.K., Ivanov, D., Dolgui, A. (2021). Machine learning in manufacturing and industry 4.0 applications. *International Journal of Production Research*, *59*(16), 4773–4778,

Rosati, R., Romeo, L., Cecchini, G., Tonetto, F., Viti, P., Mancini, A., Frontoni, E. (2023). From knowledge-based to big data analytic model: a novel iot and machine learning based decision support system for predictive maintenance in industry 4.0. *Journal of Intelligent Manufacturing*, *34*(1), 107–121,

Ruiz, A.P., Flynn, M., Large, J., Middlehurst, M., Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *35*, 401–449,

Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, *26*(1), 43–49,

Sanchez-Londono, D., Barbieri, G., Fumagalli, L. (2023). Smart retrofitting in maintenance: a systematic literature review. *Journal of Intelligent Manufacturing*, *34*(1), 1–19,

Schäfer, P. (2015). The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, *29*, 1505–1530,

Schäfer, P., & Leser, U. (2023). Weasel 2.0: a random dilated dictionary transform for fast, accurate and memory constrained time series classification. *Machine Learning*, ,

Sehr, M.A., Lohstroh, M., Weber, M., Ugalde, I., Witte, M., Neidig, J., ... Lee, E.A. (2020). Programmable logic controllers in the context of industry 4.0. *IEEE Transactions on Industrial Informatics*, *17*(5), 3523–3533,

Shinde, S.V., Hemanth, D.J., Elhoseny, M. (2023). Introduction to different computing paradigms: cloud computing, fog computing, and edge computing. *Intelligent edge computing for cyber physical applications* (pp. 1–16). Elsevier.

Sun, D., Hu, J., Wu, H., Wu, J., Yang, J., Sheng, Q.Z., Dustdar, S. (2023). A comprehensive survey on collaborative data-access enablers in the iiot. *ACM Computing Surveys*, *56*(2), 1–37,

Tavenard, R., Faouzi, J., Vandewiele, G., Divo, F., Androz, G., Holtz, C., ... Woods, E. (2020). Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, *21*(118), 1-6, Retrieved from http://jmlr.org/papers/v21/20-091.html

Usuga Cadavid, J.P., Lamouri, S., Grabot, B., Pellerin, R., Fortin, A. (2020). Machine learning applied in production planning and control: a state-of-the-art in the era of industry 4.0. *Journal of Intelligent Manufacturing*, *31*, 1531–1558,

Wolsey, L. (2020). *Integer programming.* Wiley.

Xi, L., Wang, W., Chen, J., Wu, X. (2023). Appending-inspired multivariate time series association fusion for tool condition monitoring. *Journal of Intelligent Manufacturing*, 1–14,

Xu, L.D., Xu, E.L., Li, L. (2018). Industry 4.0: state of the art and future trends. *International Journal of Production Research*, *56*(8), 2941–2962,

Zheng, Y., Liu, Q., Chen, E., Ge, Y., Zhao, J.L. (2014). Time series classification using multi-channels deep convolutional neural networks. *International conference on web-age information management* (pp. 298–310).

Appendix A Settings of classification algorithms

Table A.1 presents the settings for all hyperparameters of the classifiers used in experiments of Section 5. For each classifier we present the value of the hyperparameters of its Python 3.12.3 implementation:

- **SVM**: C-Support Vector Classification of `sklearn.svm.SVC` in `scikit-learn` library, version 1.2.1;
- **KNN**: k-neareast neighbors of `sklearn.neighbors.KNeighborsClassifier` in `scikit-learn` library, version 1.2.1;
- **TREE**: decision tree of `sklearn.tree.DecisionTreeClassifier` in `scikit-learn` library, version 1.2.1;
- **BAYES**: Gaussian naïve Bayes of `sklearn.naive_bayes` in `scikit-learn` library, version 1.2.1;
- **DTW**: k-neareast neighbors with DTW metric of `pyts.classification.KNeighborsClassifier` in `pyts` library, version 0.12.0;
- **GAK**: C-Support Vector Classification with global alignment kernel of `tslearn.svm.TimeSeriesSVC` in `tslearn` library, version 0.5.3.

Appendix B Key performance indicators

We evaluate the performance of the classifiers with traditional key performance indicators (KPI). For a formal definition of our KPIs, let: n be the number of sample to predict, P be the number of observations that actually belong to the class (positive observation) and N the number of observations that do not belong to the class (negative observation), and let TP, TN, FP and FN be the value of true positive, true negative, false positive and false negative found during classification, correspondingly.

As a result, we define the KPIs as follows:

- the accuracy, that is the fraction of correct predictions $= \frac{TP+TN}{n}$
- the precision, that measures the ability of a classifier to avoid misclassifying an observation $= \begin{cases} 0 & \text{if } TP+FP=0 \\ \frac{TP}{TP+FP} & \text{otherwise} \end{cases}$
- the recall, that measures the ability of the classifier to predict all the observation belonging to the class $= \begin{cases} 0 & \text{if } TP+FN=0 \\ \frac{TP}{TP+FN} & \text{otherwise} \end{cases}$
- the F_1-score, that is the harmonic mean of precision and recall and synthetically summarize both metrics $= 2\frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$

Moreover, weighted versions of precision and recall are considered, in which metrics are computed for each label separately, and then average weighted by the support of the label (that is the number of actual occurrences of the label in the dataset):

- weighted precision $= \frac{\sum_{l\in\text{labels}} \text{precision}(l)\cdot\text{support}(l)}{n}$
- weighted recall $= \frac{\sum_{l\in\text{labels}} \text{recall}(l)\cdot\text{support}(l)}{n}$

Table A.1: Settings of hyperparameters of the classifiers in their Python 3.12.3 implementation

SVM	KNN	TREE	BAYES	DTW	GAK
kernel=rbf	n_neighbors=5	criterion=gini	priors=None	metric=dtw	c=1.0
gamma=scale	weights=uniform	splitter=best	var_smoothing=1e−9	n_neighbors=5	kernel=gak
C=1	algorithm=auto	max_depth=None		algorithm=auto	degree=3
degree=3	leaf_size=30	min_samples_split=2		weights=uniform	gamma=auto
coef0=0	p=2	min_samples_leaf=1		leaf_size=30	coef0=0
shrinking=True	metric=minkowski	min_weight_fraction_leaf=0		p=2	shrinking=True
probability=False	metric_params=None	max_features=None		metric_param=1	probability=True
tol=0.001	n_jobs=None	random_state=None		n_jobs=1	tol=1e − 03
cache_size=200		max_leaf_nodes=None			cache_size=200
class_weight=None		min_impurity_decrease=0			verbose=0
verbose=False		class_weight=None			max_inter=1
decision_function_shape = ovr		ccp_alpha=0			decision_function_shape =ovr
break_ties=False					
random_state=None					
max_iter=-1					