Received: 28 November 2020 | Revised: 18 May 2021 | Accepted: 23 June 2021

DOI: 10.1111/exsy.12762

ORIGINAL ARTICLE

Expert Systems WILEY

Rule-based preprocessing for data stream mining using complex event processing

Aurora Ramírez[1] | Nathalie Moreno[2] | Antonio Vallecillo[2]

[1]Departamento de Informática y Análisis Numérico, Universidad de Córdoba, E-14071 Córdoba, Spain

[2]ITIS Software, Universidad de Málaga, Málaga, Spain

Correspondence
Aurora Ramírez, Departamento de Informática y Análisis Numérico, Universidad de Córdoba, E-14071 Córdoba, Spain.
Email: aramirez@uco.es

Funding information
Andalusian Regional Government, Grant/Award Number: DOC_00944; Spanish Government under project COSCA, Grant/Award Number: PGC2018-094905-B-I00; European Commission (FEDER); University of Córdoba.

Abstract

Data preprocessing is known to be essential to produce accurate data from which mining methods are able to extract valuable knowledge. When data constantly arrives from one or more sources, preprocessing techniques need to be adapted to efficiently handle these data streams. To help domain experts to define and execute preprocessing tasks for data streams, this paper proposes the use of active rule-based systems and, more specifically, complex event processing (CEP) languages and engines. The main contribution of our approach is the formulation of preprocessing procedures as event detection rules, expressed in an SQL-like language, that provide domain experts a simple way to manipulate temporal data. This idea is materialized into a publicly available solution that integrates a CEP engine with a library for online data mining. To evaluate our approach, we present three practical scenarios in which CEP rules preprocess data streams with the aim of adding temporal information, transforming features and handling missing values. Experiments show how CEP rules provide an effective language to express preprocessing tasks in a modular and high-level manner, without significant time and memory overheads. The resulting data streams do not only help improving the predictive accuracy of classification algorithms, but also allow reducing the complexity of the decision models and the time needed for learning in some cases.

KEYWORDS
complex event processing, data preprocessing, data stream mining, data streams, ECA rules

1 | INTRODUCTION

Data preprocessing is a crucial step in any knowledge discovery process (García et al., 2014). Cleaning, transforming, filtering, mapping and integrating data often represent laborious but essential tasks (Zhang et al., 2003). Furthermore, the application of an adequate preprocessing method can have a positive impact on the results of the mining process (Crone et al., 2006; Uysal & Gunal, 2014). However, data preprocessing is normally achieved using non-specialized languages, ad hoc operators and is usually performed using cumbersome and error-prone processes, which demand some knowledge of statistics and specialized programming skills. Domain experts could greatly benefit from high-level solutions and assistance to effectively perform preprocessing tasks (Bilalli et al., 2019). Tools like Weka and software packages like those available in R and Python provide support for common preprocessing tasks, but they require a considerable learning curve.

Besides, domain experts will most probably need to fine-tune parameters or adapt general methods to their specific application domain (Huang et al., 2016), making it difficult to reuse procedures and integrate them into expert systems.

The growing interest in knowledge discovery from data streams (Gama, 2010) is also demanding novel computational techniques for real-time data mining (Gaber, 2012; Ghomeshi et al., 2019). Despite its relevance, data preprocessing has not been so frequently studied in the context of data stream mining (Ramírez-Gallego et al., 2017). Most of the available methods try to adapt algorithms applied in traditional data mining or assume that data does not experience any variation over time. Even though specific approaches for dynamic feature selection (Barddal et al., 2019) or discretization (Ramírez-Gallego et al., 2018) are beginning to appear, they are rarely available in software libraries.

When the transformation and filtering of data streams should be done by domain experts, who may not have deep knowledge of data mining methods and tools, providing them with alternatives to express and implement preprocessing operations becomes essential. In this sense, active rule-based languages (Corporate Act-Net Consortium, 1996) could be explored as a novel and effective mechanism for specifying and implementing data preprocessing tasks in online data mining systems. On the one hand, experts might find it easier to express the required preprocessing steps in the form of rules, which are frequently adopted in expert systems due to its high interpretability (Grosan & Abraham, 2011). On the other hand, these kinds of languages provide mature constructs and operators for effectively manipulating data in event-driven architectures. In particular, fast-processing stream engines (Affetti et al., 2017) might represent a perfect candidate platform to implement preprocessing procedures, specially if the input data contain temporal information. Among them, complex event processing (CEP) systems allow users to detect situations of interest and propagate decisions (Cugola & Margara, 2012). A distinctive characteristic of these kinds of rule-based systems is that they offer rich languages to specify rules and patterns based on an SQL-like syntax. CEP has experienced a growing interest in the last years due to relevance that internet of things (IoT) applications are gaining for, for example, environmental monitoring (Gomes et al., 2020; Sun et al., 2019), healthcare (Loreti et al., 2019), smart mobility (Kousiouris et al., 2018), or waste management (Pardini et al., 2020).

In short, the idea underlying our proposal is that the specific characteristics of active rule-based systems, and more specifically CEP languages and engines – fast stream processing, rule-based engine and SQL-like syntax – make them especially well-suited for the specification and implementation of stream preprocessing tasks in online data mining processes. To the best of our knowledge, this paper represents the first time CEP is applied to define preprocessing rules for data streams. As a first contribution, we formulate data stream preprocessing as a rule-based procedure, for which we define several types of ECA (event-condition-action) rules serving different purposes, for example, data transformation and combination. Next, we show how CEP languages provide a set of concepts that are particularly well suited for data stream preprocessing – in particular, temporal and spatial windows, pattern matching and filtering, temporal correlations, and joining and splitting operators. These elements allow the generation of new features based on a short-term history, which might help to perform more accurate predictions in an incremental learning environment. Moreover, CEP is also able to naturally deal with multiple input flows and with rich datatypes.

To demonstrate the applicability of our proposal we provide an implementation[1] that integrates Esper,[2] a Java-based CEP engine, with massive online analysis (MOA), a library for online data mining (Bifet, Holmes, Kirkby, & Pfahringer, 2010). To evaluate our proposal, we have conducted three experiments using publicly available datasets from different domains. The experiments cover a representative set of preprocessing tasks, after which classification algorithms are applied. The results reveal that not only the specification of the preprocessing operations can be more naturally modelled using a rule-based approach, but also that the use of the data streams generated by CEP notably improves the prediction performance of the algorithms, reduce the complexity of some decision models and preserve time and memory requirements. Finally, we compare our approach with respect to other preprocessing alternatives in terms of performance and execution time, and discuss their advantages and limitations. The main contributions of this paper are summarized as follows:

- A novel rule-based approach to design domain-oriented preprocessing tasks for data streams, with formal definitions of different types of preprocessing rules.
- A publicly available implementation of the approach based on CEP, which also allows the execution of popular data mining algorithms over the resulting data streams.
- An experimental evaluation of the approach that analyzes the influence of different configurations of preprocessing rules on classification performance, time requirements and decision models.
- A comparison with other preprocessing techniques to analyse the trade-off between classification performance and execution time, which shows the advantages of using a CEP language instead of algorithms coded in general purpose programming languages.

The rest of the paper is organized as follows. Section 2 introduces the main concepts related to stream data mining, data stream preprocessing and CEP that will be used throughout the paper, and summarizes the related work. Our proposal for the use of rule-based systems, and in particular of CEP, for data stream preprocessing is explained in Section 3, and experimentally evaluated in Section 4. The strengths and limitations of the approach are discussed in Section 5, including an experimental comparison against other preprocessing techniques. Finally, Section 6 concludes and outlines future lines of research.

2 | FUNDAMENTAL CONCEPTS AND RELATED WORK

In this section, we introduce the fundamental concepts that will be used throughout the paper and review the main advances made in different application domains that are related to our approach. First, we briefly review the most outstanding works in the field of stream data mining in Section 2.1. Then, in Section 2.2, we analyse the contributions made in the area of data stream preprocessing that could constitute a starting point for our work. And finally, we give a brief introduction to CEP and highlight some proposals in the literature where this technology has been successfully used in conjunction with data mining.

2.1 | Stream data mining

A data stream is a potentially unbounded and ordered sequence of instances continuously observed over time (Gaber, 2012). Data stream mining differs from traditional mining processes in a number of aspects (Gama, 2010; Nguyen et al., 2015). From the data perspective, the set of instances is not available beforehand, so they have to be processed one by one or in chunks. Each instance can be accessed a limited number of times (usually only once) and then discarded to optimize the use of memory and storage space. This implies that the decision model is built incrementally, adding new relevant information and forgetting outdated data. Apart from being able to cope with these issues, stream mining algorithms are expected to provide real-time responsiveness, avoid data queuing and present low memory requirements. Due to the strict conditions in which these algorithms might operate, approximate results are acceptable.

Machine learning methods are popular techniques to carry out the mining step, also when the input data is a stream. Both supervised and unsupervised learning approaches have been developed in the last years (Jin & Agrawal, 2007; Nguyen et al., 2015). Regression and classification are supervised methods that try to predict the value or class of a variable, respectively. In contrast, clustering, whose purpose is to identify groups of related items, and association rule mining, oriented towards extracting patterns describing the data, belong to unsupervised learning.

Focusing on classification, traditional techniques like decision trees need to scan the dataset multiple times. Therefore, the building process has to be adapted to deal with the particularities of data streams. Other algorithms frequently applied in batch learning, such as k-nearest neighbours (kNN) and naive Bayes, do not require substantial modifications as they were originally designed to learn incrementally (Gama, 2010). Ensemble methods, that is, those building multiple classifiers, can be designed to learn from data streams too (Krawczyk et al., 2017), and have become reference techniques due to its good performance and adaptability (Cano & Krawczyk, 2019).

The performance of classification algorithms for stream mining can be severely affected when data streams are non-stationary, that is, the distribution of the attributes or the concept to be predicted changes over time (Krawczyk et al., 2017). This phenomenon, known as concept drift, frequently occurs in real-world data streams and can manifest under different patterns, for example, from gradual shifts to abrupt changes (Souza et al., 2020). Dealing with concept drift is an extensive and very active research field, in which different approaches have emerged (Cano & Krawczyk, 2019). Concept drift detectors are algorithms that analyse the stream in order to update the classifier when a change is detected (Liu, Song, et al., 2017; Lu et al., 2016). Sliding-window mechanisms create a buffer with samples that is continuously updated. Recent proposals in this area explore the use of multiple windows or dynamically adapt the window size (Liu, Zhang, & Lu, 2017; Zhang et al., 2017). Finally, classification algorithms can incorporate concept drift detection capabilities, so that they can adapt their learning process (Cano & Krawczyk, 2019; Hammami et al., 2020). Ensemble methods have been also proposed under this approach, updating the role of the base classifiers dynamically (Ancy & Paulraj, 2019; Cano & Krawczyk, 2020).

2.2 | Data stream preprocessing

Data preprocessing comprises techniques that enable the efficient adaption of the content or the format of raw data coming from multiple, possibly heterogeneous, sources (García et al., 2014). Enhancing the accuracy, completeness and consistency of data does not only make its management easier during the knowledge discovery process, but also has a positive influence on the effectiveness of mining algorithms (Han et al., 2012). Table 1 summarizes the principal preprocessing tasks, which are usually classified into two main areas, namely data preparation and data reduction (García et al., 2014).

TABLE 1 Data preprocessing tasks (adapted from [García et al., 2014])

	Data preparation		Data reduction
Cleaning	Detect and manage missing values and noise.	Feature selection	Discard irrelevant or redundant features.
Transformation	Convert data types and format.	Instance selection	Extract data samples, e.g., for validation.
Normalization	Scale numerical values.	Discretization	Divide continuous data into intervals.
Integration	Join data from different sources.	Value generation	Add new features or generate instances.

When dealing with data streams, preprocessing techniques should be as lighter and automatic as possible (Gaber et al., 2005). Additional factors need to be considered due to the dynamic nature of the data. For instance, accurate information regarding its distribution, for example, ranges of values or dependencies among variables, is not fully available. Due to these particularities, algorithms specifically conceived for data stream preprocessing have been recently proposed for both data preparation and data reduction tasks.

Focusing on data preparation, Bollegala proposes the so-called 'update equations' to estimate the mean and the standard deviation in the context of data normalization (Bollegala, 2017). His analysis reveals that a simple average estimation performs relatively well. Ideas from time series analysis can be used to deal with missing values. In this sense, the impact of temporal windows on seven imputation methods has been recently studied in the context of solar irradiance forecasting (Demirhan & Renwick, 2018). The compared methods range from simple statistics, for example, mean, median and mode, to both linear and nonlinear interpolation. Experiments were conducted injecting missing values at different time rates, a factor that has proven to influence the effectiveness of imputation methods.

Regarding data reduction, dynamic feature selection is probably the most studied topic (Barddal et al., 2017). Recent approaches rely on different statistics that incrementally determine the relevance of each feature (Barddal et al., 2019; Bolon-Canedo et al., 2016). The first method sorts features using the χ^2 metric, counting how many times each category appears in instances of the same class. The second method updates a merit function based on entropy that maximizes feature relevance while minimizing redundancy. Interestingly, sometimes the time overhead due to feature selection is compensated with a reduction of the learning time. Finally, studies for stream discretization adapt clustering, entropy and frequency-based methods to work in an online setting (Gama & Pinto, 2006). A recent proposal is local online fusion discretizer (LOFD), a discretization algorithm that updates interval definition by splitting and merging intervals (Ramírez-Gallego et al., 2018). LOFD, together with other discretization algorithms, is available in MOAReduction (Ramírez-Gallego et al., 2017), an extension of the popular MOA tool for massive online learning (Bifet, Holmes, Kirkby, & Pfahringer, 2010). This extension also contains filters for feature and instance selection.

2.3 | Complex event processing

CEP is a form of information processing aimed at detecting situations of interest from events (Luckham, 2001). In CEP terminology, a *simple* event corresponds to low-level data, for example, sensor measurements, whereas *complex* events are those derived from simple ones. In order to infer the occurrence of complex events, the expert has to specify rules, which are defined by means of patterns that identify the events to select, and the actions to be accomplished by the rule (Cugola & Margara, 2012). Traditional CEP systems follow a *publish-subscribe* approach, where subscribers collect rule outcomes to further process them.

Events, rules and patterns are defined using an *Event Processing Language* (EPL). In some CEP implementations, the language allows expressing the rules in a declarative way, following an SQL-like syntax. In such a case, the rule is usually comprised of three parts: (a) select, in which the relevant attributes are indicated (or the whole event, using *); (b) from, in which event flows or patterns over them are specified; and (c) where, in which the conditions to be fulfilled are detailed. The expressiveness of the EPL language is reflected in its broad range of mathematical, logical and temporal operators, and the flexibility to combine them. Table 2 collects the main types of operators, together with a brief description of their purpose.

Another distinctive characteristic of CEP is the definition of *windows* to limit the scope of the rule. A window can be *spatial*, that is, composed of a number of events, or *temporal*, that is, time-based defined. Listing 1 shows an example of a simple CEP rule that returns the date and identifier of bank transactions for those cases when the total transferred amount in 10 min exceeds 300,000 euros.

Listing 1: Example of a complex event processing rule using SQL syntax

```
select date, id.
from transaction#time(10 minutes).
where sum(amount) > 300,000
```

Given that manually defining CEP rules is a laborious task, some authors have tried to infer them using data mining and other techniques. A first proposal, iCEP, breaks down the problem of rule definition into several steps, such as identifying relevant event attributes or determining a proper window size (Margara et al., 2014). Similarly, autoCEP learns sequences from time series and transforms them into ready-to-deploy CEP rules (Mousheimish et al., 2017). Recently, GP4CEP proposes the use of genetic programming to build the rules that best describe the occurrence of a predefined complex event (Bruns et al., 2019). Finally, Adaptive CEP is focused on rule update, for which clustering and Markov probabilistic models are applied (Lee & Jung, 2017).

TABLE 2 Types of operators in CEP (adapted from Cugola & Margara, 2012)

Selection	Filter events establishing content constraints.
Projection	Extract pieces of information from the events.
Logic	Build conjunctive, disjunctive and negation expressions.
Sequence	Select events using temporal or order conditions.
Flow	Join, divide and sort several event streams.
Creation	Generate new flows and insert events.
Arithmetic	Compute mathematical expressions and statistics.

Abbreviation: CEP, complex event processing.

Other synergies between CEP and data mining have been explored in the last years, specially in the context of predictive analytics (Flouris et al., 2017; Fülöp et al., 2012). However, these approaches are mostly focused on applying learning algorithms to enhance CEP capabilities. For instance, ReCEPTor implements three association rule mining algorithms as EPL clauses to discover patterns as new events arrive (Ölmezogullari & Ari, 2013). More recently, Evolving Bayesian Networks have been applied to model changes in the distribution of the event stream processed by CEP (Wang et al., 2018). Finally, complex events detected by manually defined CEP rules have been used as the target to build predictive models (Fülöp et al., 2012).

3 | RULE-BASED PREPROCESSING FOR DATA STREAMS

This section presents our approach to express preprocessing procedures as ECA rules. Firstly, we discuss the principles guiding our solution and provide a formal definition of the term *preprocessing rule*. Then, we particularize this general concept in four types of rules to cover different preprocessing scenarios. Lastly, we present a CEP-based solution that supports the design and implementation of preprocessing and allows the execution of data mining algorithms over the resulting data streams.

3.1 | Principles and formal definition

On the basis of the particularities of data streams and current approaches for their preprocessing, we have identified the following desirable properties for stream-oriented preprocessing methods:

1. A reactive nature to be able to capture and transform incoming raw data as they arrive.
2. Native functionalities to establish temporal conditions and incorporate short-term information in data preprocessing.
3. Availability of frequently applied statistics and logical operations, as well as support to include user-defined ones.
4. A human-readable language to allow domain experts to define preprocessing processes, at the right level of abstraction.
5. A fast-processing engine easy to integrate with online data mining systems.

Based on these premises, our proposed method extends the scope of ECA rules and adapts CEP as the base system to perform data stream preprocessing. Formally, given a set of events $\{E_1,\ldots,E_p\}$, and a condition C about them (i.e., a Boolean expression that must be fulfilled by the events for the rule to be triggered), and a set of actions $\{A_1,\ldots,A_q\}$, a preprocessing rule (R) is an expression of the form:

$$R:\{E_1,\ldots,E_p\} \rightsquigarrow \{A_1,\ldots,A_q\},\ \text{if}\ \{E_1,\ldots,E_p\} \vDash C \tag{1}$$

It can be read as follows: if there is an occurrence of events $\{E_1,\ldots,E_p\}$ and they fulfil the condition C, then the set of actions $\{A_1,\ldots,A_q\}$ is triggered. The simplest rule consists on the occurrence of a single event, and a single action (e.g., the generation of a new instance), and this is why we will write them as $R: E \rightsquigarrow A$, if $E \vDash C$.

Here, an event corresponds to raw data to be preprocessed before applying an online data mining algorithm. Each event presents a set of attributes $\{a_1,\ldots,a_n\}$, formally, $E(a_1,\ldots,a_n)$. The condition C refers to a set of constraints $\{c_1,\ldots,c_k\}$ on the input events that should be considered during preprocessing. They can refer to attribute values or temporal restrictions, for example, windows (w) or the partial order between events in the streams. Windows can be either spatial, of size s, or temporal, valid for a time period t. We will note $E \vDash C$ when the attributes of event E satisfy the constraints defined by C. When no constraints are needed, the rule will be triggered for every incoming event. In the most common scenario, the rule action produces instances I comprised of features $\{f_1,\ldots,f_m\}$ ready to learn from. Following our notation, an instance is therefore defined as $I(f_1,\ldots,f_m)$. Later we will also describe a scenario of an action that is not intended to generate instances.

It should be noted that preprocessing rules can be inspired by the same principles guiding current preprocessing algorithms, but properly adapted to the language syntax and specifically oriented to the application domain. Moreover, window handling facilities open up new possibilities for improving existing data preprocessing procedures and for defining new ones. With the use of specific operators and a human-oriented syntax, rule-based preprocessing languages share the characteristics of domain specific languages (DSLs). It has been demonstrated that DSLs are easier to understand and to use than general purpose languages (GPLs), with improvements up to 15% on how users learn, perceive and evolve programmes developed with DSLs when compared to GPLs (Kosar et al. 2009; Kosar et al., 2010). This fact is even more significant in the case of the use of rule-based languages to represent the queries, where improvements have also been observed in terms of usability, effectiveness and efficiency by users with respect to general purpose languages (Groth, 2004).

Next, we present four scenarios where preprocessing rules are applicable, detailing the goal pursued, the actions to be performed and suitable operators. These scenarios cover all preprocessing tasks described in Table 1, with the exception of data integration, which would require an extension of our definition to support heterogeneous flows.

3.2 | Types of preprocessing rules

Each preprocessing task described in Section 2.2 has its own objective, but some of them share the way the instance is manipulated. For instance, both normalization and discretization tasks take an existing numerical attribute and change its value. The difference lies in the conditions that guide such modification and the operations used to perform it. Based on this assumption, we have identified four types of preprocessing rules, which are described next.

3.2.1 | Filter rules

A filter rule aims at identifying events meeting some criteria, expressed as conditions. Events that do not satisfy the constraints are simply discarded. Equation (2) provides a formal definition. In this case, one instance is created from each event that satisfies the k constrains specified in the rule. Therefore, the rule directly maps each event attribute to the instance feature, that is, $f_i = a_i$ and $n = m$. Selection and logical operators are the natural choices to set constraints over the event content.

$$F : E(a_1,\dots,a_n) \rightsquigarrow I(f_1,\dots,f_n),\ \text{if } E \vDash C \tag{2}$$

This type of rule is applicable to both data preparation and data reduction tasks. First, a filter rule is useful to identify events without missing values or noise. Second, filter rules enable instance selection, just by setting the condition that should be used to separate instances, for example, a categorical attribute.

3.2.2 | Transformation rules

The purpose of a transformation rule is to apply some change to the event, either in terms of structure or content. Therefore, the objective of the rule is to create or delete features, modify the attribute value, or change its type prior to transferring the value. A formal definition is given in Equation 3. Each feature is the result of an operation (*op*) over one or more attributes. Optionally, the operation can be defined within a window, thus returning a value computed by aggregating the attribute values from a sequence of past events. For this type of rule, projection and arithmetic operators provide the necessary functionality to select attributes and compute statistics, respectively. In addition, sequence operators might be needed to select particular events within the window W.

$$T :\ \{E_1,\dots,E_p\} \rightsquigarrow I(f_1 = op_1(E_1,\dots,E_p,W),\dots,f_m = op_m(E_1,\dots,E_p,W)) \tag{3}$$

Transformation rules are applicable in a broad range of preprocessing tasks. Focusing on data preparation, data type conversion and format manipulation are common procedures that can be encoded using operators in the rule action. Similarly, normalization also applies a numerical transformation, although it requires additional parameters to scale the value. Given that these parameters are subject to changes in the data distribution process, this task represents a good example of the need of window-defined operators to dynamically update them. Regarding data reduction, discretization also involves an operation over the raw attribute data, which should be replaced by a discrete interval. Finally, operations based on aggregation functions serve to generate new features with temporal information.

3.2.3 | Configuration rules

Due to the dynamic nature of stream data, preprocessing tasks might need some adjustments to cope with new conditions. Additional rules could be defined to control the parameters that influence preprocessing rules, or even activate and deactivate them. Therefore, these configuration rules do not react to events or produce instances. Instead, conditions are based on temporal factors and actions refer to the preprocessing process itself. Sequence operators and window facilities are key elements for the creation of configuration rules.

We conceive two situations in which configuration rules are useful. First, any transformation operation based on past statistics could need resetting such statistics after some time in order to obtain a more accurate estimation of the current flow. Second, configuration rules provide a simple mechanism to detect class imbalance, just counting how many instances of each class have been preprocessed. In the presence of imbalance, a rule used to generate instances should be triggered to produce artificial instances or to duplicate previous events belonging to the minority class.

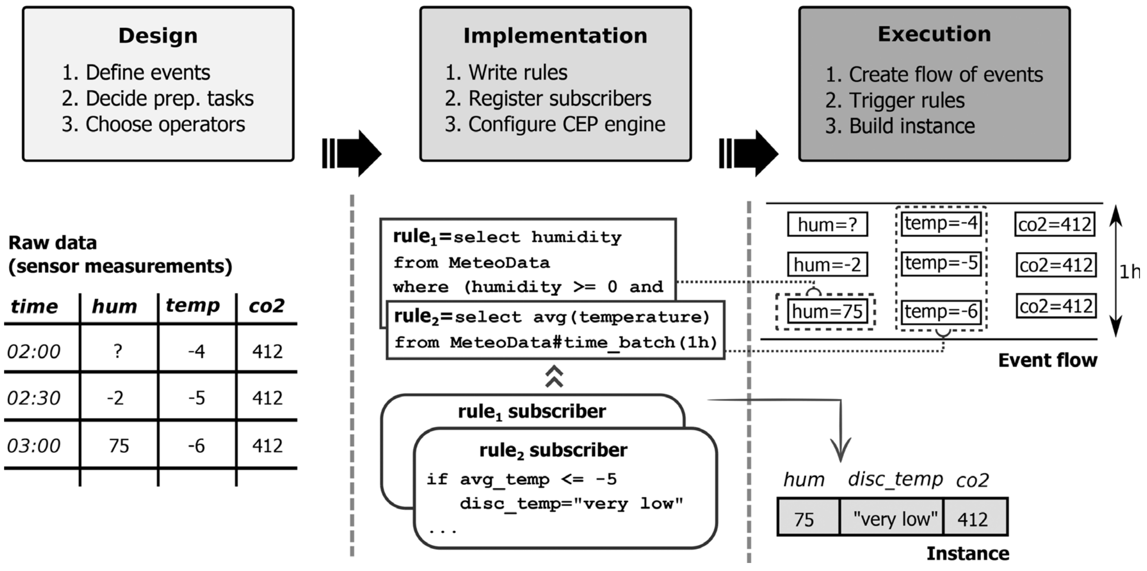
3.2.4 | Combined rules

An additional type of rule covers those preprocessing tasks that need to perform different transformations depending on the incoming data. In these cases, rules should include both detection and transformation capabilities. Formally, these rules are transformation rules (see Equation 3) that support the definition of constraints as defined for filter rules (see Equation 2). Therefore, combined rules unite the potential of both types of rules, applying projection and arithmetic operators over events filtered out by logical and selection operators.

Combined rules are useful for cleaning procedures that seek to curate data with missing values or noise. This way, imputation methods will only affect those events that present such issues. In addition, this type of rule is useful to apply multiple transformations in parallel, allowing the definition of different treatments for groups of instances sharing some characteristics, for example, the same class value.

3.3 | A CEP-based solution to support rule-based preprocessing

Rule-based preprocessing is a general approach that could be implemented in different ways. This section presents our proposed solution based on CEP, explaining how the previous concepts are applied in a preprocessing workflow adapted to data streams. We decided to build it on top of Esper, a CEP engine that provides native support for stream processing and a high-level SQL-like language for rule definition. Esper is a CEP implementation written in Java that fulfils the requirements established in Section 3.1 and can be easily integrated with MOA, a reference tool for data stream mining. Figure 1 shows an overview of the process, which consists on three phases: design, implementation and execution. For each phase, we describe its inputs, outputs and internal steps accompanied by a dummy example. In our example, a meteorologist wants to apply a learning algorithm to predict the air quality based on atmospheric indicators, such as humidity, temperature or CO_2 levels.



time	hum	temp	co2
02:00	?	-4	412
02:30	-2	-5	412
03:00	75	-6	412

```
rule1=select humidity
from MeteoData
where (humidity >= 0 and
rule2=select avg(temperature)
from MeteoData#time_batch(1h)
```

```
if avg_temp <= -5
   disc_temp="very low"
...
```

hum	disc_temp	co2
75	"very low"	412

FIGURE 1 Workflow of rule-based preprocessing using complex event processing

The first phase is focused on the identification and design of the preprocessing tasks needed in a particular domain. As input of this phase, a sequence of raw data is expected. In our example, such inputs correspond to the measurements collected at regular intervals by a meteorological station. Each sample in the stream represents an event in CEP, whose attributes contain the values of each atmospheric variable. Once the event has been defined, the domain expert should decide the type of preprocessing task to be applied. For instance, noise removal and discretization could be needed to clean climate data and make it more manageable for learning. These decisions will determine the type of preprocessing rule to be defined. Filter rules are useful for noise removal, whereas discretization requires a transformation rule (see Section 3.2). Next, the domain expert has to specify the conditions under which each rule should be triggered. For the case of noise removal, humidity can vary between 0 and 100, as it is expressed as a percentage according to the sensor instructions. As for discretization, he/she defines a number of intervals and their range of values, for example, temperature continuous values could be discretized into five levels: "very low" (-5° or below), "low" (between -4° and 10°), "normal" (between 11° and 25°), "high" (between 26° and 40°) and "very high" (more than 41°). Depending on the type of rule, the domain expert should make additional decisions, such as choosing the arithmetic operators to compute statistics in transformation rules and the definition of temporal or spatial window. In our example, the meteorologist decides that, instead of discretizing every single temperature value, it is more efficient to apply it over the averaged value measured during one hour. As outputs of the design phase, the expert has the set of events (*E*), actions (*A*), constraints (*C*) and operators (*op*) adapted to his/her application domain. The Appendix provides a reference guide of clauses and operators relevant to each data preprocessing task.

In the second phase, events and preprocessing rules are implemented using the design information from the previous phase. Also, a CEP engine should be configured to receive events and register the rules. In the case of Esper, the CEP implementation used in this work, events are actually Java objects that will be sent to a flow. Therefore, each type of event *E* is declared as a Java class, whose attributes are the variables associated to that event. In our example, a Java class "MeteoData" is created with three attributes: humidity, temperature and CO2 level. In Esper, rules are defined using an SQL-like language and the CEP engine uses a publish-subscribe service. Algorithm 1 specifies how this service is created and configured.[3] Firstly, rules are defined (lines 2–5) as Esper statements, which receive the query that represents the rule in EPL syntax as parameter. In our example, the first rule establishes the valid range for humidity values as a way to filter noise. The second rule applies the average operator over the temperature variable within a time window (1 h). Next, subscribers are created and associated to each statement (lines 8–11). A subscriber is a class that overrides an update method whose parameters should match with the output(s) of one rule. For instance, the subscriber of the discretization rule will receive the average temperature computed by the rule, and will assign the discrete value depending on that result. Once rules and subscribers are defined, the CEP service can be created and initialized (lines 13–17). This process consists in setting the type of event to be processed (a reference to the Java class) and registering the rules. After that, the CEP service is ready to receive incoming events from a flow.

The last phase corresponds to the execution of preprocessing rules, that is, the registered rules will react to the incoming events and produce instances from which a ML algorithm can make predictions. The first step in this phase consists of creating a flow of events, that is, Java objects in the case of Esper. Those events injected into the flow that satisfy the constraints will trigger the rules, that is, they will filter or transform the

Algorithm 1: Configuration of rules and subscribers in the CEP engine (Esper implementation)

```
// Define rules
String noiseStatement = "select humidity from MeteoData where (humidity >= 0 and humidity <= 100)"
EPStatement noiseRule = EPAdministrator.createEPL(noiseStatement).
String discretizeStatement = "select avg(temperature) from MeteoData#time_batch(1h)"
EPStatement discretizeRule = EPAdministrator.createEPL(discretizeStatement)
...
// Define subscribers
NoiseRuleSubscriber noiseSubscriber = new NoiseRuleSubscriber().
noiseRule.setSubscriber(noiseSubscriber).
DiscretizeRuleSubscriber discretizeSubscriber = new DiscretizeRuleSubscriber().
discretizeRule.setSubscriber(discretizeSubscriber)
...
// Create the CEP service
EPServiceProvider service = EPServiceProviderManager.createService()
service.initialize()
service.addEvent("event.MeteoData")
service.registerRule(noiseRule)
service.registerRule(discretizationRule)
...
```

matched events as represented in Figure 1. If a rule is triggered, its associated subscriber will collect and process the rule output. The subscriber is responsible of building the instance by combining the original event attributes with the outputs of the rule. Following with the meteorological example, the humidity values of the events will be analysed by the filtering rule, whereas the temperature values will be processed by the discretization rule. The humidity value within a valid range, the discretized temperature and the rest of atmospheric variables will be used to create an instance from the last event. Since our proposal is oriented towards real-time data mining, instances are directly sent to the learning algorithm. As mentioned above, MOA is the library that provides the algorithms and utilities required during the data mining phase. Currently available algorithms in our MOA wrapper implement supervised approaches: regression by gradient descent and six classification methods (decision tree, kNN, naive Bayes, a rule-based classifier and two ensemble methods). In our example, the air quality level could be predicted by a classification method using the preprocessed atmospheric variables as instance features. Finally, it should be noted that the MOA wrapper can be easily extended to invoke other algorithms and utilities from MOA, including drift detectors. For experimental purposes, our implementation also allows saving and loading instances in ARFF format.[4]

As explained above, event and rule definition depend on the application domain and therefore should be programmed by the domain specialist. To assist in this process and to evaluate the benefits of rule-based preprocessing, the following section presents three illustrative examples that show how our approach has been used in practice.

4 | EXPERIMENTAL EVALUATION

The proposed experiments cover three different preprocessing tasks: feature generation (Section 4.2), data transformation (Section 4.3) and missing value management (Section 4.4). These tasks allow us to apply all types of preprocessing rules, as well as exploit temporal features in CEP. The methodology followed to conduct the experimentation is detailed first.

4.1 | Methodology

Input data streams are generated from ARFF datasets publicly available in two well-known repositories: UCI[5] and OpenML.[6] Some of these datasets often appear in stream data mining studies (Ghomeshi et al., 2019; Nguyen et al., 2015; Ramírez-Gallego et al., 2017; Webb et al., 2018), so they are representative examples of flows of temporal information that should be processed in real time. The datasets are loaded instance by instance, thus simulating the arrival of simple events to CEP.

A *test-then-train* approach is followed in the learning phase. Each new instance is firstly used to test the current decision model, and then used for training (Bifet, Holmes, Kirkby, & Pfahringer, 2010). The decision models are evaluated from the usual perspectives in real-time data mining: prediction performance, time and memory. When applicable, other aspects of the models are discussed.

Four algorithms have been selected for comparative purposes among those available in MOA: *Hoeffding tree* (Hulten et al., 2001), which dynamically induces a decision tree; *k-nearest neighbours* (Nguyen et al., 2015), which classifies according to the k most similar instances; *naive Bayes* (Nguyen et al., 2015), which predicts the class probability of an instance based on the Bayes theorem; a *rule-based classifier* (Gama & Kosina, 2011), which derives a set of *if* → *then* classification rules; and two ensemble methods with Hoeffding trees as base classifiers, one adopting a bagging strategy (*leveraging bag*; Bifet, Holmes, & Pfahringer, 2010) and other including diversity mechanisms and a drift detector (*adaptive random forest*; Gomes et al., 2017). These algorithms follow different learning strategies and have been previously used in the literature (Prasad & Agarwal, 2016; Ramírez-Gallego et al., 2018).

To evaluate their performance, we use the window-based evaluator available in MOA with default window size (1000). The following measures are computed: accuracy, that is, percentage of correctly (positive or negative) classified instances; precision, which represents the percentage of true positive instances among those classified as positive; recall, which returns the percentage of correctly classified instances among all the positive ones; and F_1, the harmonic mean between precision and recall. The three last measures are computed per class and then reported on average. They all are obtained from the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) as follows:

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \tag{4}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} \tag{5}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{6}$$

$$F_1 = 2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7}$$

Algorithms are executed with and without applying CEP rules, so that the influence of preprocessing decisions, for example, operators and window size, can be studied. Kruskal–Wallis and Wilcoxon tests ($\alpha = 0.05$) are applied to statistically analyse the results. For pairwise comparisons, *p*-values are adjusted using the Holm method. As we are interested in analysing the relative performance improvement, all algorithms are executed with default parameters (see the MOA documentation). We consider this would be a common usage scenario for non-experts in data mining. All experiments were run in a Debian 8 computer with 8 cores Intel Core i7-2600 CPU at 3.40 GHz and 16 GB RAM. To reduce any possible bias due to machine overload, each experiment was repeated 10 times to measure time and memory. Since the two ensemble methods are randomized, a different random seed is configured in each repetition and averaged results are reported. Code, datasets, extended results and statistical validation are available for reproducibility purposes.[7]

4.2 | Experiment 1: Generating temporal features

4.2.1 | Experiment description

Predicting service demand require analysing users' consumption habits and price trends. As a practical example, a dataset containing the electricity price and demand of two Australian states – New South Wales and Victoria – is studied in this experiment. The dataset[8] is comprised of 45,312 instances sampled every 30 min. Among the nine features, the dataset contains four numerical features that indicate the price and demand of electricity in each state. The goal is to predict whether the price will rise or drop, so the problem is tackled from a binary classification perspective.

The rationale behind this experiment is twofold. Adding temporal statistics might be relevant to detect the conditions under which price and demand change. In addition, it is necessary to experimentally determine the amount of past information, that is, window size that leads to the best performance.

4.2.2 | Definition of preprocessing rules

For this experiment, a transformation rule is defined to include new features representing the short-term history. More specifically, we will use transformation operations over four attributes: electricity price and demand in New South Wales (nswprice and nswdemand), and equivalent attributes for Victoria state (vicprice and vicdemand). To study the behaviour of different aggregation functions, two preprocessing rules are proposed. Firstly, four derived features are obtained as the average value of the aforementioned attributes using a spatial window of size *s*. Formally, the transformation rule is expressed as shown in Listing 2. Analogously, the second rule obtains the minimum and maximum values for the four attributes. Consequently, eight derived features are created.

Listing 2: Transformation rule that adds average price and demand as features

```
E₁(date,day,nswprice,nswdemand,vicprice,vicdemand,transfer,class)…
Eₛ(date,day,nswprice,nswdemand,vicprice,vicdemand,transfer,class) ⇝
I(date,day,nswprice,nswdemand,vicprice,vicdemand,transfer,class,
  avgnswprice = avg(nswprice,w(s)),avgnswdemand = avg(nswdemand,w(s)),
  avgvicprice = avg(vicprice,w(s)),avgvicdemand = avg(vicdemand,w(s)))
```

Listing 3: The transformation rule defined in Listing 2 written in EPL syntax (Esper) 3

```
select *, avg(nswprice), avg(nswdemand), avg(vicprice), avg(vicdemand)
from Electricity#length(size)
```

Using Esper syntax, the first rule applies the avg operator to compute the average value of each of these attributes within a sliding spatial window (see Listing 3). The rule also selects the whole event (*) to keep its original information. For the second rule, min and max are the appropriate operators.

We planned two different scenarios to analyse the influence of the derived features. First, the instance is enriched with the temporal features. Second, the original price and demand attributes are replaced by the temporal features. Therefore, the CEP engine actually produces four streams: enriched with average values, reduced with average values, enriched with minimum and maximum values, and reduced with minimum and maximum values. Five window sizes are considered: 10, 50, 100, 500 and 1000.

4.2.3 | Results

Table 3 presents the results of the four algorithms in terms of accuracy and F_1. Symbols (+) and (−) stand for enriched and reduced configurations, respectively. The following acronyms are used for the algorithms: HT (Hoeffding tree), kNN (k-nearest neighbours), NB (naive Bayes), RC (rule-based classifier), LB (leveraging bag) and ARF (adaptive random forest).

A first interesting finding is that temporal features contribute to improving the performance of most of the algorithms, but only when original features are kept too. This suggests that the current price and demand are relevant for the learning process and should not be omitted. Even so, historical information is highly valuable to complement the decision models, revealing that changes in electricity prices might also be explained by recent consumption habits. Indeed, 30 out of the 60 combinations of algorithm and enriched data stream produce better classification results than using the default data stream.

The suitability of temporal features is specially evident when applying HT, NB and ARF, since the majority of the combinations of aggregation function and window size guarantee better predictions. This fact is a noteworthy achievement considering that ARF already was the second best algorithm among the six tested. Three algorithms with high initial performance (LB, ARF and HT) reach more than 95% accuracy and F_1 under the same preprocessing configuration: average(+) with window size equal to 50. In contrast, no improvement is observed for kNN, a method that

TABLE 3 Accuracy and F_1 results for electricity price change prediction. Figures expressed as percentage, higher values being preferred

Window Size		Accuracy						F_1					
		HT	kNN	NB	RC	LB	ARF	HT	kNN	NB	RC	LB	ARF
Original		*81.60*	***82.50***	*75.30*	*73.70*	*92.80*	*92.20*	*81.72*	***82.36***	*73.30*	*72.00*	*92.87*	*92.31*
Average (−)	10	68.50	76.00	60.80	54.50	77.49	83.26	68.44	75.78	58.33	51.64	77.23	88.06
	50	67.20	73.10	54.70	59.80	75.66	82.20	67.35	72.98	55.11	57.53	75.36	81.85
	100	71.10	72.40	58.00	66.00	75.46	82.27	71.14	72.28	59.09	64.67	75.10	81.90
	500	68.10	71.30	51.60	57.70	75.68	82.34	68.21	71.23	51.65	56.38	75.30	82.00
	1000	67.20	72.50	50.00	57.40	74.19	82.47	66.43	72.32	49.60	55.57	73.86	82.11
Average (+)	10	80.80	79.90	71.80	71.50	92.87	91.70	80.49	79.88	69.90	70.55	92.90	91.77
	50	**98.00**	80.20	75.70	**84.80**	**97.98**	**96.46**	**97.98**	80.05	74.13	**85.24**	**97.98**	**96.48**
	100	88.90	80.20	76.90	82.70	92.67	94.66	88.62	79.98	75.43	83.00	92.72	94.73
	500	78.50	78.70	75.60	71.70	91.93	92.50	78.31	78.69	74.03	72.67	91.93	92.61
	1000	84.10	80.10	75.40	63.10	92.29	92.19	83.84	79.91	73.81	60.84	92.29	92.23
Min/Max (−)	10	74.40	74.40	62.10	62.90	80.13	83.74	74.67	74.14	59.97	60.44	79.62	83.38
	50	72.20	72.10	58.50	55.90	74.14	81.43	72.52	72.06	59.69	53.87	73.94	81.16
	100	64.40	71.90	59.80	59.30	73.49	80.88	64.67	72.06	61.29	58.16	73.45	80.65
	500	66.40	71.60	50.80	61.30	75.06	82.31	65.75	71.53	50.78	58.95	74.69	82.01
	1000	64.10	71.40	47.90	59.10	75.67	81.61	62.82	71.27	47.33	57.42	75.46	81.25
Min/Max (+)	10	80.90	80.50	70.70	82.50	92.60	91.82	80.86	80.43	68.79	81.74	92.62	91.83
	50	91.00	78.40	76.60	65.70	94.19	93.90	91.00	78.40	75.11	63.28	94.27	93.98
	100	88.70	78.40	**77.10**	78.40	91.56	93.22	88.91	78.46	**75.65**	77.93	91.71	93.34
	500	85.40	77.10	75.60	72.80	90.84	92.38	85.96	77.09	74.03	72.81	91.01	92.44
	1000	82.90	76.50	75.40	77.30	89.89	92.59	82.90	76.46	73.81	76.61	90.09	92.64

Note: Italics stands for the baseline value. Bold + underline stands for the best value.

classifies according to the *k* most similar instances. Given that instances closer in time present similar values for the aggregated functions, kNN tend to assign the same class to all of them, not properly capturing the price change due to other features. This issue has a greater impact when the original features are discarded.

In terms of accuracy, the percentage of improvement depends on the applied algorithm: between 1.59 and 20.10% for Hoeffding tree, between 0.13% and 2.39% for naive Bayes, between 4.88 and 15.06% for the rule-based classifier, between 0.08 and 5.58% for leverage bagging and between 0.20 and 4.62% for adaptive random forest. Taking all preprocessed data streams as reference, HT, kNN, LB and ARF outperform NB and RC according to Kruskal–Wallis and two-tailed Wilcoxon tests. As for F_1 measure, most of the algorithms provide a good trade-off between precision and recall. The low values achieved by naive Bayes are due to a marked difference in the recall values for each class. Although this factor is also observed when the original data stream is used, the rate of false positives increases for the configurations with reduced features.

Focusing on preprocessing decisions, the average price and demand seem to be more informative than the minimum and maximum values. Recommended values for the window size are 50 and 100 in both cases, depending on the preferred algorithm. These facts suggest that electricity prices often present fluctuations that are better captured by the average function.

Differences in execution times can be mostly attributed to the selected algorithm, CEP preprocessing only requiring between 0.15 and 0.4 s to process all instances. Almost no difference is observed when the window size is increased, with the exception of the enriched stream with minimum and maximum values. Using kNN increases learning time up to 80 s, since more features necessarily imply more time to measure the distance between instances. For the rest of algorithms, learning time tends to be superior with the preprocessed data streams too, but the achieved performance improvement compensates this fact, specially when using a fast algorithm like Hoeffding tree. Both this algorithm and naive Bayes are quite stable and always require less than 1 s to conclude. Ensemble methods are considerably more costly in time with either the original stream or those obtained after preprocessing (between 8 and 20 s). The rule-based classifier is the only algorithm able to reduce the learning time a few seconds with respect to the original data stream. Even so, other algorithms are faster and provide better classification performance. As for the memory, running the algorithms with the preprocessed data streams might require nearly double of the memory used with the original data stream, though it never exceeds 11 MB.

It is worth mentioning that some of the decision models derived from the preprocessed data streams are less complex than the original ones. For instance, the number of nodes of the decision tree has been reduced from 57 to 51, while simultaneously increasing accuracy nearly 12% (min/max $(+)$). Furthermore, the highest gain observed in the rule-based classifier (15.6%, avg $(+)$) corresponds to a model with 25 rules instead of the 36 rules originally required. The most discriminatory temporal features, that is, those most frequently appearing in the decision models, are the average and the maximum electricity prices in New South Wales. The simplicity of the specification of the preprocessing rules using a CEP language, in this case Esper, is also worth noting.

Finally, we visually analyse the positive influence of the added temporal features during the learning phase. Figure 2 shows how accuracy evolves as the data stream is processed. We selected the HT algorithm because it achieves a good trade-off between accuracy and execution time, and reported the highest accuracy (98%). The learning curve when no preprocessing is performed is shown too (dashed line). The addition of the average values of price and demand with a window size equal to 50 provides the best response. The fact that CEP rules operate on a window basis helps capturing changes in the data stream, allowing the algorithm to recover its predictive capability in less time.

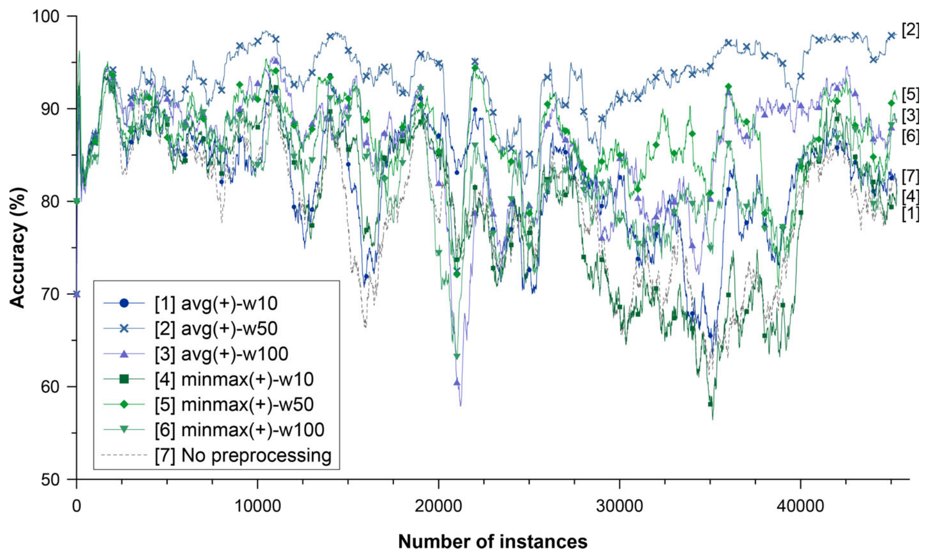


FIGURE 2 Learning curve of a Hoeffding tree for electricity data stream enriched with temporal features

4.3 | Experiment 2: Integrating categories

4.3.1 | Experiment description

Predicting flight delays and identifying their causes might alleviate inconveniences to travellers. To address this problem from a ML perspective, the airlines dataset[9] provides data from 539,383 real flight schedules. This dataset includes four categorical features: the departure and arrival airports, the airline and the day of the week. Airports can take up to 293 distinct values, whereas flights are operated by 18 airlines. The dataset also contains numerical features, including the elapsed time and the travelling distance. The class attribute indicates whether the flight was delayed (1) or not (0).

The presence of a high number of categories makes it difficult to generalize decision models. Indeed, a default execution of HT returns a decision tree with 8582 nodes and 8518 leaves, but only four levels of depth. This means that decisions are mostly made on a case-by-case basis, that is, first splitting by departure airport, then checking the airline and finally deciding depending on the arrival airport. Therefore, the purpose of this experiment is to study how categorical information can be indirectly considered in the learning process, in an attempt to not only improve classification performance, but also produce more manageable and understandable decision models.

4.3.2 | Definition of preprocessing rules

In this case, the objective of the preprocessing rule is to replace a categorical attribute by a numerical feature that keeps the information of each category regarding the presence of delays. Therefore, a combined rule is needed to detect those events that share the categorical value and then transform it. Since the event contains four categorical attributes, that is, airportTo, airportFrom, airline and dayOfWeek, four preprocessing rules are defined. Listing 4 shows an example considering that the departure airport is the attribute of interest. Note that the rule includes a constraint to specify the category and a spatial window, meaning that only the previous s flights whose departure airport is 'PHX' are used to count the number of delays. The day of the week is also retrieved, as this feature contains a small number of categories. Similar rules can be defined for the rest of attributes.

Listing 4: Example of a combined rule that replaces the categorical attribute airportTo by a numerical feature that represents the number of delays in the specified departure airport

E_1`(dayOfWeek,time,length,airportTo,airportFrom,airline,delay)` …
E_s`(dayOfWeek,time,length,airportTo,airportFrom,airline,delay)` $\rightsquigarrow$.
`I(dayOfWeek,time,length,delaysInAirportTo = sum(delay,w(s)),delay) if E.airportTo ='PHX'`

Listing 5 shows how the previous preprocessing rule is expressed in Esper. We apply the group by clause to split the flow by the categories of the attribute of interest, thus avoiding the need of registering one rule for each value. When the day of week is used in the group by clause, the attribute is removed from the select clause. We include the sum function to count the number of delays within the defined sliding spatial window, but only considering those events that share the same category of the feature appearing in the group by clause.

Listing 5: CEP rule that computes the number of flight delays by departure airport

```
select dayOfWeek, time, length, sum(delay), delay
from Flight#length(size)
group by airportTo
```

Using this syntax for the CEP rule, instances are grouped by the category of one feature only, thus allowing us to study the influence of each categorical attribute. We define another preprocessing rule to create instances enriched with four counters, one per categorical feature, so that we can analyse if their combination leads to better results or not. Although Esper allows to include more than one category in the group by clause, the resulting rule would consider those instances sharing all categorical values, which might not be a frequent case if the window size is small. In contrast, the rule shown in Listing 6 is able to produce an instance with four delay counters, one per categorical feature. It includes subqueries to compute each delay in an independent window, whose name is composed by the prefix ‘w’ and the name of the feature. The where clauses match the events in the subquery and the global query. The result of each subquery is renamed using the as operator, so that it is easily identified.

Listing 6: CEP rule that computes the number of flight delays for all attributes

```
select time, length, delay
(select sum(delay) from Flight#length(size) wAirportF
where wAirportF.airportFrom = wEvent.airportFrom) as dAirportF,
(select sum(delay) from Flight#length(size) wAirportT
where wAirportT.airportTo = wEvent.airportTo) as dAirportTo,
(select sum(delay) from Flight#length(size) wAirline
where wAirline.airline = wEvent.airline) as dAirline,
(select sum(delay) from Flight#length(size) wDayOfWeek
where wDayOfWeek.dayOfWeek = wEvent.dayOfWeek) as dDayOfWeek,
from Flight#length(size) wEvent
```

4.3.3 | Results

Table 4 shows the accuracy and F_1 values obtained for the five data streams produced by the CEP rules. As a baseline, the first row indicates the prediction results for the original data stream.

In view of the high number of shaded cells, relying on delay information of flights sharing certain characteristics leads to a substantial increase of classification performance in all algorithms. Considering either the arrival or the departure airport in a short-term window produces a significant improvement, achieving more than 80% of accuracy with five out of the six algorithms. Slightly better results are obtained with the departure airport (more than 90% of accuracy with all algorithms), suggesting that more delays are explained by the conditions at the origin.

In contrast, the day of the week seems to be less informative to predict delays. Accurate information is only gathered if a small number of previous flights is considered, and the achieved improvement – up to 37% – is relatively low compared to airport attributes – up to 102%. Focusing on the airline, improvements are similar (up to 34%). Furthermore, both accuracy and F_1 results are less dependent on the configured window size. It follows from these results that flights operated by a specific airline are systematically delayed or not, regardless of when and where previous delays occurred.

Even though some improvements are observed when grouping by one feature, combining the derived delay information from the four counters provides the best performance by far. With a window size equal to 10, all algorithms achieve an accuracy greater than 92% (and up to 99% for four algorithms), representing an average percentage of improvement of 66% with respect to the results obtained using the original data stream. F_1 values confirm that high precision and recall are obtained for both positive and negative samples. Ensemble methods, the rule-base classifier and Hoeffding tree stand out as the best methods, nearly reaching perfect classifications. Furthermore, gains are now more generalized across window sizes, kNN and NB being the sole algorithms for which improvement is not possible in two cases. The one-tailed Wilcoxon test confirms that RC, LB and ARF perform better when the four delay counters – with any window size – replace categorical features in the input data stream.

In terms of preprocessing time, CEP rules that group delays by one feature run in less than 2 s, not being affected by the window size. However, the rule producing the four counters requires between 3 and 30 s to conclude, depending on the window size. Additionally, some differences among algorithms are perceived. Hoeffding tree and naive Bayes perform the learning phase with the original data stream in around 5 and 4 s, respectively. Both algorithms become faster after transforming the data stream, requiring less than 3 s to conclude when one counter is applied. Note that the total execution time is less than that of the original stream even if preprocessing time is added. In contrast, the rule-based classifier, which initially required around 10 s, suffers from the presence of numerical attributes. Now it takes several minutes to compute the internal statistics[10] to decide which values increase the predictive performance the most. No great differences are observed for kNN, since categorical values are internally treated as numbers in MOA. Only a slightly increase is perceived for the configuration with four counters. kNN requires around 5 min to execute in all cases, including training with the original data stream. Lastly, ensemble methods (LB and ARF) significantly reduce the time needed for learning if preprocessing is enabled. These algorithms initially required more than 10 min to conclude, dropping to less than 3 min after using CEP preprocessing.

TABLE 4 Accuracy and F_1 results for flight delay prediction. Figures expressed as percentage, higher values being preferred

Window size		Accuracy						F_1					
		HT	kNN	NB	RC	LB	ARF	HT	kNN	NB	RC	LB	ARF
Original		*64.00*	*64.60*	*63.90*	*45.00*	*58.86*	*62.28*	*62.21*	*62.29*	*63.04*	*51.93*	*58.57*	*62.01*
Airport from	10	84.00	80.50	83.20	62.90	83.58	83.44	85.95	81.02	85.44	57.82	85.72	85.18
	50	65.20	64.40	61.60	56.50	68.23	68.63	69.47	62.46	62.50	52.62	71.40	70.40
	100	63.30	62.80	58.10	63.00	62.37	63.07	64.03	59.91	59.12	63.14	65.01	64.85
	500	55.40	60.70	49.80	58.50	59.87	56.55	57.41	57.49	51.70	52.59	61.51	59.00
	1000	61.10	62.10	50.10	52.80	57.91	56.44	59.86	58.48	52.63	57.46	59.66	58.62
Airport to	10	96.40	**<u>96.40</u>**	**<u>96.40</u>**	90.90	96.40	96.38	96.88	**<u>96.88</u>**	**<u>96.88</u>**	90.32	96.88	96.86
	50	87.60	86.70	84.00	65.40	87.59	87.44	89.25	87.81	84.87	60.81	89.24	88.99
	100	80.80	78.30	66.10	70.90	80.75	80.02	81.75	78.86	63.59	70.08	83.09	81.95
	500	60.90	61.60	57.40	59.30	62.59	63.72	63.56	59.25	54.54	62.43	65.98	66.16
	1000	57.40	59.00	54.70	55.00	58.87	58.97	58.07	55.80	52.29	56.59	61.56	61.59
Day of week	10	65.30	61.30	63.60	61.90	65.09	64.22	63.12	58.45	63.91	59.98	63.29	62.61
	50	60.00	57.20	56.60	54.90	60.10	58.38	58.27	53.38	58.20	57.13	57.62	56.60
	100	58.90	57.40	55.20	58.30	59.59	57.14	56.06	53.75	57.55	59.61	56.88	55.12
	500	58.40	59.00	46.20	53.90	60.29	57.69	57.52	54.60	52.69	57.24	58.23	56.30
	1000	54.20	58.40	42.30	52.70	59.55	57.55	55.33	53.86	50.00	57.15	58.45	56.89
Airline	10	78.90	72.70	62.20	57.70	78.92	78.94	81.68	72.43	56.58	50.00	81.72	81.40
	50	64.40	63.40	63.10	60.20	64.22	63.68	66.00	61.44	59.88	65.48	66.87	66.70
	100	63.00	62.30	60.60	52.90	62.61	63.10	64.34	59.48	56.87	59.03	64.56	64.54
	500	62.40	61.70	61.20	53.90	60.96	60.32	61.42	58.23	57.17	58.51	61.02	61.05
	1000	62.70	62.30	61.30	56.80	60.62	59.77	60.52	58.50	57.76	58.75	60.64	59.97
4 counters	10	**<u>99.20</u>**	96.30	92.30	**<u>99.40</u>**	**<u>99.40</u>**	**<u>99.40</u>**	**<u>99.31</u>**	96.79	93.33	**<u>99.48</u>**	**<u>99.48</u>**	**<u>99.48</u>**
	50	94.80	85.30	72.60	83.20	94.80	94.81	95.49	85.72	73.07	81.31	95.49	95.50
	100	89.50	75.10	65.50	85.90	89.75	89.66	90.74	73.85	65.40	86.14	91.10	90.95
	500	68.20	64.20	55.80	66.60	68.73	68.72	70.55	61.22	56.30	70.36	70.32	70.15
	1000	62.30	63.50	55.30	61.00	64.56	64.64	64.59	60.26	56.97	63.59	64.25	65.05

Note: Italics stands for the baseline value. Bold + underline stands for the best value.

Regarding memory requirements, it is worth mentioning that all algorithms consume less memory after preprocessing with the exception of RC. The most significant reductions correspond to the Hoeffding tree method, from nearly 16 to 0.4 MB (on average). The increase of memory of the rule-based classifier, from 3 to 8 MB, is attributed to the growth of the data structures due to a higher number of possible comparisons to build the rules. Despite this, the total memory consumption is still acceptable.

Differences regarding the characteristics of the decision models should be highlighted. The structure of the decision tree – four depth levels, 8582 nodes and 8518 leaves for the original data stream – has drastically changed. The most accurate tree after preprocessing presents 15 depth levels, 187 nodes and 94 leaves. The most frequently appearing delay counter is the day of week, followed by the airline, the departure airport and the arrival airport. As for the rule-based classifier, the original data stream led to a set of 182 rules, most of them only referring to the departure airport in their antecedent. When the four delay counters are considered, the number of rules decreased down to 50, and they combine information from all attributes. In fact, flight distance and duration, which did not appear neither in the decision tree nor in the rule set, are now frequently included. Consequently, experts are provided with less complex models, either in the form of a decision tree or a rule set, that include additional decision factors.

Finally, Figure 3 illustrates the evolution of the accuracy (measured in window intervals) to analyse the impact of changes in the stream. Again, we chose the Hoeffding tree as the learning algorithm, whose original performance is depicted with a dashed line. We can see how both the drop rate and the recovering time can be mitigated after preprocessing with CEP rules. In particular, grouping the flight delays by either departure or arrival airport allow reaching high accuracy rates quickly and, more importantly, maintain them above 70% afterwards. The good results obtained by applying the four counters simultaneously are visible from the beginning, and the accuracy remains fairly constant afterwards. Therefore, it is a robust configuration for online learning.

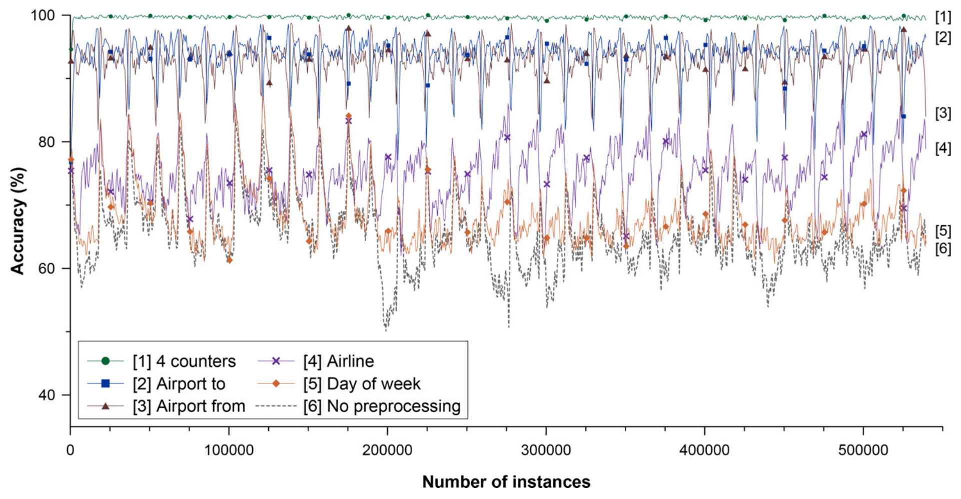


FIGURE 3 Learning curve of a Hoeffding tree for airlines data stream with delay counters (window size = 10)

4.4 | Experiment 3: Dealing with missing values

4.4.1 | Experiment description

The data streams produced by error-prone sensors are expected to contain missing values that should be properly handled. Alternatives vary from removing incomplete data to designing imputation methods to replace them. The purpose of this experiment is to illustrate how these procedures can be encoded as preprocessing rules.

This experiment considers a dataset that includes 20,560 measurements from light, temperature, humidity and CO_2 sensors,[11] from which room occupancy is detected (Candanedo & Feldheim, 2016). Missing values are inserted following two strategies: random distribution and periodical sequences of missing values. For the random strategy, a percentage (5%, 10%, 25%, 50%) of instances is replaced by missing values. For the sequences, both the frequency (100, 500) and the length (5, 10, 15) are configured.

4.4.2 | Definition of preprocessing rules

In this experiment, events that contain missing values should be treated differently from those that are complete. Such a situation allows us to show how different types of preprocessing rules can be simultaneously considered, but automatically triggered in different moments. Firstly, complete events just need to be detected and passed to the learning algorithm. For this, we define the filter rule shown in Listing 7, and translate it into CEP (Listing 8). Assuming that the sensor failure causes a lack of temperature measurement, the temperature attribute (`temp`) would be *null*. Alternatively, other attributes could be included in the `where` clause, or even the absence of the whole event could be detected with the `exists` operator.

Listing 7: Filter rule that captures complete sensor samples

```
E(temp, hum, light, CO2, humRatio, occupancy) ⇝
I(temp, hum, light, CO2, humRatio, occupancy)
if E.temp ≠ null
```

Listing 8: CEP rule to preprocess complete sensor samples 8

```
select *
from Sensor where (temp is not null);
```

If only the previous rule is registered in the CEP engine, instances with missing values will be simply discarded. In order to complete those instances, a combined rule must also be defined. To estimate the missing values, we implement the least squares method as an external function that adjusts the distribution of each attribute. As for the class label, which specifies whether the room is occupied (1) or not (0), we propose assigning the most frequent value within the window. Listings 9 and 10 show the preprocessing rule using our formal notation and Esper syntax, respectively. In addition, a configuration rule is periodically triggered to clean out-of-date samples, with the intention of building most accurate fitting curves.

Listing 9: Combined rule to replace missing values by estimations based on previous events

```
E(temp, hum, light, CO2, humRatio, occupancy) ⇝
I(temp = estimation(temp,w(s)), hum = estimation(hum,w(s)), light = estimation(light,w(s)),
CO2 = estimation(CO2,w(s)), humRatio = estimation(humRatio,w(s)),
occupancy = mode(occupancy w(s)))
if E.temp = null
```

Listing 10: CEP rule that generates an instance by estimating missing values

```
select Function.getEstimation(),
(select window(occupancy).mostFrequent()
from Sensor#length(size))
from Sensor where (temp is null);
```

Two additional imputation methods based on CEP operators are defined for comparative purposes. The first one replicates the last value received in the stream, for which the lastOf operator is applied (see Listing 11). As this operator is window-defined, the keepall keyword is included to specify that the window should contain all previous measurements. The second imputation strategy combines the structure of the two last rules. It computes the average value of each attribute and the class label is set to the most frequent value within a sliding spatial window. For the estimation and the average methods, we evaluate three window sizes: 30, 60 and 90.

Listing 11: Rule that generates an instance by replicating the last available measurement

```
select
(select window(temp).lastOf() from Sensor#keepall where (temp is not null)),
(select window(hum).lastOf() from Sensor#keepall where (hum is not null)),
(select window(light).lastOf() from Sensor#keepall where (light is not null)),
(select window(CO2).lastOf() from Sensor#keepall where (CO2 is not null)),
(select window(humRatio).lastOf() from Sensor#keepall where (humRatio is not null)),
(select window(occupancy).lastOf() from Sensor#keepall where (occupancy is not null))
from Sensor where (temp is null)
```

4.4.3 | Results

Tables 5 and 6 collect the results for data streams with randomly injected missing values (MV) and with missing sequences, respectively. The results for estimation and average methods correspond to the best window size (30). The first row in Table 5 provides the classification performance for the data stream without missing values. In both tables, we also include the accuracy and F_1 obtained when no missing value treatment is applied (referred to as "with MV"). It should be noted that, for this experiment, we report absolute values for both performance metrics, since the window-based approach might bias the results if no missing values appear in the window. As it can be observed, all algorithms experience a decrease in their prediction capabilities under the presence of missing values, specially if a high number of randomly distributed instances is

TABLE 5 Accuracy and F_1 results for occupancy prediction (missing values at random). Figures expressed as percentage, higher values being preferred

%		Accuracy						F_1					
MV		HT	kNN	NB	RC	LB	ARF	HT	kNN	NB	RC	LB	ARF
Original		*98.92*	*98.57*	*96.04*	*90.84*	*99.14*	*99.06*	*99.03*	*98.06*	*96.76*	*84.75*	*99.09*	*98.91*
With MV	5	95.23	96.69	95.06	86.75	95.47	95.38	96.57	96.51	94.54	83.91	96.69	96.54
	10	91.69	94.87	94.18	88.86	94.11	91.82	94.30	95.14	92.48	81.45	94.58	94.24
	25	81.94	90.23	91.72	85.40	94.78	91.96	88.00	91.47	86.90	79.99	91.67	93.36
	50	69.04	83.83	88.54	77.21	89.07	93.36	79.71	86.42	78.88	64.37	84.88	93.33
Remove	5	**98.94**	98.57	96.02	91.45	**99.17**	99.07	98.97	98.04	**96.77**	85.93	99.11	**98.91**
	10	98.48	98.59	96.00	87.96	99.15	99.04	98.73	98.09	96.74	81.83	99.06	98.88
	25	98.16	98.44	95.71	86.71	99.10	99.00	98.43	97.94	96.52	79.89	99.00	98.88
	50	98.82	98.51	**96.14**	89.34	98.92	98.92	98.86	98.19	**96.77**	89.63	98.98	98.72
Estimate	5	98.25	98.51	95.72	87.97	99.08	98.98	98.45	97.99	96.43	78.44	99.02	98.79
	10	98.10	98.49	95.57	87.55	98.96	98.93	98.22	97.99	96.34	84.01	98.85	98.75
	25	98.37	98.24	95.00	91.67	98.66	98.65	98.21	97.61	95.79	85.93	98.51	98.37
	50	98.06	98.30	94.40	89.75	98.43	98.49	97.63	97.69	95.10	82.10	98.21	98.14
Average	5	98.91	98.59	96.01	92.72	99.14	99.07	98.95	98.09	**96.77**	**93.95**	**99.15**	98.89
	10	98.89	**98.61**	96.02	91.33	99.05	99.07	98.92	98.10	96.74	91.52	99.01	98.94
	25	98.74	98.59	95.88	**93.72**	99.05	99.02	98.55	98.07	96.65	93.27	98.95	98.86
	50	98.63	98.74	95.80	93.57	99.04	99.06	98.62	**98.29**	96.49	92.98	98.96	98.90
Last	5	98.93	98.59	96.04	90.84	99.13	**99.10**	**99.04**	98.10	96.75	84.75	99.10	98.94
	10	98.92	98.54	96.04	90.36	99.12	99.06	99.03	98.03	96.75	84.01	99.05	98.87
	25	98.88	98.52	95.78	90.85	99.11	99.06	98.91	98.04	96.54	89.87	99.03	98.90
	50	98.86	98.46	95.99	90.85	99.10	99.00	98.87	97.97	96.61	84.77	98.97	98.77

Note: Italics stands for the baseline value. Bold + underline stands for the best value.

missing. These reference configurations allow us to identify the room for improvement of each combination of data stream and algorithm, as well as to analyse whether the imputation method is able to recover the original prediction performance.

According to the one-tailed Wilcoxon test, all algorithms perform better when a curated data stream is received, independently of the imputation method applied. RC slightly decreases in 5 out of the 40 data streams, though the difference in performance is less than 6% lower than the expected value of accuracy or F_1. Another relevant finding is that classification performance could even be improved with respect to that of the complete data stream. There are two possible reasons explaining this deceptive outcome: the missing instances correspond to false positives or false negatives that are removed, or the imputation method has inverted the class label assigned to such instances. Since the class attribute represents room occupancy, it is expected that positive values appear in sequence, then turning into a sequence of false values, and vice versa. Therefore, the use of the last or the most frequent value within a window could miss the turning point.

Focusing on Table 5, a simple removal of missing values seems to work well. Accuracy returns to values higher than 95% for HT, kNN and NB algorithms. However, estimations based on the average and last values help the rule-based classifier to obtain more robust results, higher than 90%. Ensemble methods (LB and ARF) show similar results, with accuracy values above 98% for all kinds of preprocessing strategies. As it can be expected, imputation methods are more effective as less missing values have to be estimated. Even so, their performance when there is a high percentage of missing values is noteworthy, achieving accuracy percentages only 1% inferior to the ones obtained with the complete data stream. Since samples are taken every minute, consecutive instances in the data stream present similar sensor measurements, so the values returned by window-based estimation methods are quite close to the original ones.

When missing instances appear in sequences, there is no great difference in the final accuracy achieved by the methods (see "with MV" in Table 6). Although all algorithms have time to recover after receiving a missing sequence, a high rate of such sequences, for example, 15 missing samples every 100 samples ({100,15}), degrade their accuracy up to 10% (e.g., from 98.92 to 86.82 for Hoeffding tree). kNN and ensemble methods are less affected in this sense. Again, removing missing values becomes a competitive solution, though accuracy rates closer to the original ones are obtained with imputation methods. In particular, the estimation by the least squares method is highly appropriate to predict next values within the sequence. Methods based on the average and last values are less effective here because they repeat previous measurements. This is reflected in more configurations exceeding the initial accuracy, since more instances share similar values.

TABLE 6 Accuracy and F_1 results for occupancy prediction (sequences of missing values). Figures expressed as percentage, higher values being preferred

		Accuracy						F_1					
	{frequency, length}	HT	kNN	NB	RC	LB	ARF	HT	kNN	NB	RC	LB	ARF
With MV	{100,5}	95.08	96.61	95.36	88.25	95.43	95.21	96.47	96.54	94.67	82.99	96.61	96.40
	{100,10}	90.82	94.48	94.39	84.23	95.45	91.41	93.73	94.80	92.39	78.71	95.19	93.96
	{100,15}	86.82	92.30	93.42	82.42	92.63	90.71	91.21	93.12	90.20	75.55	92.77	92.74
	{500,5}	98.19	98.22	95.82	89.26	98.40	98.34	98.54	97.73	96.18	81.46	98.61	98.43
	{500,10}	97.46	97.87	95.58	88.89	97.84	97.66	98.06	97.36	95.59	81.07	98.19	97.96
	{500,15}	96.72	97.52	95.36	89.13	98.20	96.97	97.55	97.02	95.02	81.44	97.87	97.45
Remove	{100,5}	98.23	98.63	96.33	92.52	99.12	99.04	98.45	98.14	96.92	88.10	99.05	98.89
	{100,10}	98.39	98.60	96.34	88.04	99.08	99.05	98.62	98.10	96.90	78.11	99.02	98.91
	{100,15}	98.14	98.59	96.34	92.78	99.03	99.05	98.53	98.11	96.97	91.17	98.98	98.84
	{500,5}	98.91	98.55	96.04	88.51	99.12	99.03	99.01	98.02	96.72	84.48	99.06	98.84
	{500,10}	98.90	98.55	96.03	89.98	99.10	99.04	98.99	98.02	96.67	83.65	99.06	98.86
	{500,15}	98.88	98.54	96.05	91.44	99.08	99.00	98.94	97.98	96.66	86.74	99.04	98.83
Estimate	{100,5}	98.19	98.57	96.27	88.21	98.99	99.07	98.33	98.09	96.45	83.74	99.00	98.90
	{100,10}	98.22	98.56	96.14	91.32	98.95	99.04	98.53	98.11	96.02	86.53	98.99	98.89
	{100,15}	98.16	98.61	96.17	**93.67**	98.89	99.03	98.51	**98.26**	96.16	92.08	98.98	98.87
	{500,5}	98.85	98.53	96.16	90.85	99.11	99.05	98.87	98.02	96.65	84.89	99.05	98.90
	{500,10}	98.81	98.54	96.21	90.58	99.13	99.04	98.94	98.03	96.54	84.71	99.06	98.86
	{500,15}	98.81	98.53	96.24	88.36	99.13	99.05	98.94	98.00	96.58	84.09	99.09	98.86
Average	{100,5}	**98.92**	**98.64**	96.30	91.71	99.11	99.04	98.92	98.17	96.90	91.41	99.02	98.90
	{100,10}	98.20	98.63	**96.44**	93.39	99.10	99.07	98.39	98.12	**97.04**	**94.39**	99.02	98.91
	{100,15}	98.16	98.63	96.41	87.82	99.06	99.04	98.41	98.14	97.03	85.96	99.01	98.89
	{500,5}	**98.92**	98.57	96.00	90.40	99.13	99.06	**99.04**	98.06	96.66	84.10	99.08	98.89
	{500,10}	98.90	98.53	95.98	88.53	99.14	99.06	99.00	97.98	96.61	83.32	**99.10**	98.88
	{500,15}	98.86	98.49	95.89	90.90	99.11	99.05	98.98	97.90	96.53	91.19	99.09	98.87
Last	{100,5}	**98.92**	98.57	96.37	90.83	99.14	99.05	99.02	98.08	96.97	84.73	99.08	98.88
	{100,10}	98.88	98.55	96.43	88.90	99.10	99.04	98.99	98.09	97.00	80.16	99.05	98.85
	{100,15}	98.86	98.56	96.37	88.14	99.08	98.97	98.96	98.10	96.99	79.92	99.06	98.79
	{500,5}	**98.92**	98.57	96.04	90.84	**99.15**	**99.08**	99.03	98.06	96.76	84.75	99.09	**98.92**
	{500,10}	**98.92**	98.57	96.04	90.84	99.14	99.05	99.03	98.06	96.76	84.75	99.09	98.89
	{500,15}	**98.92**	98.57	96.05	90.84	**99.15**	99.07	99.03	98.06	96.77	84.75	99.09	98.90

Note: Italics stands for the baseline value. Bold + underline stands for the best value.

Some differences arise regarding preprocessing time. Removal (between 0.06 and 0.1 s) and estimation (between 0.11 and 0.19 s) are the faster methods, followed by average (between 0.17 and 0.25 s) and last (up to 10 s) operators. The use of keepall in the last rule seems to be the reason behind such difference. Since this experiment does not change the number or type of features, learning time and memory requirements stay quite stable. Hoeffding tree and naive Bayes are the faster algorithms, with execution times inferior to 1 s. Only a slight decrease in time is observed when a high number of instances with missing values is removed. For the same reasons, the decision models obtained before and after preprocessing present similar structures and decision variables.

5 | COMPARISON AND DISCUSSION

After the experimental evaluation, our results should be contrasted with those obtained by other alternatives available to the domain expert. To do this, this section presents a comparison between our rule-based preprocessing approach and other preprocessing techniques included in MOA and MOAReduction. The results are compared in terms of execution time and classification performance achieved after applying different preprocessing algorithms. Then, a deeper analysis of the characteristics of each tool allows us to discuss their strengths and limitations.

5.1 | Comparison with other preprocessing techniques

To contextualize the results obtained by our rule-based preprocessing approach, this section presents a comparison against other preprocessing techniques. We focus on publicly available techniques coded in the same programming language than our CEP-based solution (Java), which ensures a fair comparison in terms of execution time. Firstly, we have considered the five discretization algorithms available in MOAReduction: incremental discretization algorithm (IDA; Webb, 2014), incremental flexible frequency discretization (IFFD) algorithm (Lu et al., 2006), LOFD (Ramírez-Gallego et al., 2018), partition incremental discretization (PiD) algorithm (Gama & Pinto, 2006), and Online ChiMerge (OC) algorithm (Lehtinen et al., 2012). These algorithms have been executed with default parameters for the electricity dataset in order to compare their performance against the best configuration of CEP rules reported in Section 4.2.3, that is, enriching the stream with average price and demand attributes using a window size equal to 50. These discretization algorithms constitute a representative sample of the state of the art, and have achieved good improvements with this dataset in the past (Ramírez-Gallego et al., 2017). Secondly, we have applied the MOA filter to replace missing values using the occupancy data streams with injected missing values as input (see Section 4.4.1). This MOA filter has two parameters: the type of replacement for numerical attributes (last, mean, max, min or constant value) and for nominal attributes (last or mode).[12] Combining both parameters, 10 different configurations are obtained for comparison. This filter is useful to discuss the results presented in Section 4.4.3, where we proposed four window-based strategies to impute missing values using similar operators. In both comparative studies, Hoeffding tree is chosen as the learning algorithm to be applied after preprocessing. This way, the observed differences in terms of accuracy and execution time can be attributed to the preprocessing step. We compute the same classification performance metrics described in Section 4.1, using a window-based approach for the electricity data stream and the accumulated values for the occupancy data stream.

Table 7 shows the results of the five discretization algorithms for the data stream used in our first experiment (electricity). The results of the HT algorithm without preprocessing and combined with the best rule-based preprocessing (avg($+$) with window size $=$ 50) are included for reference too. The last column in Table 7 represents the total execution time (preprocessing and learning) in seconds, reported as the average and standard deviation of 10 runs. Focusing on the classification performance, all discretization algorithms are able to increase the accuracy and F_1 values obtained when preprocessing is not applied. The improvements in accuracy range from 6% (OC) to 9% (IDA). However, the proposed rule-based preprocessing approach guarantees a considerable better improvement (20%) than the best discretizer (IDA algorithm). Furthermore, CEP preprocessing has a lower impact on the execution time than the five discretization algorithms, which can require nearly 1 min to conclude. This fact is remarkable considering that MOAReduction uses the same structures than MOA, and the discretization methods are embedded in the learning process.

Focusing on the occupancy data stream, Figure 4 shows the results of the different imputation methods in terms of accuracy. For each kind of MV injection (random or block), the first column (in grey) represents the accuracy of the Hoeffding tree algorithm without doing any preprocessing. As for the MOA filter, changing the option for nominal attributes (last or mode) does not report any difference in accuracy values, so we only keep the combinations applying the last operator because they are faster. We also rule out the use of the last operator for numerical attributes in the MOA filter, since it never produces an improvement in the learning phase. From the rest of combinations, none of the configurations of the MOA filter is able to recover from the loss of accuracy when only 5% of instances present missing values. In contrast, all CEP preprocessing strategies increase the accuracy values obtained by the learning algorithm. When more missing values appear either randomly or in blocks, the mean operator works better than the rest of options in MOA. Only for two data streams with injected MV (block-100-10 and block-

TABLE 7 Classification performance and execution time of discretization algorithms compared to our best results for electricity dataset

Preprocessing technique	Accuracy	F_1	Execution time (s)
No preprocessing	81.60	81.72	0.44 ± 0.08
IDA	89.00	89.06	56.69 ± 10.22
IFFD	88.20	88.25	54.98 ± 11.57
LOFD	88.80	88.88	57.60 ± 9.58
OC	86.50	86.33	55.87 ± 10.92
PID	87.60	87.53	54.04 ± 12.38
CEP rule (avg($+$)-w50)	98.00	97.98	0.89 ± 0.11

Abbreviations: CEP, complex event processing; IDA, incremental discretization algorithm; IFFD, incremental flexible frequency discretization; LOFD, local online fusion discretizer; OC, Online ChiMerge.

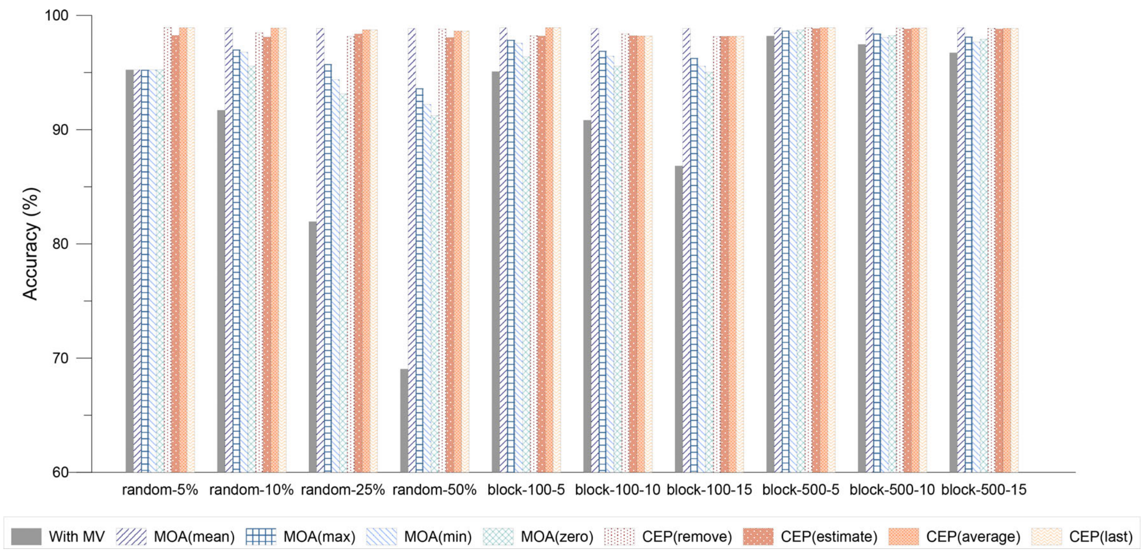


FIGURE 4 Percentage of accuracy achieved by Hoeffding tree after applying MOA filter and complex event processing preprocessing to the occupancy dataset

100-15), the MOA filter with mean replacement allows achieving higher accuracy than any of the CEP preprocessing strategies. However, results after CEP preprocessing are more stable, with values always greater than 98% with independence of the amount and distribution of MV. Any CEP operator guarantees better accuracy than the MOA filter with max, min or zero replacement, for which accuracy varies from 91% to 98% depending on the selected operator.

Finally, both preprocessing techniques can be compared in terms of execution time. Table 8 shows the average time (preprocessing and learning) of the 10 runs, as well as its standard deviation. Time increased by the MOA filter is quite constant and almost negligible, since learning from the data streams with injected MV took between 0.07 and 0.09 s. CEP preprocessing show more variation in this sense, although only the last operator requires more than 1 s to conclude. The difference between MOA filter with mean replacement and CEP preprocessing using the average operator – the two options more similar – can be explained by three factors. Firstly, CEP includes additional operations to convert events into samples, something not required in MOA. Secondly, the CEP engine is event-driven, triggering different rules depending on the incoming event. Doing this distinction is faster in MOA, since all samples are processed by the same class. Thirdly, the CEP operator is configured to use a sliding window, a mechanism not available in MOA at the moment. This last characteristic, together with the possibility of applying a different operators to each attribute, give more flexibility to CEP preprocessing at a minimum cost in execution time.

5.2 | Strengths and limitations of rule-based preprocessing

This section provides an overview of the strengths and limitations of rule-based preprocessing compared with traditional algorithmic procedures. To discuss the differences between each approach, we take MOA, MOAReduction and Esper as reference. Table 9 summarizes their characteristics regarding stream definition, implementation details and currently supported preprocessing tasks. As each framework imposes its own processing language, some differences arise regarding how they operate with data streams. In this sense, Esper includes efficient native implementations of concepts that are inherent to stream processing. The support for window definition, both spatial and temporal, as well as the availability of numerous pattern operators, such as followed by or group by should be highlighted in favour of rule-based preprocessing using CEP. These functionalities are not currently supported by MOA, meaning that the user would have to implement them from scratch using Java mechanisms, for example, queues to simulate windows. The same applies to MOAReduction, though some of its algorithms can be parameterized to specify the frequency of application, a sort of window mechanism.

Another advantage of Esper is the possibility of defining and monitoring multiple input flows, from which rules can be triggered in parallel. Implementing such a functionality in MOA or MOAReduction would require knowledge of concurrency in Java. In contrast, MOA is highly efficient in the management of instances, which are encoded as double arrays. Although MOAReduction follows the same approach, some methods perform instance conversion between the MOA and the Weka implementation of an instance. Conversion is also needed in Esper, since each instance has to be constructed from events implemented as user-defined Java classes. However, the overhead is low according to the results shown in Section 5.1. Other formats (e.g., JSON or Avro) could be explored in the future, although some cost associated with the conversion to MOA instances will still remain.

TABLE 8 Execution time (in seconds) of Hoeffding tree with different preprocessing techniques to impute MV in the occupancy data stream

Data stream	MOA (mean)	MOA (max)	MOA (min)	MOA (zero)	CEP (remove)	CEP (estimate)	CEP (average)	CEP (last)
Random (5%)	0.09 ± 2.89E-4	0.10 ± 5.21E-4	0.10 ± 1.28E-3	0.10 ± 1.35E-3	0.17 ± 6.87E-3	0.22 ± 1.49E-2	0.28 ± 9.96E-3	3.25 ± 0.26
Random (10%)	0.09 ± 1.13E-3	0.09 ± 1.19E-3	0.10 ± 1.26E-3	0.10 ± 1.76E-3	0.16 ± 1.80E-3	0.23 ± 1.85E-2	0.29 ± 1.02E-2	5.40 ± 0.54
Random (25%)	0.09 ± 7.99E-4	0.10 ± 4.30E-3	0.11 ± 4.18E-4	0.10 ± 1.29E-3	0.14 ± 1.62E-3	0.24 ± 1.89E-2	0.31 ± 7.94E-3	8.22 ± 0.49
Random (50%)	0.09 ± 3.36E-3	0.11 ± 9.12E-3	0.11 ± 7.75E-3	0.10 ± 5.51E-3	0.12 ± 1.08E-2	0.23 ± 9.44E-3	0.34 ± 1.34E-2	9.80 ± 0.19
Block {100,5}	0.09 ± 4.68E-4	0.10 ± 4.04E-3	0.10 ± 1.42E-3	0.10 ± 2.46E-3	0.17 ± 1.30E-2	0.23 ± 1.65E-2	0.29 ± 1.86E-2	2.39 ± 0.16
Block {100,10}	0.10 ± 1.32E-3	0.10 ± 2.50E-3	0.10 ± 1.76E-3	0.10 ± 8.44E-4	0.16 ± 2.47E-3	0.23 ± 1.45E-2	0.29 ± 1.76E-2	3.39 ± 0.10
Block {100,15}	0.10 ± 1.71E-3	0.10 ± 3.00E-3	0.10 ± 3.74E-4	0.10 ± 7.12E-4	0.16 ± 1.40E-3	0.23 ± 1.95E-2	0.31 ± 2.29E-2	3.87 ± 0.18
Block {500,5}	0.10 ± 3.03E-3	0.10 ± 3.25E-4	0.09 ± 2.20E-3	0.10 ± 1.92E-3	0.17 ± 7.01E-3	0.24 ± 1.95E-2	0.27 ± 7.95E-3	0.94 ± 0.04
Block {500,10}	0.10 ± 1.92E-3	0.10 ± 1.01E-3	0.10 ± 4.48E-4	0.10 ± 9.68E-4	0.18 ± 1.59E-2	0.23 ± 1.57E-2	0.27 ± 9.41E-3	1.21 ± 0.06
Block {500,15}	0.09 ± 3.88E-4	0.10 ± 4.28E-4	0.09 ± 3.54E-4	0.10 ± 1.44E-3	0.17 ± 1.85E-3	0.23 ± 1.73E-2	0.28 ± 1.41E-2	1.56 ± 0.04

Abbreviation: CEP, complex event processing.

TABLE 9 Main characteristics of MOA, MOAReduction and Esper w.r.t. data preprocessing

	MOA	MOAReduction	Esper
Stream characteristics			
Window	No	Partial	Yes
Pattern operators	No	No	Yes
Multiple inputs	No	No	Yes
Needs conversion	No	Partial	Yes
Implementation details			
Instance	Double array	Double array	User-defined event
Data types	ARFF types	ARFF types	Java types
Procedure	Filter	Filter, algorithm	Rules
I/O format	ARFF, comma-separated values (CSV)	ARFF	ARFF, CSV
Preprocessing tasks			
Data preparation	Replace MV		Replace MV
	Add noise		Data transformation
Data reduction	Attribute selection	Discretization	Attribute selection
		Instance selection	Feature generation
		Feature selection	

Regarding input and output formats, MOA and MOAReduction support ARRF data types, that is, numerical, nominal, string and date, whereas Esper allows objects and primitive types to be part of the event. In order to use our learning component – which is built on top of MOA –, output instances have to comply with the ARFF specification too. However, other tools could be considered for learning purposes after CEP preprocessing, if desired. Regarding data format, Esper supports CSV but MOA has some restrictions with that format when used for classification purposes, due to the lack of attribute meta-information. Therefore, we opted to use the ARFF format to maintain the same format throughout the whole workflow.

The most distinctive characteristic of our solution is the way preprocessing procedures are implemented. Rules are intuitive mechanisms, but not all preprocessing tasks might be easily expressed in the form of rules. A typical algorithmic flow may be more directly applicable in some situations. For instance, feature selection methods often rely on the dynamic analysis of the classification performance of groups of features, or are embedded in the learning algorithm (Ramírez-Gallego et al., 2017). At the moment, our approach is designed to apply preprocessing rules independently from the learning process. Contrary to Weka, MOA lacks a preprocessing tab in the GUI, so filters have to be configured as part of the mining process or invoked through code. Likewise, MOAReduction embeds some preprocessing methods into ML algorithms, though independent filters are predefined too.

Focusing on the type of preprocessing tasks currently supported by each framework, some differences also exist. MOAReduction is a specialized extension for data reduction, which covers most of its common tasks (see Table 1). MOA and CEP are more general solutions in this regard, and both can be used to replace missing values and select attributes. This latter option differs from feature selection, only available in MOAReduction, in that no algorithmic procedure is applied to automatically decide the features to be kept. Regarding value generation and data transformation, MOA only includes a filter to add noise, while discretization methods in MOAReduction are the only ones that alter numerical features. As for Esper, our experiments have shown how data transformation and feature generation can be addressed by combining CEP operators. Neither MOA nor MOAReduction implement any filter able to enrich the stream with new features in such a flexible way.

6 | CONCLUDING REMARKS

This paper proposes a rule-based approach to address preprocessing tasks in the context of stream data mining. To formalize the idea, we take inspiration from CEP concepts, handling raw data as events and expressing how to manipulate them by means of ECA rules. Different types of preprocessing rules are defined, thus providing a novel and readable way to express common procedures to transform, enrich or curate incoming flows of information.

To materialize the idea, we have used Esper, a CEP implementation that provides an SQL-like syntax for high-level rule definition and a wide range of operators. These characteristics are expected to help domain experts in their preprocessing activities. The native definition of window mechanisms and the possibility of user-defined functions will allow them to easily adapt their requirements to the data specific conditions and to include domain-specific factors. Our solution is integrated with MOA, the most popular library for online data mining, thus making it possible to perform preprocessing and learning in one single step.

Three experiments illustrate how rule-based preprocessing can be effectively used for data transformation, feature generation and missing value replacement for diverse application domains. Simple yet powerful preprocessing rules have proven to be able to generate high quality data streams, improving the accuracy of supervised algorithms. In addition, some of the obtained decision models are less complex than those learned from the original data stream, thus making them and the obtained knowledge more manageable and understandable.

These benefits compensate the time and effort required by preprocessing. Our experiments show that time and memory resources do not dramatically increase when new information is added to the learning phase. Also, feeding standard algorithms with CEP-preprocessed streams achieve better results than other types of preprocessing techniques, such as discretization, in less time.

In the future, we plan to develop more examples to show the suitability of our rule-based approach for other preprocessing tasks, as well as extend its definition to support heterogeneous flows. Our conceptual framework could be implemented in other stream processing engines like Flink or Azure and stream mining solutions like SAMOA (Morales & Bifet, 2015) and scikit-Multiflow (Montiel et al., 2018). We hypothesize that the event-based nature of CEP might also be exploited to detect concept drift (Sobolewski & Wozniak, 2017). More specifically, new types of CEP rules could be defined to analyse the stream data distribution and trigger events depending on the type of concept drift detected. Finally, empirical studies could be designed to analyse whether using a rule-based approach makes data stream preprocessing a lighter task.

ACKNOWLEDGEMENTS

The authors would like to thank the reviewers of the paper for their insightful comments and very valuable suggestions, which have helped us significantly to improve this work. This work is partially supported by the European Commission (FEDER), the Spanish Government under project COSCA (PGC2018-094905-B-I00) and the Andalusian Regional Government (DOC_00944). Open access funding enabled and organized by University of Córdoba.

CONFLICT OF INTEREST

The authors declare there is no conflict of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in Github at https://github.com/atenearesearchgroup/CEP-Preprocessing.

ORCID

Aurora Ramírez https://orcid.org/0000-0002-1916-6559
Nathalie Moreno https://orcid.org/0000-0001-9470-0283
Antonio Vallecillo https://orcid.org/0000-0002-8139-9986

ENDNOTES

[1] Available from https://www.github.com/atenearesearchgroup/CEP-Preprocessing

[2] Esper v8: https://www.espertech.com/esper/ (accessed 18 May 2021).

[3] For code details, the reader is referred to the Github repository: https://www.github.com/atenearesearchgroup/CEP-Preprocessing

[4] ARFF format specification: https://waikato.github.io/weka-wiki/formats_and_processing/arff_stable/ (accessed 18 May 2021).

[5] UCI repository: https://archive.ics.uci.edu/ml (accessed 18 May 2021).

[6] OpenML repository: https://www.openml.org (accessed 18 May 2021).

[7] https://www.github.com/atenearesearchgroup/CEP-Preprocessing

[8] https://www.openml.org/d/151 (accessed 18 May 2021).

[9] https://www.openml.org/d/1169 (accessed 18 May 2021).

[10] Note that this method temporarily stores such statistics in a text file.

[11] https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+ (accessed 18 May 2021).

[12] We omit the option 'Nothing' in both cases, since the resulting data stream is not modified. Default constant value is zero.

[13] https://github.com/atenearesearchgroup/CEP-Preprocessing/blob/master/Technical_Report_CEP_for_data_preprocessing.pdf

[14] https://www.espertech.com/esper/esper-documentation/ (accessed 18 May 2021).

REFERENCES

Affetti, L., Tommasini, R., Margara, A., Cugola, G., & Della Valle, E. (2017). Defining the execution semantics of stream processing engines. *Journal of Big Data*, *4*(1), 12. https://doi.org/10.1186/s40537-017-0072-9

Ancy, S., & Paulraj, D. (2019). Online learning model for handling different concept drifts using diverse ensemble classifiers on evolving data streams. *Cybernetics and Systems*, *50*(7), 579–608. https://doi.org/10.1080/01969722.2019.1645996

Barddal, J. P., Enembreck, F., Gomes, H. M., Bifet, A., & Pfahringer, B. (2019). Merit-guided dynamic feature selection filter for data streams. *Expert Systems with Applications*, *116*, 227–242. https://doi.org/10.1016/j.eswa.2018.09.031

Barddal, J. P., Gomes, H. M., Enembreck, F., & Pfahringer, B. (2017). A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*, *127*, 278–294. https://doi.org/10.1016/j.jss.2016.07.005

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). MOA: Massive online analysis. *Journal of Machine Learning Research*, *11*, 1601–1604.

Bifet, A., Holmes, G., & Pfahringer, B. (2010). Leveraging bagging for evolving data streams. In J. L. Balcázar, F. Bonchi, A. Gionis, & M. Sebag (Eds.), *Machine learning and knowledge discovery in databases* (pp. 135–150). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-15880-3_15

Bilalli, B., Abelló, A., Aluja-Banet, T., & Wrembel, R. (2019). PRESISTANT: Learning based assistant for data pre-processing. *Data & Knowledge Engineering*, *123*, 101727. https://doi.org/10.1016/j.datak.2019.101727

Bollegala, D. (2017). Dynamic feature scaling for online learning of binary classifiers. *Knowledge-Based Systems*, *129*, 97–105. https://doi.org/10.1016/j.knosys.2017.05.010

Bolon-Canedo, V., Fernández-Francos, D., Peteiro-Barral, D., Alonso-Betanzos, A., Guijarro-Berdiñas, B., & Sánchez-Maroño, N. (2016). A unified pipeline for online feature selection and classification. *Expert Systems with Applications*, *55*, 532–545. https://doi.org/10.1016/j.eswa.2016.02.035

Bruns, R., Dunkel, J., & Offel, N. (2019). Learning of complex event processing rules with genetic programming. *Expert Systems with Applications*, *129*, 186–199. https://doi.org/10.1016/j.eswa.2019.04.007

Candanedo, L. M., & Feldheim, V. (2016). Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and Buildings*, *112*, 28–39. https://doi.org/10.1016/j.enbuild.2015.11.071

Cano, A., & Krawczyk, B. (2019). Evolving rule-based classifiers with genetic programming on gpus for drifting data streams. *Pattern Recognition*, *87*, 248–268. https://doi.org/10.1016/j.patcog.2018.10.024

Cano, A., & Krawczyk, B. (2020). Kappa Updated Ensemble for drifting data stream mining. *Mach.Learn.*, *109*(1), 175–218. https://doi.org/10.1007/s10994-019-05840-z

Corporate Act-Net Consortium. (1996). The active database management system manifesto: A rulebase of ADBMS features. *SIGMOD Record*, *25*(3), 40–49. https://doi.org/10.1145/234889.234896

Crone, S., Lessmann, S., & Stahlbock, R. (2006). The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European Journal of Operational Research*, *173*(3), 781–800. https://doi.org/10.1016/j.ejor.2005.07.023

Cugola, G., & Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, *44*(3), 15:1–15:62. https://doi.org/10.1145/2187671.2187677

Demirhan, H., & Renwick, Z. (2018). Missing value imputation for short to mid-term horizontal solar irradiance data. *Applied Energy*, *225*, 998–1012. https://doi.org/10.1016/j.apenergy.2018.05.054

Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Kamp, M., & Mock, M. (2017). Issues in complex event processing: Status and prospects in the big data era. *Journal of Systems and Software*, *127*, 217–236. https://doi.org/10.1016/j.jss.2016.06.011

Fülöp, L. J., Beszédes, A., Tóth, G., Demeter, H., Vidács, L., & Farkas, L. (2012). *Predictive complex event processing: A conceptual framework for combining complex event processing and predictive analytics* (pp. 26–31). In Proceedings of the 5th Balkan Conference in Informatics (BCI). doi: https://doi.org/10.1145/2371316.2371323

Gaber, M. M. (2012). Advances in data stream mining. *WIREs Data Mining and Knowledge Discovery*, *2*(1), 79–85. https://doi.org/10.1002/widm.52

Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005). Mining data streams: A review. *ACM SIGMOD Record*, *34*(2), 18–26. https://doi.org/10.1145/1083784.1083789

Gama, J. (2010). *Knowledge discovery from data streams* (1st ed.). Chapman & Hall.

Gama, J., & Kosina, P. (2011). Learning decision rules from data streams (pp. 1255–1260). In *Proceedings of the 22th International Joint Conference on Artificial Intelligence (IJCAI)*. doi: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-213

Gama, J., & Pinto, C. (2006). Discretization from data streams: Applications to histograms and data mining (pp. 662–667). In *Proceedings of the ACM Symposium on Applied Computing (SAC)*. ACM. doi: https://doi.org/10.1145/1141277.1141429

García, S., Luengo, J., & Herrera, F. (2014). *Data preprocessing in data mining* (1st ed.). Springer. https://doi.org/10.1007/978-3-319-10247-4

Ghomeshi, H., Gaber, M. M., & Kovalchuk, Y. (2019, May 01). EACD: Evolutionary adaptation to concept drifts in data streams. *Data Mining and Knowledge Discovery*, *33*(3), 663–694. https://doi.org/10.1007/s10618-019-00614-6

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., & Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, *106*, 1469–1495. https://doi.org/10.1007/s10994-017-5642-8

Gomes, J. B. A., Rodrigues, J. J. P. C., Rab˘elo, R. A. L., Tanwar, S., Al-Muhtadi, J., & Kozlov, S. (2020). A novel internet of things-based plug-and-play multigas sensor for environmental monitoring. *Transactions on Emerging Telecommunications Technologies*, *32*, e3967. https://doi.org/10.1002/ett.3967

Grosan, C., & Abraham, A. (2011). Rule-based expert systems. In *Intelligent systems: A modern approach* (pp. 149–185). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-21004-4_7

Groth, D. P. (2004). An evaluation of a rule-based language for classification queries. In *Proceedings of the International Conference on Applications of Declarative Programming and Knowledge Management Workshop on Logic Programming* (Vol. 3392, pp. 79–97). Springer. doi: https://doi.org/10.1007/11415763_6

Hammami, Z., Sayed-Mouchaweh, M., Mouelhi, W., & Said, L. B. (2020). Neural networks for online learning of non-stationary data streams: A review and application for smart grids flexibility improvement. *Artificial Intelligence Review*, *53*, 6111–6154. https://doi.org/10.1007/s10462-020-09844-3

Han, J., Kamber, M., & Pei, J. (2012). Data preprocessing. In *Data mining* (3rd ed., pp. 83–124). Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-381479-1.00003-4

Huang, M.-W., Lin, W.-C., Chen, C.-W., Ke, S.-W., Tsai, C.-F., & Eberle, W. (2016). Data preprocessing issues for incomplete medical datasets. *Expert Systems*, *33*(5), 432–438. https://doi.org/10.1111/exsy.12155

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the 7th ACM Sigkdd International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 97–106). doi: https://doi.org/10.1145/502512.502529

Jin, R., & Agrawal, G. (2007). Frequent pattern mining in data streams. In C. C. Aggarwal, (Ed.), *Data streams: Models and algorithms* (pp. 61–84). Springer US. https://doi.org/10.1007/978-0-387-47534-9_4

Kosar, T., Mernik, M., Crepinsek, M., Henriques, P. R., da Cruz, D. C., Pereira, M. J. V., & Oliveira, N. (2009). Influence of domain-specific notation to program understanding. In *Proceeding of the International Multiconference on Computer Science and Information Technology (IMCSIT)* (pp. 675–682). IEEE. doi: https://doi.org/10.1109/IMCSIT.2009.5352767

Kosar, T., Oliveira, N., Mernik, M., Pereira, M. J. V., Crepinsek, M., da Cruz, D. C., & Henriques, P. R. (2010). Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems*, *7*(2), 247–264. https://doi.org/10.2298/CSIS1002247K

Kousiouris, G., Akbar, A., Sancho, J., Ta-shma, P., Psychas, A., Kyriazis, D., & Varvarigou, T. (2018). An integrated information lifecycle management framework for exploiting social network data to identify dynamic large crowd concentration events in smart cities applications. *Future Generation Computer Systems*, *78*, 516–530. https://doi.org/10.1016/j.future.2017.07.026

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., & Wozniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion*, *37*, 132–156. https://doi.org/10.1016/j.in_us.2017.02.004

Lee, O.-J., & Jung, J. E. (2017). Sequence clustering-based automated rule generation for adaptive complex event processing. *Future Generation Computer Systems*, *66*, 100–109. https://doi.org/10.1016/j.future.2016.02.011

Lehtinen, P., Saarela, M., & Elomaa, T. (2012). Online ChiMerge algorithm. In D. E. Holmes & L. C. Jain (Eds.), *Data mining: Foundations and intelligent paradigms: Volume 2: Statistical, bayesian, time series and other theoretical aspects* (pp. 199–216). Springer. https://doi.org/10.1007/978-3-642-23241-1_10

Liu, A., Song, Y., Zhang, G., & Lu, J. (2017). Regional concept drift detection and density synchronized drift adaptation. In C. Sierra (Ed.), *Proceedings of 26th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 2280–2286). IJCAI (International Joint Conferences on Artificial Intelligence Organization). https://doi.org/10.24963/ijcai.2017/317

Liu, A., Zhang, G., & Lu, J. (2017). Fuzzy time windowing for gradual concept drift adaptation. In Proceedings of the 2017 IEEE International Conference on Fuzzy Systems (fuzz-IEEE) (pp. 1–6). IEEE. doi: https://doi.org/10.1109/FUZZ-IEEE.2017.8015596

Loreti, D., Chesani, F., Mello, P., Roffia, L., Antoniazzi, F., Cinotti, T. S., Paolini, G., Masotti, D., & Costanzo, A. (2019). Complex reactive event processing for assisted living: The habitat project case study. *Expert Systems with Applications*, *126*, 200–217. https://doi.org/10.1016/j.eswa.2019.02.025

Lu, J., Yang, Y., & Webb, G. I. (2006). Incremental discretization for naïve-bayes classifier. In L Xue, R. Osmar & L. Zhanhuai, (Eds.), *Advanced data mining and applications* (pp. 223–238). Springer.

Lu, N., Lu, J., Zhang, G., & de Mántaras, R. L. (2016). A concept drift-tolerant case-base editing technique. *Artificial Intelligence*, *230*, 108–133. https://doi.org/10.1016/j.artint.2015.09.009

Luckham, D. (2001). *The power of events: An introduction to complex event processing in distributed Enterprise systems*. Addison-Wesley.

Margara, A., Cugola, G., & Tamburrelli, G. (2014). Learning from the past: Automated rule generation for complex event processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS)* (pp. 47–58). doi: https://doi.org/10.1145/2611286.2611289

Montiel, J., Read, J., Bifet, A., & Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, *19*(72), 1–5 Retrieved from http://jmlr.org/papers/v19/18-251.html

Morales, G. D. F., & Bifet, A. (2015). Samoa: Scalable advanced massive online analysis. *Journal of Machine Learning Research*, *16*, 149–153 Retrieved from http://jmlr.org/papers/v16/morales15a.html

Mousheimish, R., Taher, Y., & Zeitouni, K. (2017). Automatic learning of predictive CEP rules: Bridging the gap between data mining and complex event processing. In *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems (DEBS)* (pp. 158–169). doi: https://doi.org/10.1145/3093742.3093917

Nguyen, H.-L., Woon, Y.-K., & Ng, W.-K. (2015). A survey on data stream clustering and classification. *Knowledge and Information Systems*, *45*(3), 535–569. https://doi.org/10.1007/s10115-014-0808-1

Ölmezogullari, E., & Ari, I. (2013). Online association rule mining over fast data. In *Proceedings of the IEEE International Congress on Big Data* (pp. 110–117). doi: https://doi.org/10.1109/BigData.Congress.2013.77

Pardini, K., Rodrigues, J. J., Diallo, O., Das, A. K., de Albuquerque, V. H. C., & Kozlov, S. A. (2020). A smart waste management solution geared towards citizens. *Sensors*, *20*(8), 2380. https://doi.org/10.3390/s20082380

Prasad, B., & Agarwal, S. (2016). Stream data mining: Platforms, algorithms, performance evaluators and research trends. *International Journal of Database Theory and Application*, *9*(9), 201–218.

Ramírez-Gallego, S., García, S., & Herrera, F. (2018). Online entropy-based discretization for data streaming classification. *Future Generation Computer Systems*, *86*, 59–70. https://doi.org/10.1016/j.future.2018.03.008

Ramírez-Gallego, S., Krawczyk, B., García, S., Wozniak, M., & Herrera, F. (2017). A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, *239*, 39–57. https://doi.org/10.1016/j.neucom.2017.01.078

Sobolewski, P., & Wozniak, M. (2017). SCR: Simulated concept recurrence–anon-supervised tool for dealing with shifting concept. *Expert Systems*, *34*(5), e12059. https://doi.org/10.1111/exsy.12059

Souza, V. M., dos Reis, D. M., Maletzke, A. G., & Batista, G. E. (2020). Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery*, *34*, 1805–1858. https://doi.org/10.1007/s10618-020-00698-5

Sun, A. Y., Zhong, Z., Jeong, H., & Yang, Q. (2019). Building complex event processing capability for intelligent environmental monitoring. *Environmental Modelling & Software*, *116*, 1–6. https://doi.org/10.1016/j.envsoft.2019.02.015

Uysal, A. K., & Gunal, S. (2014). The impact of preprocessing on text classification. *Information Processing & Management*, *50*(1), 104–112. https://doi.org/10.1016/j.ipm.2013.08.006

Wang, Y., Gao, H., & Chen, G. (2018). Predictive complex event processing based on evolving Bayesian networks. *Pattern Recognition Letters*, *105*, 207–216. https://doi.org/10.1016/j.patrec.2017.05.008

Webb, G. I. (2014). Contrary to popular belief incremental discretization can be sound, computationally efficient and extremely useful for streaming data. In *Proceedings of the IEEE International Conference on Data Mining* (pp. 1031–1036). doi: https://doi.org/10.1109/ICDM.2014.123

Webb, G. I., Lee, L. K., Goethals, B., & Petitjean, F. (2018). Analyzing concept drift and shift from sample data. *Data Mining and Knowledge Discovery*, *32*(5), 1179–1199. https://doi.org/10.1007/s10618-018-0554-1

Zhang, L., Lin, J., & Karim, R. (2017). Sliding window-based fault detection from high-dimensional data streams. *IEEE Transactions on Systems, Man, and Cybernetics: Systems.*, *47*(2), 289–303. https://doi.org/10.1109/TSMC.2016.2585566

Zhang, S., Zhang, C., & Yang, Q. (2003). Data preparation for data mining. *Applied Artificial Intelligence*, *17*(5–6), 375–381. https://doi.org/10.1080/713827180

AUTHOR BIOGRAPHIES

Aurora Ramírez received the M.Sc. and Ph.D. degrees in Computer Science from the University of Córdoba, Córdoba, Spain, in 2012 and 2018, respectively. She is currently a postdoctoral researcher of the Knowledge Discovery and Intelligent Systems Research Laboratory from the same university, where she conducts her research in artificial intelligence applied to software engineering. Her research interests are search-based software engineering, data mining and knowledge discovery, explainable artificial intelligence, and bio-inspired computing. She has authored more than 25 publications in journals and conferences. She has also participated as a member of the program committee in more than 20 international conferences, and acts as reviewer for journals like IEEE Transactions on Software Engineering, Applied Soft Computing, and Information Sciences, among others.

Nathalie Moreno received the M.Sc. and Ph.D. degrees in Computer Science from the Department of Computer Science, University of Málaga, Spain, in 2012, where she is currently an Assistant Professor. Her research interests are mainly oriented towards model-driven development. In particular, she focuses on conceptual modeling methodologies, business process modeling, model transformation languages, and uncertainty on complex event processing systems for its application on the Internet of Things.

Antonio Vallecillo is Professor of Software Engineering at the University of Málaga, Spain, where he leads the Atenea Research Group. His current research interests include Model-based Software Engineering, Open Distributed Processing, and Software Quality. Information about his publications, research projects, and activities can be found at http://www.lcc.uma.es/˜av.

How to cite this article: Ramírez, A., Moreno, N., & Vallecillo, A. (2021). Rule-based preprocessing for data stream mining using complex event processing. *Expert Systems*, e12762. https://doi.org/10.1111/exsy.12762

APPENDIX

Esper operators for data stream preprocessing

Based on the classification of preprocessing tasks and the types of CEP operators presented in Section 2, Table A1 summarizes suitable CEP operators for each preprocessing task, including examples of EPL functions and clauses that implement such operators in Esper. Note that most of these functions exist in other CEP engines, though their semantics and behaviour might be slightly different. The interested reader is referred to our technical report for a more detailed analysis,[13] and the Esper API documentation[14] for further information on Esper functions and clauses.

TABLE A1 Applicable operators, clauses and functions for each type of preprocessing task

Data preparation		
Task	Operators	Functions and clauses
Cleaning	- Selection	any, all, like, some
	- Logic	distinct`Of`, except, exists
Transformation	- Selection	all, any, like, some
	- Arithmetic	cast, Math. round, toDate
Normalization	- Arithmetic	avg, maxever, minever, stddev
	- Logic	>, >=, <, <=
Integration	- Flow	join, order by, group by
	- Sequence	first, last, prev, take, ⇝
	- Creation	insert into, output
Data reduction		
Task	Operators	Functions and clauses
Feature selection	- Projection	select < *attribute* >
Instance selection	- Selection	any, all, like, some
	- Logic	distinct`Of`, except
	- Flow	group by
	- Sequence	first, last, prev, take, ⇝
Discretization	- Selection	between, in, not in
	- Logic	>, >=, <, <=
Value generation	- Arithmetic	avg, min, max, count, sum
	- Creation	insert into
	- Sequence	leastFrequent, mostFrequent