Technical Report

Technical Report: Data stream preprocessing for online data mining using complex event processing

Aurora Ramírez, Nathalie Moreno,
Antonio Vallecillo

September 2019

Abstract Data preprocessing is known to be essential to produce accurate data from which mining methods are able to extract valuable knowledge. Domain experts have to decide how to transform, clean or reduce data, defining and sometimes implementing such procedures by themselves. Furthermore, when data constantly arrives from one or more sources, preprocessing techniques need to be adapted to efficiently handle these data streams. The use of an stream engine to prepare streams for online data mining represents an unexplored alternative. Complex event processing (CEP) systems are information processing engines able to detect events and propagate decisions by means of domain-specific rules. Making a correspondence between events and instances, CEP could support experts in the definition of understandable rules to express preprocessing procedures.

This paper analyses the use of Complex Event Processing (CEP) for data stream preprocessing, and materialises the idea into a Java solution that integrates a CEP engine with a library for online data mining. The main contribution of our approach is the definition of preprocessing tasks using event detection rules expressed in a SQL-like language. To assist in this process, we first provide an analysis of the CEP functionalities that could be applied for fast data stream preprocessing. Then, we present three practical scenarios to show how CEP rules can serve to preprocess data streams with the aim of adding temporal information, transforming features and handling missing values. The main findings of our research are: 1) CEP rules provide a domain-oriented language to express preprocessing tasks, 2) CEP operators are specially well-suited to deal with temporal information, 3) after CEP preprocessing, machine learning algorithms can significantly improve their accuracy and reduce the complexity of their decision models, and 4) the cost of CEP preprocessing does not imply significant time and memory overheads, even reducing the learning time in some cases.

Aurora Ramírez, Nathalie Moreno, Antonio Vallecillo are at Universidad de Málaga, Spain.
E-mail: {aramirez,moreno,av}@lcc.uma.es

Contents

1 Introduction . . . 2
2 Background . . . 4
  2.1 Data preprocessing . . . 4
  2.2 Stream data mining . . . 5
  2.3 Complex event processing . . . 6
3 Related work . . . 7
  3.1 Combining data mining and CEP . . . 8
  3.2 Data stream preprocessing techniques . . . 8
4 CEP for data stream preprocessing . . . 9
  4.1 Analysis of CEP functionalities . . . 10
  4.2 Implementation . . . 12
5 Experimental evaluation . . . 13
  5.1 Methodology . . . 14
  5.2 Experiment 1: Generating temporal features . . . 15
  5.3 Experiment 2: Integrating categories . . . 18
  5.4 Experiment 3: Dealing with missing values . . . 22
6 Discussion . . . 25
  6.1 Experimental findings . . . 26
  6.2 Strengths and limitations of MOA and CEP preprocessing . . . 29
7 Threats to validity . . . 30
8 Concluding remarks . . . 31

1 Introduction

Data preprocessing is a crucial step in any knowledge discovery process [20]. Cleaning, transforming and integrating data often represent laborious but essential tasks, since patterns are more easily extracted from structured data [42]. Furthermore, the application of an adequate preprocessing method can have a positive impact on the results of the mining process [11,40]. However, data preprocessing is usually a manual error-prone process, which demands some knowledge on statistics and programming skills. Tools like Weka and software packages like those available in R and Python provide support for common preprocessing tasks, but they require a learning curve for non-familiarised domain specialists. Besides, it is likely that they will need to fine-tune parameters or adapt general methods to optimally work on their application domain, making it difficult to reuse procedures.

With the growing interest of organisations in integrating advanced analytics to make informed decisions [37], novel computational techniques to extract knowledge from data streams [17] are highly demanded. The speed at which huge volumes of data are generated pose new challenges to both preprocessing and machine learning (ML) techniques [39]. Despite its relevance, data preprocessing has not been so frequently studied in the context of stream data mining [36]. 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 [4] or discretisation [35] are beginning to appear, they are rarely available in software libraries.

Fast-processing stream engines [1] might represent a candidate platform to implement preprocessing procedures, specially if the input data contain temporal information. Among them, complex event processing (CEP) systems allow users not only to process information flows, but also detect situations of interest and propagate decisions [12]. CEP has experienced a growing interest in the last years due to relevance that Internet of Things (IoT) applications are gaining for, e.g., environmental monitoring [38], healthcare [26], or smart mobility [24].

A distinctive characteristic of CEP systems is that they offer a rich language to express rules and patterns following a SQL-like syntax. For those cases in which transformation and filtering processes depend on the application domain, experts might find it easier to express them in the form of rules. In addition, CEP systems provide specific operators to establish temporal or spatial windows to operate with. This would 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.

Bearing these factors in mind, this paper explores the use of CEP as a novel mechanism to preprocess data streams before the data mining process. In short, the idea underlying our proposal is that CEP characteristics — fast stream processing, rule-based engine and SQL-like syntax — are specially well-suited for domain specialists who might not have deep knowledge of data mining methods and tools. Furthermore, our solution could reduce the burden needed to integrate such mining processes in organisations, thus promoting big data adoption [3]. The development of our proposal is guided by the following research questions (RQ):

RQ1: *How can CEP be used to prepare data streams for online data mining?* A throughout analysis of the CEP functionalities should be carried out to identify suitable operators and clauses for common preprocessing tasks. The possibility of defining windows to limit the scope of the rules should also be exploited as a way to derive useful temporal information.

RQ2: *Which benefits does CEP bring over other preprocessing methods?* An experimental study is needed to determine whether the resulting data streams have a positive impact on the results of data mining algorithms compared to not applying any preprocessing. Such improvements should be analysed not only in terms of prediction accuracy, but also from other perspectives like model complexity and computational resources (e.g., time and memory).

To the best of our knowledge, this paper represents the first time CEP is applied to define preprocessing rules for data streams. The contributions of this paper are enumerated below:

1. We formulate data stream preprocessing as a rule-based procedure. A theoretical analysis of CEP identifies those functionalities best suited for common preprocessing tasks. As a result, we provide a guide to help domain-specialists choose appropriate operators and functions to define their own stream preprocessing procedures.

2. We present a publicly available implementation of the approach[1] that integrates Esper,[2] a Java-based CEP engine, with MOA, a library for online data mining [6]. Having both preprocessing and mining steps under the same framework reduces integration efforts and makes information management easier.
3. We conduct three experiments using public datasets from different domains to illustrate the suitability of our proposal in practice. The experiments cover a representative set of preprocessing tasks, after which classification algorithms are applied. The results reveal 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. These factors are essential to ensure a wider adoption of stream data mining processes in real-world applications.
4. We also discuss the benefits of our approach with respect to other preprocessing alternatives. From this comparison, practitioners gain additional insights of CEP-based preprocessing in terms of native stream characteristics, implementation aspects and adaptability to diverse preprocessing tasks.

The rest of the paper is organised as follows. Section 2 introduces concepts related to data mining and complex event processing. Next, Section 3 presents related work. Our proposal for the use of CEP in data stream preprocessing is explained in Section 4 and experimentally evaluated in Section 5. The strengths and limitations of the approach are discussed in Section 6. Threats to validity are stated in Section 7. Finally, Section 8 concludes and discusses future lines of research.

2 Background

This section explains relevant concepts regarding data preprocessing and stream data mining. Next, the principles underlying CEP are presented.

2.1 Data preprocessing

One of the most important phases within the knowledge discovery from databases (KDD) process consists in preparing the data, a.k.a. instances, from which relevant information has to be extracted. Data preprocessing comprises techniques that adapt either the content or the format of raw data coming from multiple, possibly heterogeneous, sources [20]. The objective is to enhance the accuracy, completeness and consistency of data [21], thus making its management easier in subsequent phases of the KDD process. During the preprocessing step, it is possible to curate the input data by discarding irrelevant or incorrect values, restoring missing values where possible, or detecting outliers. The way

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

[2] Esper v8: `https://www.espertech.com/esper/` (accessed September 17, 2019).

Table 1 Data preprocessing tasks (adapted from [20]).

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

in which the attributes, a.k.a. features, of the instances are transformed can largely influence the effectiveness of mining algorithms. When algorithms are provided with quality structured data, they find it easier to detect patterns and produce more reliable predictions in less time [21].

Data preprocessing techniques are usually classified in two main groups according to its purpose: data preparation and data reduction. Table 1 summarises the principal tasks associated to each group [20], which can be applied in combination. Data preparation encompasses methods that detect and fix inconsistencies, e.g., missing values and noise, change data formats or transform values. Sometimes, data coming from multiple sources follow different schema that should be unified too. In contrast, data reduction is focused on altering the number of features or instances to keep only those that satisfy certain criteria. For instance, it is frequent to select subsets of features to reduce the dimensionality of the dataset, whereas some algorithms cannot deal with continuous data, so discretisation is required.

Data preprocessing becomes even more relevant in big data scenarios, where a variety of features and a vast volume of instances are expected [36]. When dealing with data streams, preprocessing techniques should be as lighter and automatic as possible [16]. Additional factors need to be considered due to the dynamic nature of the data. Accurate information regarding its distribution, e.g., ranges of values or dependencies among variables, is not fully available. In the presence of concept drift, the relative importance of the features might change over time [5], making feature selection even harder.

2.2 Stream data mining

Stream data mining differs from traditional mining processes in a number of aspects [17,32]. From the data perspective, each instance should be considered only once. Only in some applications with more relaxed constraints, small samples can be temporarily retained. The decision model has to be continuously updated, adding new relevant information and forgetting outdated data. Algorithms also need to include mechanisms to detect concept drift. Apart from being able to cope with these issues, stream mining algorithms are expected to provide real-time response and present low memory requirements. Due to the strict conditions in which these algorithms might operate, approximate results are acceptable.

Learning can be planned in two different ways [32]. Algorithms following an incremental learning approach update the decision model as new instances arrive. In turn, a two-phase learning combines an online and an offline phase, so that only a few instances are processed in real time. Then, the learning process takes place offline using a summary of the data, for which *windows* can be defined. Diverse strategies can be followed to create them, such as keeping only the most recent items or assigning weights to reduce the importance of old data [17].

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 [32,23]. Regression and classification are supervised methods that try to predict the value or class of a variable, respectively. Clustering, whose purpose is to identify groups of related items, and association rule mining, oriented towards extracting patterns describing the data, belong to the category of 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 [17]. Implementations of the most commonly used algorithms are available in MOA [6], a Java tool for massive online learning developed by the same institution than Weka. Preprocessing algorithms in MOA are known as *filters*, though the collection is not so extensive than in Weka. Current filters are able to add random noise, replace missing values and select attributes. Recently, an extension of MOA[3] has appeared with implementations of algorithms for feature and instance selection, as well as discretisation [36].

2.3 Complex event processing

Complex event processing is a form of information processing aimed at detecting situations of interest from events [27]. In CEP terminology, a *simple* event corresponds to low-level data, e.g., 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. Once registered, the CEP engine will analyse the rules to detect relevant events and accomplish the actions [12]. Traditional CEP systems follow a *publish-subscribe* approach, where subscribers collect

[3] MOAReduction: `https://moa.cms.waikato.ac.nz/moa-extensions/` (accessed September 17, 2019).

Table 2 Types of operators in CEP (adapted from [12]).

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.

rule outcomes to further process them. CEP rules commonly go through three phases:

1. *Selection.* The events triggering the rule are identified within the selected window.
2. *Matching.* The conditions that the event and its attributes should satisfy are checked.
3. *Production.* Complex events are created, possibly computing new attributes with aggregation functions.

Events, rules and patterns are defined using an *event processing language* (EPL), whose syntax is similar to SQL. Listing 1 show the general syntax of a CEP rule in EPL, comprised of three parts: 1) `select`, in which the relevant attributes are indicated (or the whole event, using *); 2) `from`, in which event flows or patterns over them are specified; 3) `where`, in which the conditions to be fulfilled are detailed.

```
select *|<attribute(s)>
from <flow(s)>|<pattern(s)>
where <condition(s)>
```

Listing 1 Syntax of a CEP rule in EPL.

The richness 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*, i.e., composed of a number of events, or *temporal*, i.e., time-based defined.

3 Related work

Our work is related to two main lines of research: the combination of data mining and CEP, and the use of data stream preprocessing techniques to improve the results of ML algorithms.

3.1 Combining data mining and CEP

Synergies between CEP and data mining have been explored in the last years, specially in the context of predictive analytics [15,14]. However, current approaches are mostly focused on applying learning algorithms to enhance CEP capabilities. For instance, ReCEPTor integrates three association rule mining algorithms into a CEP system [33]. The algorithms are defined as new EPL clauses, so that they can be applied by ReCEPTor rules to discover patterns as new events arrive. More recently, Evolving Bayesian Networks have been applied to model changes in the distribution of the event stream processed by CEP [41]. Complex events detected by manually-defined CEP rules have been used as the target to build predictive models [15]. This way, the system is able to anticipate their future appearance.

Given that manually defining CEP rules is a laborious task that largely depends on expert's knowledge, other authors have studied the possibility of automatically inferring such rules. 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 [28]. Different strategies are proposed to solve each task, combining ML techniques like support vector machines (SVM) with information provided by experts. Similarly, autoCEP is a two-phase approach that learns sequences from time series and transforms them into ready-to-deploy CEP rules [31]. Recently, GP4CEP proposes the use of genetic programming to find the conditions that best describe the occurrence of a predefined complex event [9]. In this case, the learned rules are expressed using logical and sequential operators, and can include both temporal and spatial windows. Finally, Adaptive CEP is focused on rule update, for which clustering and Markov probabilistic models are applied [25]. This approach is able to identify similar simple events and then build and dynamically update patterns representing complex events.

3.2 Data stream preprocessing techniques

Several general algorithms to address common data stream preprocessing tasks can be found in the literature. Online methods for normalisation have been recently proposed [7]. For unsupervised learning, the author uses the so-called 'update equations' to estimate the mean and the standard deviation that are employed to normalise the data. For supervised learning, values are scaled using a sigmoid function. Diverse 'update equations' are compared to determine its best scaling factors. The experimental analysis reveals that a simple average estimation performs relatively well. Neither implementation details nor preprocessing times are reported.

Ideas from time series analysis can be used to deal with missing values in numerical features. Recently, the impact of temporal windows on seven imputation methods has been studied in the context of solar irradiance forecasting [13]. Implemented as part of an R package, the compared methods range from simple statistics, e.g., 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 [5]. Recent approaches rely on different statistics that incrementally determine the relevance of each feature [8,4]. The first method sorts features using the χ^2 metric, counting how many times each category of a feature appears in instances of the same class. The method proposed in the second study to determine the relevance of each feature is considerably more complex. The authors dynamically update a merit function based on entropy that maximises feature relevance while minimising redundancy. Implemented in MOA, the method is evaluated in combination with four classifiers among those available in the library. Interestingly, sometimes the time overhead due to feature selection is compensated with a reduction of the learning time.

Finally, studies for stream discretisation adapt clustering, entropy and frequency-based methods to work in an online setting [19]. A recent proposal is LOFD, a discretisation algorithm that updates interval definition by splitting and merging intervals [35]. The method is included in the MOA extension for data reduction.

4 CEP for data stream preprocessing

This section lays the foundations of a new synergy between CEP and data mining: the use of CEP to build preprocessing rules for data streams. Formally, a preprocessing rule is an expression in the form $C \rightarrow A$, where C is the condition that triggers the rule and A is the action to be performed. For data streams, C represents the arrival of n types of events (E) with different attributes (at): $C = E_1\{at_1, ..., at_i\} \wedge ... \wedge E_n\{at_1, ..., at_j\}$. The action is a set of m operations that transform the events into an instance (I) with new features (f): $A = I\{f_1 = op_1(E_1, ..., E_n, w(s)), ..., f_m = op_m(E_1, ..., E_n, w(t))\}$, where w symbolises an optional window, either spatial (of size s) or temporal (valid for a time interval, t).

These rules can be inspired by the same principles guiding current preprocessing algorithms, but properly adapted to the application domain and expert's knowledge. Furthermore, the use of CEP specific operators and its powerful window handling facilities opens up new possibilities for improving existing data preprocessing procedures and for defining new ones. In response to RQ1, we carry out an analysis of CEP functionalities to identify suitable operators for different kinds of preprocessing tasks. Then, we describe the implementation of the proposed approach, which integrates a CEP engine and incremental ML algorithms.

Table 3 Applicable operators, clauses and functions for each type of preprocessing task.

Data preparation		
Task	**Operators**	**Functions and clauses**

Cleaning	- Selection
	- Logic
		`any`, `all`, `like`, `some`
		`distinctOf`, `except`, `exists`

Transformation	- Selection
	- Arithmetic
		`all`, `any`, `like`, `some`
		`cast`, `Math.round`, `toDate`

Normalisation	- Arithmetic
	- Logic
		`avg`, `maxever`, `minever`, `stddev`
		`>`, `>=`, `<`, `<=`

Integration	- Flow
	- Sequence
	- Creation
		`join`, `order by`, `group by`
		`first`, `last`, `prev`, `take`, $\rightarrow$
		`insert into`, `output`

Data reduction		
Task	**Operators**	**Functions and clauses**
Feature selection	- Projection	`select`<*attribute*>

Instance selection	- Selection
	- Logic
	- Flow
	- Sequence
		`any`, `all`, `like`, `some`
		`distinctOf`, `except`
		`group by`
		`first`, `last`, `prev`, `take`, $\rightarrow$

Discretisation	- Selection
	- Logic
		`between`, `in`, `not in`
		`>`, `>=`, `<`, `<=`

Value generation	- Arithmetic
	- Creation
	- Sequence
		`avg`, `min`, `max`, `count`, `sum`
		`insert into`
		`leastFrequent`, `mostFrequent`

4.1 Analysis of CEP functionalities

Starting from the classification of preprocessing tasks and the types of CEP operators presented in Section 2, we have identified suitable CEP operators for each preprocessing task. Table 3 summarises our analysis, including examples of EPL functions and clauses that implement such operators in Esper. Notice that most of these functions exist in other CEP engines, though their semantic and behaviour might be slightly different.

Regarding data cleaning, CEP rules act as filters that keep only those instances — events, in CEP — satisfying certain conditions. Such constraints should be specified in the `where` clause as logical expressions over event attributes, i.e., the features of the incoming instance. In addition to simple comparison operators ($>$, $<$, $\neq$), CEP provides functions like `exists`, which could be applied to detect missing values. More complex expressions can be designed in form of subqueries, using operators like `any` or `some`. Two relevant operators for noise and outlier detection are `distinctOf` and `except`, as they enable setting the values that attributes should present.

Rules for data transformation apply selection operators to identify those event attributes whose values need to be modified. The operators listed in Table 3 are highly flexible, as they work with regular expressions. The arithmetic operator `cast` serves to convert data types, including date formats. Using Esper as CEP engine, it is also possible to include functions from Java packages, e.g., *Math*. If the goal is to normalise values, functions to compute the average (`avg`) and standard deviation (`stddev`) are available in CEP too. As for maximum values, Esper distinguishes three variants (equivalent for minimum): `max`, to determine the maximum within a window; `fmax`, to include a filter in the expression; and `maxever`, to get the historical maximum.

CEP systems support multiple input streams, which become a relevant feature to tackle data integration during preprocessing. A first approach consists in selecting events from multiple streams using the `from` clause combined with constraints associating related events. This way, it would be possible to unify bank transactions received from different payment systems just by adding a equality comparison on account numbers. The alternative is to employ the `join` clause, which provides a specific syntax to combine streams. CEP includes operators to sort or limit events, combining `output` with `order by` and `limit`, respectively. In addition, events can be redirected to different flows by means of `insert into`. When windows are defined, operators like `first` and `last` are useful to retrieve specific events. Another relevant operator is `followed by` (represented as $\rightarrow$), since it allows establishing precedence relationships among events. E.g., this operator could be applied to detect purchase habits from which recommendations can be made.

Focusing on data reduction tasks, the `select` clause serves to extract subsets of attributes. This is not actually feature selection as such, since the features of interest have to be known in advance. However, CEP allows nesting `select` clauses, a functionality that could be used to create more complex expressions, e.g., features as result of dynamic queries. Furthermore, it would be possible to define several rules, each one selecting different features, and configure the CEP engine to dynamically activate and deactivate them in response to data distribution changes or temporal conditions.

In contrast, rules for instance selection seem to be easier to materialise, since they allow establishing constraints (`where` clause) on both event content and time of arrival. The combination of selection and logic operators provides the required flexibility to define conditions on attribute values. A simple example here would be a rule that only takes bank transactions exceeding a transfer amount in fraud detection systems. In presence of categorical attributes, the operator `group by` is really effective to split data instances by categories. This would allow specific treatments of data coming from different categories, including their combination. For instance, it would be necessary to create different disease prediction models for men and women.

For discretisation purposes, CEP provides specific operators to express conditions based on numerical intervals. In Esper, intervals are specified by means of the `between` operator as part of the `where` clause. Esper also defines `in` and `not in`, which should be followed by a sequence of values or a query result. Bounds can be established using standard logical operators ($>$, $<$) too. In any case, the ranges could be either set in advance or dynamically adjusted depending on data distribution statistics.

Value generation refers to both adding new features and inserting instances. In the former case, arithmetic operators, e.g., `avg`, `min` and `count`, are included in the `select` clause, requiring the name of the attribute for which they are computed as parameter. Such functions consider all previous events in the stream or within a particular window, including the current one. As the function is computed for every incoming event, it is actually a new feature that stores temporal information. Adding this type of feature would allow ML algorithms to take short-term history into account, adapting the decision model to data fluctuation. Applications in weather forecasting or those relying on market prices illustrate this point clearly.

Functions inspecting previous events in the flow are relevant for instance generation too. Instead of creating purely artificial instances, a preprocessing method can fill the instance with values obtained by historical statistics. Similarly, frequency-count functions are extremely useful to detect unbalanced data. In such situations, a rule responsible of injecting instances for the minority class should be activated. Previous events might be quite informative to make estimations or identify recurrent values as part of imputation methods to replace missing values.

Most of the aforementioned functions and operators are compatible with the definition of windows to limit their application scope. Furthermore, CEP is characterised by a great parameterisation and nesting capabilities. Finally, many CEP implementations allow users to integrate their own aggregation functions, thus supporting the definition of domain-specific transformations.

4.2 Implementation

Figure 1 shows the high-level structure of the proposed solution, which has been implemented in Java and is available from Github.[4] It comprises two main components: the first one (*CEP*) implements data stream preprocessing with CEP, while the second (*DM*) allows invoking data mining algorithms. Each component is explained next in more detail.

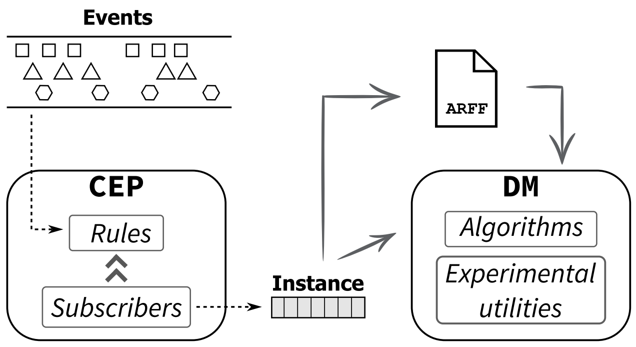

Fig. 1 Core elements of the proposed solution and their relationship.

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

The *CEP* component receives data in the form of simple events, which might come from multiple streams (represented with different shapes in Figure 1). A number of rules defining the preprocessing actions to be applied should be defined and conveniently registered. Likewise, a subscriber should be associated to each rule, which will be in charge of collecting and processing rule outcomes. From original events and information derived by the rules, the *CEP* component creates the instances to be sent to the incremental learning algorithm. The features of the output instance might represent original event attributes, with or without transformed values, as well as new information (derived attributes). Notice that, if needed, original attributes could be discarded too.

The *DM* component includes the algorithms and utilities required during the data mining phase. Since our proposal is oriented towards real-time data mining, instances are directly sent to the algorithm to train and validate its decision model. Currently available algorithms implement supervised approaches: regression by gradient descent and four classification methods (decision tree, kNN, naive Bayes and a rule-based classifier). For experimental purposes, our implementation also allows loading the generated instances from an ARFF[5] file, a common format among data mining tools.

Each component has been developed in Java as a *wrapper*. Esper provides the implementation of the CEP engine, whereas the MOA API gives access to data mining algorithms and performance evaluation measures. With the proposed solution, preprocessing a data stream is translated into the following workflow:

1. Define raw data as simple events with attributes in the CEP component (Java object).
2. Define the rule(s) to preprocess the events in EPL language using the operators described in Table 3.
3. Configure the CEP service to register the rules and inject the events in a flow.
4. Process the rule output to create the instance from which the ML algorithm will learn.

Steps 1 and 2 depend on the application domain and therefore should be programmed by the domain specialist. To assist in this process and evaluate the benefits of domain-oriented preprocessing, the next section presents three application scenarios that contribute showing how our approach could be used in practice.

5 Experimental evaluation

This section presents three experiments aimed at illustrating CEP capabilities in representative application domains. The experiments cover different types of

[5] ARFF format specification: `https://waikato.github.io/weka-wiki/formats_and_processing/arff_stable/` (accessed September 17, 2019)

preprocessing tasks: feature generation (Section 5.2), data transformation and integration (Section 5.3) and missing value management (Section 5.4). From these experiments, it is possible to analyse the benefits of CEP preprocessing as stated in RQ2. The methodology followed to conduct the experimentation is detailed first.

5.1 Methodology

Input data streams are generated from datasets publicly available in two well-known repositories: UCI[6] and OpenML.[7] Datasets in ARFF format are loaded instance by instance, thus simulating the arrival of simple events into CEP. Each dataset is described in its corresponding experiment, together with the preprocessing rules, as they are specific of the problem domain.

A *test-then-train* approach is followed in the learning phase. Each new instance is firstly used to test the current decision model, which is then trained with the instance [6]. 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 are selected for comparative purposes among those available in MOA: *Hoeffding tree* [22], which dynamically induces a decision tree; *k-nearest neighbours* [32], which classifies according to the k most similar instances; *naive Bayes* [32], which predicts the class probability of an instance based on the Bayes theorem; and a *rule-based classifier* [18], which derives a set of *if* $\rightarrow$ *then* classification rules.

To evaluate their performance, commonly used measures for binary classification are considered here: accuracy, i.e., percentage of correctly classified instances (either positives or negatives); 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 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:

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

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Algorithms are executed with and without applying CEP rules, so that the influence of preprocessing decisions, e.g., operators and window size, can

[6] UCI repository: `https://archive.ics.uci.edu/ml` (accessed September 17, 2019)
[7] OpenML repository: `https://www.openml.org` (accessed September 17, 2019)

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 studying the relative performance improvement, all algorithms are executed with default parameters (see MOA documentation). We consider this would be a common usage scenario for non-experts in data mining. All experiments were run in a Windows 10 computer with Intel Core i7-4790 at 3.60 GHz and 8 GB RAM. To reduce any possible bias due to machine overload, each experiment was repeated ten times to measure time and memory. Code, datasets, extended results and statistical validation are available for reproducibility purposes.[8]

5.2 Experiment 1: Generating temporal features

5.2.1 Experiment description

Predicting service demand often require analysing users' consumption habits and price trends. Both aspects are expected to change over time, meaning that any data mining algorithm would probably make better predictions if temporal factors are considered. As a practical example of this situation, a dataset containing the electricity price and demand of two Australian states — New South Wales and Victoria — is studied in this experiment. The dataset,[9] which is frequently used to evaluate stream data mining algorithms [32], is comprised of 45,312 instances, sampled every 30 minutes. 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, i.e., window size, that lead to the best performance, since fluctuations of price and demand are not known in advance.

5.2.2 Definition of CEP rules

In order to include the short-term history, we define two CEP rules using aggregation functions over four attributes: electricity price and demand in New South Wales (`nswprice` and `nswdemand`), and equivalent attributes for Victoria state (`vicprice` and `vicdemand`). The first rule (see Listing 2) computes the average value of each of these attributes within a sliding spatial window. Apart from applying the `avg` operator, the rule selects the whole event (*) to keep its original information. Therefore, four derived features are generated.

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

[9] `https://www.openml.org/d/151` (accessed September 17, 2019)

Analogously, the second rule obtains the minimum and maximum values of the attributes. Consequently, eight derived features are created.

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

Listing 2 Rule that computes the average price and demand of electricity.

We planned two different scenarios to analyse the influence of the derived features. Firstly, the instance is enriched with the temporal features. Secondly, 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. In addition, five different window sizes are considered when computing aggregation functions: 10, 50, 100, 500 and 1000.

5.2.3 Results

The four classification algorithms explained in Section 5.1 are executed for the original data stream and those generated by CEP. Table 4 compiles the results in terms of accuracy and F_1. The 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) and RC (rule-based classifier).

A first significant finding is that temporal features contribute to improve the performance of most of the algorithms (shaded cells), 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 in general, revealing that changes in electricity prices might also be explained by recent consumption habits. Indeed, 20 out of the 40 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, since any combination of aggregation function and window size guarantees better predictions. The one-tailed Wilcoxon test confirms this fact, which is a noteworthy achievement considering that HT already was the best algorithm among the four tested. In contrast, no improvement is observed for kNN, a method that 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. The one-tailed Wilcoxon test rejects the hypothesis that NB and RC perform better with enriched data streams, since accuracy decreases for some window sizes.

Table 4 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*	*HT*	*kNN*	*NB*	*RC*
Original		*79.20*	***78.38***	*73.36*	*73.21*	*78.48*	***77.96***	*70.18*	*71.58*
Average (−)	10	68.22	72.51	62.79	63.81	66.53	71.83	58.48	62.09
	50	64.97	72.03	60.32	61.29	63.58	71.34	57.46	58.84
	100	66.93	71.87	60.77	63.46	66.12	71.10	59.15	60.80
	500	66.76	71.39	59.47	62.54	65.84	70.57	56.32	60.44
	1000	66.60	71.37	59.49	61.73	65.65	70.56	56.15	58.66
Average (+)	10	82.37	77.09	70.76	71.91	82.01	76.66	67.35	70.47
	50	**93.12**	77.74	73.76	**79.23**	**93.09**	77.35	70.66	**77.63**
	100	86.15	77.32	**74.13**	73.25	85.79	76.86	**71.17**	71.66
	500	81.73	76.33	73.40	73.06	80.79	75.86	70.11	71.59
	1000	81.07	75.92	73.27	72.21	80.77	75.40	69.98	71.42
Min/Max (−)	10	68.54	72.89	63.97	62.94	67.08	72.06	60.25	60.11
	50	66.98	71.46	60.91	61.67	66.01	70.88	58.69	59.42
	100	63.85	71.27	60.98	61.80	62.18	70.65	59.41	58.55
	500	63.37	70.91	58.97	62.46	61.59	70.16	56.15	59.60
	1000	64.92	71.35	58.53	62.78	63.24	70.56	55.06	60.09
Min/Max (+)	10	79.70	76.68	69.40	74.69	79.18	76.09	66.15	72.73
	50	86.95	76.59	73.78	74.38	86.76	76.22	70.66	72.96
	100	83.17	76.08	73.46	74.81	82.76	75.65	70.44	73.07
	500	81.09	75.09	72.33	72.33	80.57	74.51	69.33	70.19
	1000	81.43	75.48	71.97	72.77	80.48	74.90	68.98	70.43

In terms of accuracy, the percentage of improvement depends on the applied algorithm: between 0.64% and 17.58% for Hoeffding tree, between 0.05% and 1.05% for naive Bayes, and between 0.06% and 8.22% for the rule-based classifier. Taking all preprocessed data streams as reference, both HT and kNN 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 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 time can be mostly attributed to the selected algorithm, CEP preprocessing only requiring between 0.1 and 0.2 s to process all instances. Almost no difference is observed when increasing the window size, with the exception of the enriched stream with minimum and maximum values. Using kNN increases learning time up to 30 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 0.4 s to conclude. 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, the rest of 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.

Finally, 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 10% (*min/max(+)*). Furthermore, the highest gain observed in the rule-based classifier (8.22%, *avg(+)*) corresponds to a model with 25 rules instead of the 36 rules originally required. The most discriminatory temporal features, i.e., those most frequently appearing in the decision models, are the average and the maximum electricity prices in New South Wales.

5.3 Experiment 2: Integrating categories

5.3.1 Experiment description

Airports constantly schedule and control flights with the aim of providing good service to travellers. Predicting delays and, most importantly, identifying their causes might alleviate inconveniences to users. To address this problem from a ML perspective, the airlines dataset[10] 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 three numerical features: flight identification number (omitted in ML studies), elapsed time and 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 generalise decision models. Indeed, a default execution of the Hoeffding tree returns a decision tree with 8,582 nodes and 8,518 leaves, but only four levels of depth. This means that decisions are mostly made on a case-by-case basis, i.e., first splitting by departure airport, then checking the airline and finally deciding depending on the arrival airport. Even though categorical information seems to be highly relevant for learning, this type of decision model is neither manageable nor reusable. 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 understandable decision models.

[10] `https://www.openml.org/d/1169` (accessed September 17, 2019)

5.3.2 Definition of CEP rules

Two types of rules are defined in this experiment. Firstly, we apply the `group by` clause to aggregate information from the flights according to each categorical feature, i.e., `airportTo`, `airportFrom`, `airline`, and `dayOfWeek`. Listing 3 shows the syntax of the template rule, which selects the original numerical features, i.e., `time` and `length`, as well as the class label (`delay`). Notice that the day of the week is also retrieved, as this feature contains a small number of categories. 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.

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

Listing 3 Rule that computes the number of flight delays by category.

The previous rule groups instances by the category of one feature only, thus allowing us to study the influence of each categorical attribute. Although more than one category could be included 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 detailed in Listing 4 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 by the subscriber.

```
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
```

Listing 4 Rule that computes the number of flight delays for all attributes.

5.3.3 Results

The five data streams produced by CEP rules are used to train the four classifiers under the same conditions than the previous experiment. Table 5 shows

Table 5 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*	*HT*	*kNN*	*NB*	*RC*
Original		*65.08*	*67.15*	*64.55*	*60.87*	*62.77*	*65.58*	*62.16*	*59.74*
Airport from	10	92.90	91.87	90.51	69.12	93.58	92.36	90.80	66.02
	50	83.12	79.72	65.64	64.24	84.68	80.14	62.90	61.35
	100	75.11	70.88	62.28	60.10	76.64	70.59	59.33	56.89
	500	64.46	61.78	58.84	61.22	63.60	60.06	55.56	59.30
	1000	62.42	60.84	58.61	63.05	60.84	58.94	55.39	62.65
Airport to	10	93.85	93.42	93.82	70.62	94.36	93.82	94.39	67.55
	50	84.41	81.93	70.08	72.89	85.45	82.39	67.98	71.75
	100	76.42	72.57	64.96	66.19	77.76	72.44	62.52	65.01
	500	64.25	61.52	58.99	63.04	63.29	59.78	55.77	62.22
	1000	62.28	60.30	58.25	60.89	60.63	58.33	54.88	59.03
Day of week	10	68.24	64.97	67.97	64.00	67.07	63.44	67.14	62.59
	50	62.50	59.41	62.05	61.55	61.36	57.41	61.49	60.52
	100	62.26	59.74	61.59	61.46	61.09	57.77	61.19	60.14
	500	61.68	59.52	60.83	61.14	60.61	57.51	60.43	59.69
	1000	61.56	59.51	60.55	61.10	60.49	57.52	60.17	59.80
Airline	10	74.99	71.03	65.67	70.30	75.97	70.74	62.82	70.01
	50	69.54	66.03	63.83	69.23	69.37	64.69	60.58	69.59
	100	67.07	64.71	64.38	65.02	65.63	63.13	61.22	63.86
	500	65.57	63.98	64.22	64.39	63.42	62.24	61.05	62.70
	1000	65.40	63.91	64.07	64.31	63.13	62.13	60.84	62.41
4 counters	10	**99.65**	**95.88**	**94.44**	**96.33**	**99.67**	**96.25**	**94.98**	**95.98**
	50	97.22	88.80	77.86	82.71	97.48	89.15	77.16	81.07
	100	91.88	78.36	72.63	86.72	92.67	77.93	71.76	86.57
	500	74.17	66.92	66.97	71.23	74.50	65.46	65.82	71.04
	1000	70.44	65.78	66.27	68.00	70.04	64.16	65.05	67.28

the accuracy and F_1 values obtained in each case. As a baseline, the first row indicates the prediction results for the original data stream.

At the sight of the high number of shaded cells, relying on delay information of flights sharing certain characteristics leads to a substantial increase of classification performance for all algorithms. Considering either the arrival or the departure airport in a short-term window makes a significant contribution, achieving more than 90% of accuracy with three out of the four algorithms. Slightly better results are obtained with the departure airport, suggesting that more delays are explained by 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 5% — is relatively low compared to airport attributes — up to 44% —. Focusing on the airline, improvements increase up to 15%. Furthermore, both accuracy and F_1 results are less dependant on the configured window size. From these results, it is inferred that flights operated by a specific airline are systematically delayed or not, no matter 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 ten, all algorithms achieve an accuracy greater than 94%, representing an average percentage of improvement of 50% 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. Hoeffding tree stands out as the best method, nearly reaching perfect classification. Furthermore, gains are now more generalised across window sizes, kNN being the sole algorithm for which improvement is not possible in two cases. With the exception of kNN, the one-tailed Wilcoxon test confirms that algorithms performs better when the four delay counters replace categorical features in the input data stream.

In terms of preprocessing time, those CEP rules that group delays by one feature run in less than 0.8 s, not being affected by the window size. However, the rule producing the four counters requires between 1 and 17 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 3 and 2 s, respectively. Both algorithms become faster after transforming the data stream, requiring less than 1 s to conclude when one counter is applied. Notice that the total execution time is inferior to that of the original stream even if preprocessing time is added. In contrast, the rule-based classifier, which initially required around 11 s, suffers from the presence of numerical attributes. Now it takes several minutes to compute the internal statistics[11] to decide which values increase the predictive performance the most. Lastly, 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 a couple of minutes to execute in all cases, including training with the original data stream.

Regarding memory requirements, it is worth mentioning that Hoeffding tree, kNN and naive Bayes consume less memory after preprocessing. 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.

Finally, differences regarding the characteristics of the decision models should be highlighted. The structure of the decision tree — four depth levels, 8,582 nodes and 8,518 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 counters are the day of week, 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

[11] Notice that this method temporarily stores such statistics in a text file.

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.

5.4 Experiment 3: Dealing with missing values

5.4.1 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 CEP rules, as well as to analyse their influence in the prediction capabilities of classification algorithms. A common practice to study the performance of imputation methods is to inject missing values following different configurations [2].

For this experiment, we consider a dataset that includes information from light, temperature, humidity and CO2 sensors,[12] whose measurements are useful to detect room occupancy [10]. The two samples distributed by the authors are joined to create the input data stream, with 20,560 instances in total. Missing values are inserted following two strategies: random distribution and periodical sequences of missing values. In both cases, we vary the amount of missing values to study its effect on the learning process. 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.

5.4.2 Definition of CEP rules

For this experiment, CEP rules should act differently if the incoming instance contains missing values or not. Complete instances can be detected by means of the rule shown in Listing 5. 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. The instance is simply passed to the learning algorithm.

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

Listing 5 Rule that captures complete sensor samples.

[12] `https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+` (accessed September 17, 2019)

If only the previous rule is registered in the CEP engine, instances with missing values will be simply discarded. With the purpose of filling those instances, a second rule should be defined. A possibility is to estimate the missing value considering previous values in the temporal series. We implement an external function to adjust each attribute distribution using the least squares method. When a missing value is detected, the next value in the fitting curve is returned. 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. Listing 6 shows the resulting rule. Notice that the subscriber associated to the rule managing complete instances has to be slightly modified to provide the estimator with samples. In addition, a third rule is periodically triggered to clean out-of-date samples, with the intention of building most accurate fitting curves.

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

Listing 6 Rule that generates an instance by estimating missing values.

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 7). As this operator is window-defined, `keepall` 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.

```
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)
```

Listing 7 Rule that generates an instance by replicating the last available measurement.

5.4.3 Results

As in previous experiments, we train the four classification algorithms with all the data streams produced by CEP rules. Tables 6 and 7 collect the results for data streams with randomly injected missing values (MV) and with missing sequences, respectively. Due to space limitations, the results for estimation and average methods correspond to the best window size (30). The first row in Table 6 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'). 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 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. NB slightly decreases in 8 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 6, 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, superior to 90%. As it might 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 big difference in the final accuracy achieved by the methods (see 'with MV' in Table 7). Although all algorithms have time to recover after receiving a missing sequence, a high rate of such sequences, e.g., 15 missing samples every 100 samples ({100,15}), degrade their accuracy up to 10% (from 98.92 to 86.82 for Hoeffding tree). Again, removing missing values becomes a competitive solution,

Table 6 Accuracy and F_1 results for occupancy prediction (missing values at random). Figures expressed as percentage, higher values being preferred.

	% MV	**Accuracy** *HT*	*kNN*	*NB*	*RC*	F_1 *HT*	*kNN*	*NB*	*RC*
Original		*98.92*	*98.57*	*96.04*	*90.84*	*99.03*	*98.06*	*96.76*	*84.75*
With MV	5	95.23	96.69	95.06	86.75	96.57	96.51	94.54	83.91
	10	91.69	94.87	94.18	88.86	94.30	95.14	92.48	81.45
	25	81.94	90.23	91.72	85.40	88.00	91.47	86.90	79.99
	50	69.04	83.83	88.54	77.21	79.71	86.42	78.88	64.37
Remove	5	**98.94**	98.57	96.02	91.45	98.97	98.04	**96.77**	85.93
	10	98.48	98.59	96.00	87.96	98.73	98.09	96.74	81.83
	25	98.16	98.44	95.71	86.71	98.43	97.94	96.52	79.89
	50	98.82	98.51	**96.14**	89.34	98.86	98.19	**96.77**	89.63
Estimate	5	98.25	98.51	95.72	87.97	98.45	97.99	96.43	78.44
	10	98.10	98.49	95.57	87.55	98.22	97.99	96.34	84.01
	25	98.37	98.24	95.00	91.67	98.21	97.61	95.79	85.93
	50	98.06	98.30	94.40	89.75	97.63	97.69	95.10	82.10
Average	5	98.91	98.59	96.01	92.72	98.95	98.09	**96.77**	**93.95**
	10	98.89	**98.61**	96.02	91.33	98.92	98.10	96.74	91.52
	25	98.74	98.59	95.88	**93.72**	98.55	98.07	96.65	93.27
	50	98.63	98.74	95.80	93.57	98.62	**98.29**	96.49	92.98
Last	5	98.93	98.59	96.04	90.84	**99.04**	98.10	96.75	84.75
	10	98.92	98.54	96.04	90.36	99.03	98.03	96.75	84.01
	25	98.88	98.52	95.78	90.85	98.91	98.04	96.54	89.87
	50	98.86	98.46	95.99	90.85	98.87	97.97	96.61	84.77

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.

Some differences arise regarding preprocessing time. Removal (0.05 s) and estimation (between 0.05 and 0.06 s) are the faster methods, followed by average (between 0.08 and 0.09 s) and last (up to 6 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.

6 Discussion

This section provides additional insights from the experimentation. We also discuss the strengths and limitations that we have observed when using MOA and CEP for data stream preprocessing.

Table 7 Accuracy and F_1 results for occupancy prediction (sequences of missing values). Figures expressed as percentage, higher values being preferred.

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

6.1 Experimental findings

To contextualise our experimental results, we have inspected current milestones for the electricity and airlines datasets in OpenML, a platform where researchers can register the outcomes of their ML algorithms. We aim to know whether relatively simple classification algorithms with default parameters, but feeded with preprocessed streams, are competitive compared to state-of-the-art techniques. Looking at the executions collected for the electricity dataset,[13] the best method so far is Ada boost, an ensemble classifier based on decision trees, which achieves 95% of accuracy. Considering that ensemble methods train multiple classifiers, the combination of CEP processing and Hoeffding tree is a competitive alternative for this dataset, since 93% of accuracy is obtained in less than 1 s. Notice that some of the best methods registered in OpenML require more than 5 s to conclude, and might have been trained following an offline approach.

[13] `https://www.openml.org/t/219` (accessed September 17, 2019)

Evaluations of the airlines dataset in OpenML[14] confirm that it is a challenging dataset, since the best accuracy value currently registered is 67.77%. This result corresponds to the MOA implementation of OzaBag, another ensemble classifier. MOA implementations of kNN, naive Bayes and Hoeffding tree have also been tried with different parameter settings, but all of them learn from the original dataset. Therefore, the choice of the algorithm and its parametrisation seems not to be so relevant compared to producing a high-quality data stream for learning.

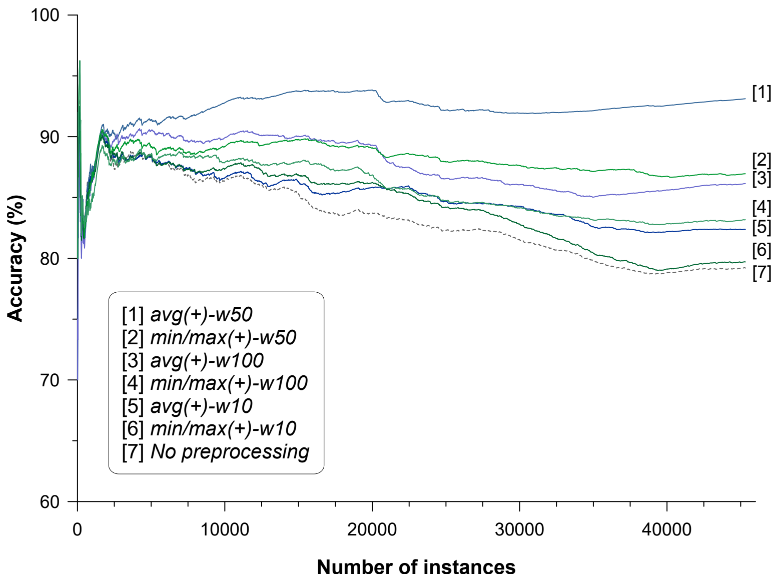

Fig. 2 Learning curve of a Hoeffding tree for electricity data stream enriched with temporal features.

However, not all kinds of preprocessing might work for this dataset. For instance, reported accuracy values after discretisation are not higher than 66% for Hoeffding tree and naive Bayes in [35]. These results represent a minor improvement with respect to the values we obtained for the original data stream under our experimental conditions (65.08% and 64.55%, respectively). This suggest that it is rather difficult to improve the accuracy just by modifying numerical features. Indeed, we found that neither flight duration nor distance did appear in the inferred decision tree, which strongly relies on categorical features. The preprocessing proposed here for this data stream allows reducing the presence of categorical information yet keeping its predictive power. As a result, the overall accuracy is increased up to 99% and other features acquire relevance in the decision models. This fact illustrates the need of care-

[14] `https://www.openml.org/t/7275` (accessed September 17, 2019)

fully analysing the problem domain and the behaviour of classifiers to take the most of preprocessing methods. In this respect, counting on languages and tools specific to data stream processing can significantly improve these kinds of analyses, and the effect that different preprocessing alternatives can have on the system.

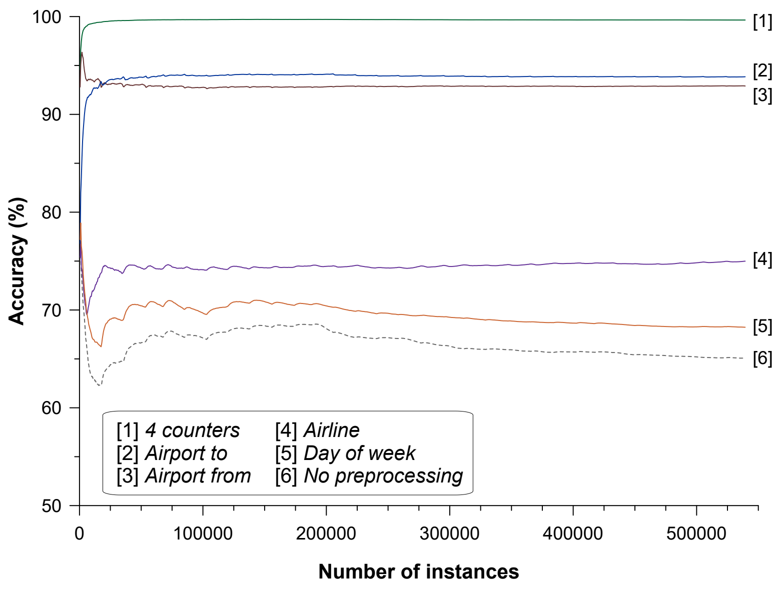

Fig. 3 Learning curve of a Hoeffding tree for airlines data stream with delay counters (window size=10).

It is also interesting to analyse the positive influence of the added temporal features during the learning phase. Figures 2 and 3 show how the accuracy of a Hoeffding tree evolves as the data stream is processed. For clarity, only the three best window sizes are depicted in Figure 2. In this plot, the learning curve when no preprocessing is performed (dashed line) has a decreasing trend that is mitigated when temporal features are included. By computing the average price and demand in the last 50 instances, the algorithm is even able to continue increasing its accuracy at the end of the stream. Regarding the airlines data stream (see Figure 3), grouping the flight delays by either departure or arrival airport allow reaching high accuracy rates quickly and, more importantly, maintain them quite constant afterwards. The good results obtained when simultaneously applying the four counters are visible since the very beginning, thus it is a robust configuration for online learning.

Table 8 Main characteristics of MOA, MOAReduction and CEP w.r.t. data preprocessing.

	MOA	**MOAReduction**	**CEP**
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, CSV	ARFF	ARFF, CSV
Preprocessing tasks			
Data preparation	replace MV add noise		replace MV data transformation
Data reduction	attribute selection	discretisation instance selection feature selection	attribute selection feature generation

6.2 Strengths and limitations of MOA and CEP preprocessing

This section provides an overview of the strengths and limitations of MOA, MOAReduction and CEP, which are summarised in Table 8. As each framework imposes its own processing language, some differences arise regarding how they operate with data streams. In this sense, CEP 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 CEP. These functionalities are not currently supported by MOA, meaning that the user would have to implement them from scratch using Java mechanisms, e.g., queues to simulate windows. The same applies to MOAReduction, though some of its algorithms can be parameterised to specify the frequency of application, a sort of window mechanism.

Another advantage of CEP 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 CEP, since each instance has to be constructed from events implemented as user-defined Java classes. Creating events and converting them to instances can be costly compared to directly handling instances in MOA. Indeed, our preliminary experiments indicate that MOA is few seconds faster than CEP rules when replacing missing values, the only experiment for which MOA provides an equivalent filter.

Regarding input and output formats, MOA and MOAReduction support ARRF data types, i.e., numerical, nominal, string and date, whereas CEP allows objects and primitive types to be part of the event. In order to use our

DM component — 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. CSV, another usual format in data analysis, is restricted to clustering in MOA and also presents some limitations in CEP with respect to supported types.

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 [36]. 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 specialised 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 choose 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 discretisation methods in MOAReduction are the only ones that alter numerical features. As for CEP, 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.

7 Threats to validity

This section presents the threats to internal and external validity, and how any possible bias is mitigated.

Internal validity It refers to the aspects of the solution that ensure the causality of the outcomes. A first threat related to how CEP rules are built is the choice of a proper window size. Although our experimental design considers several values for comparison, better values might exist. The selection of ML algorithms constitutes another threat. We opted for publicly available implementations that have previously been considered in the literature [34,35]. Furthermore, they are representative of different categories of algorithms, allowing us to analyse how their internal procedures and decision models are influenced by the preprocessed streams. The use of default parameters might represent an internal threat too. As we are mostly interested in the relative performance, i.e., before and after applying preprocessing, the default configuration serves our purpose. Moreover, default parameters provide a good baseline according to the results reported in other preprocessing studies and OpenML, as discussed in Section 6.1. Nonetheless, fine-tuned algorithms might achieve better accuracy levels for the original data streams. This may cause a reduction in the improvement attributed to CEP preprocessing, but at the cost of important efforts devoted to parameter tuning.

External validity It mainly concerns the generalisation of the experimental results. Three experiments are conducted to show the applicability of CEP in different preprocessing scenarios. For all of them, several CEP rules are proposed to illustrate the alternatives CEP offers in terms of operators and clauses. The possibilities of applying CEP rules to other preprocessing tasks should be further investigated in the future, though some limitations might exist as pointed out in Section 6.2. For each experiment, a different dataset is used in order to prove that our approach is not limited to a particular application domain. The datasets present different characteristics regarding the number and types of features, as well as number of instances. Some of these datasets often appear in stream data mining studies [32,36], thus being representative of flows of temporal information that should be processed in real time, e.g., sensor data. Additional datasets are needed to validate our approach in other domains.

8 Concluding remarks

This paper proposes the use of complex event processing as a novel approach to address data preprocessing in the context of stream data mining. Handling raw data as events, the fast-processing CEP engine is able to transform, enrich or curate incoming flows of information making them ready for mining. Such treatments are expressed by means of rules, whose SQL-like syntax helps domain specialists to code them. The analysis of CEP functionalities has revealed a wide range of operators and clauses that fit to different preprocessing purposes, and that can be complemented with the definition of windows and user-coded procedures. Our solution is provided as a Java system that connects Esper, a CEP engine, with MOA, a library for online data mining, thus making it possible to perform preprocessing and learning in one single step.

Three experiments illustrate how CEP gives support to data transformation, feature generation and missing value replacement for diverse application domains. Simple yet powerful CEP rules have proven to be able to generate high quality data streams, improving the accuracy of supervised algorithms. In particular, temporal information computed by CEP windows allows making better electricity market predictions, while compressing categorical features has revealed as a key factor to understand flight delays. 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 enable them to reach highly competitive results compared to publicly available achievements. Consequently, parameter tuning or algorithm selection might turn less relevant. In this sense, we believe our solution helps overcome some of the issues currently limiting the adoption of data analytics by organisations, such as perceived difficulty in its methods, availability of tools, and data quality and integration [3].

In the future, we plan to apply CEP to other preprocessing tasks, as well as comparing it to other stream processing engines like Flink or Azure and stream mining solutions like SAMOA [30] and scikit-Multiflow [29]. We hypothesise that the event-based nature of CEP might also be exploited to detect concept drift. Finally, empirical studies could be designed to analyse whether using CEP makes data stream preprocessing a lighter task.

Acknowledgements This work was supported by the Spanish Ministry of Economy and Competitiveness under projects TIN2014-52034-R and PGC2018-094905-B-I00.

References

1. Affetti, L., Tommasini, R., Margara, A., Cugola, G., Della Valle, E.: Defining the execution semantics of stream processing engines. Journal of Big Data **4**(1), 12 (2017). DOI 10.1186/s40537-017-0072-9
2. Andiojaya, A., Demirhan, H.: A bagging algorithm for the imputation of missing values in time series. Expert Systems with Applications **129**, 10–26 (2019). DOI 10.1016/j.eswa.2019.03.044
3. Baig, M.I., Shuib, L., Yadegaridehkordi, E.: Big data adoption: State of the art and research challenges. Information Processing & Management **56**(6), 102,095 (2019). DOI 10.1016/j.ipm.2019.102095
4. Barddal, J.P., Enembreck, F., Gomes, H.M., Bifet, A., Pfahringer, B.: Merit-guided dynamic feature selection filter for data streams. Expert Systems with Applications **116**, 227–242 (2019). DOI 10.1016/j.eswa.2018.09.031
5. Barddal, J.P., Gomes, H.M., Enembreck, F., Pfahringer, B.: A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. Journal of Systems and Software **127**, 278–294 (2017). DOI 10.1016/j.jss.2016.07.005
6. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis. Journal of Machine Learning Research **11**, 1601–1604 (2010)
7. Bollegala, D.: Dynamic feature scaling for online learning of binary classifiers. Knowledge-Based Systems **129**, 97–105 (2017). DOI 10.1016/j.knosys.2017.05.010
8. Bolon-Canedo, V., Fernández-Francos, D., Peteiro-Barral, D., Alonso-Betanzos, A., Guijarro-Berdiñas, B., Sánchez-Maroño, N.: A unified pipeline for online feature selection and classification. Expert Systems with Applications **55**, 532–545 (2016). DOI 10.1016/j.eswa.2016.02.035
9. Bruns, R., Dunkel, J., Offel, N.: Learning of complex event processing rules with genetic programming. Expert Systems with Applications **129**, 186–199 (2019). DOI 10.1016/j.eswa.2019.04.007
10. Candanedo, L.M., Feldheim, V.: Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. Energy and Buildings **112**, 28–39 (2016). DOI 10.1016/j.enbuild.2015.11.071

11. Crone, S., Lessmann, S., Stahlbock, R.: The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. European Journal of Operational Research **173**(3), 781–800 (2006). DOI 10.1016/j.ejor.2005.07.023
12. Cugola, G., Margara, A.: Processing Flows of Information: From Data Stream to Complex Event Processing. ACM Computing Surveys **44**(3), 15:1–15:62 (2012). DOI 10.1145/2187671.2187677
13. Demirhan, H., Renwick, Z.: Missing value imputation for short to mid-term horizontal solar irradiance data. Applied Energy **225**, 998–1012 (2018). DOI 10.1016/j.apenergy.2018.05.054
14. Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Kamp, M., Mock, M.: Issues in complex event processing: Status and prospects in the Big Data era. Journal of Systems and Software **127**, 217–236 (2017). DOI 10.1016/j.jss.2016.06.011
15. Fülöp, L.J., Beszédes, A., Tóth, G., Demeter, H., Vidács, L., Farkas, L.: Predictive Complex Event Processing: A Conceptual Framework for Combining Complex Event Processing and Predictive Analytics. In: Proceedings 5th Balkan Conference in Informatics (BCI), pp. 26–31 (2012). DOI 10.1145/2371316.2371323
16. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: Mining Data Streams: A Review. ACM SIGMOD Record **34**(2), 18–26 (2005). DOI 10.1145/1083784.1083789
17. Gama, J.: Knowledge Discovery from Data Streams, 1st edn. Chapman & Hall/CRC (2010)
18. Gama, J., Kosina, P.: Learning Decision Rules from Data Streams. In: Proceedings of the 22th International Joint Conference on Artificial Intelligence (IJCAI) - Volume II, pp. 1255–1260 (2011). DOI 10.5591/978-1-57735-516-8/IJCAI11-213
19. Gama, J., Pinto, C.: Discretization from Data Streams: Applications to Histograms and Data Mining. In: Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), pp. 662–667. ACM (2006). DOI 10.1145/1141277.1141429
20. García, S., Luengo, J., Herrera, F.: Data Preprocessing in Data Mining. Springer (2014)
21. Han, J., Kamber, M., Pei, J.: Data Preprocessing, 3rd edition edn., chap. 3, pp. 83–124. Morgan Kaufmann (2012). DOI 10.1016/B978-0-12-381479-1.00003-4
22. Hulten, G., Spencer, L., Domingos, P.: Mining Time-changing Data Streams. In: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 97–106 (2001). DOI 10.1145/502512.502529
23. Jin, R., Agrawal, G.: Frequent Pattern Mining in Data Streams, chap. 4, pp. 61–84. Springer US (2007). DOI 10.1007/978-0-387-47534-9_4
24. Kousiouris, G., Akbar, A., Sancho, J., Ta-shma, P., Psychas, A., Kyriazis, D., Varvarigou, T.: 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 (2018). DOI 10.1016/j.future.2017.07.026
25. Lee, O.J., Jung, J.E.: Sequence Clustering-based Automated Rule Generation for Adaptive Complex Event Processing. Future Generation Computer Systems **66**, 100–109 (2017). DOI 10.1016/j.future.2016.02.011
26. Loreti, D., Chesani, F., Mello, P., Roffia, L., Antoniazzi, F., Cinotti, T.S., Paolini, G., Masotti, D., Costanzo, A.: Complex reactive event processing for assisted living: The Habitat project case study. Expert Systems with Applications **126**, 200–217 (2019). DOI 10.1016/j.eswa.2019.02.025
27. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2001)
28. Margara, A., Cugola, G., Tamburrelli, G.: 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 (2014). DOI 10.1145/2611286.2611289
29. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-Multiflow: A Multi-output Streaming Framework. Journal of Machine Learning Research **19**(72), 1–5 (2018). URL `http://jmlr.org/papers/v19/18-251.html`
30. Morales, G.D.F., Bifet, A.: SAMOA: Scalable Advanced Massive Online Analysis. Journal of Machine Learning Research **16**, 149–153 (2015). URL `http://jmlr.org/papers/v16/morales15a.html`

31. Mousheimish, R., Taher, Y., Zeitouni, K.: 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 (2017). DOI 10.1145/3093742.3093917
32. Nguyen, H.L., Woon, Y.K., Ng, W.K.: A survey on data stream clustering and classification. Knowledge and Information Systems **45**(3), 535–569 (2015). DOI 10.1007/s10115-014-0808-1
33. Ölmezogullari, E., Ari, I.: Online Association Rule Mining over Fast Data. In: Proceedings of the IEEE International Congress on Big Data, pp. 110–117 (2013). DOI 10.1109/BigData.Congress.2013.77
34. Prasad, B., Agarwal, S.: Stream data mining: Platforms, algorithms, performance evaluators and research trends. Int. Journal of Database Theory and Application **9**(9), 201–218 (2016)
35. Ramírez-Gallego, S., García, S., Herrera, F.: Online entropy-based discretization for data streaming classification. Future Generation Computer Systems **86**, 59–70 (2018). DOI 10.1016/j.future.2018.03.008
36. Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M., Herrera, F.: A survey on data preprocessing for data stream mining: Current status and future directions. Neurocomputing **239**, 39–57 (2017). DOI 10.1016/j.neucom.2017.01.078
37. Saggi, M.K., Jain, S.: A survey towards an integration of big data analytics to big insights for value-creation. Information Processing & Management **54**(5), 758–790 (2018). DOI 10.1016/j.ipm.2018.01.010
38. Sun, A.Y., Zhong, Z., Jeong, H., Yang, Q.: Building complex event processing capability for intelligent environmental monitoring. Environmental Modelling & Software **116**, 1–6 (2019). DOI 10.1016/j.envsoft.2019.02.015
39. Tidke, B., Mehta, R.G., Dhanani, J.: Real-Time Bigdata Analytics: A Stream Data Mining Approach. In: Proc. 5th International Conference on Recent Findings in Intelligent Computing Techniques, pp. 345–351 (2018). DOI 10.1007/978-981-10-8636-6_36
40. Uysal, A.K., Gunal, S.: The impact of preprocessing on text classification. Information Processing & Management **50**(1), 104–112 (2014). DOI 10.1016/j.ipm.2013.08.006
41. Wang, Y., Gao, H., Chen, G.: Predictive complex event processing based on evolving Bayesian networks. Pattern Recognition Letters **105**, 207–216 (2018). DOI 10.1016/j.patrec.2017.05.008
42. Zhang, S., Zhang, C., Yang, Q.: Data preparation for data mining. Applied Artificial Intelligence **17**(5-6), 375–381 (2003). DOI 10.1080/713827180