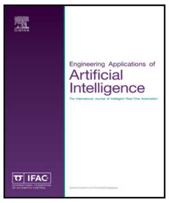
Engineering Applications of Artificial Intelligence 126 (2023) 107028

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Trace encoding in process mining: A survey and benchmarking

Gabriel M. Tavares[a], Rafael S. Oyamada[b,*], Sylvio Barbon Junior[c], Paolo Ceravolo[b]

[a] LMU Munich, Oettingenstraße 67, Munich, 80538, Germany
[b] University of Milan, Via Giovanni Celoria, 20, Milan, 20133, Italy
[c] University of Trieste, Piazzale Europa, 1, Trieste, 34127, Italy

ARTICLE INFO

Keywords:
Encoding methods
Process mining
Anomaly detection

ABSTRACT

Encoding methods are employed across several process mining tasks, including predictive process monitoring, anomalous case detection, trace clustering, etc. These methods are usually performed as preprocessing steps and are responsible for mapping complex event data information into a numerical feature space. Most papers choose existing encoding methods arbitrarily or employ a strategy based on expert domain knowledge. Moreover, existing methods are employed by using their default parameters without evaluating other options. This practice can lead to several drawbacks, such as suboptimal performance and unfair comparisons with the state-of-the-art. Therefore, this work aims at providing a comprehensive survey and benchmark on event log encoding by comparing 27 methods, from different natures, in terms of expressivity, scalability, correlation, and domain agnosticism. To the best of our knowledge, this is the most comprehensive study so far focusing on trace encoding in process mining. It contributes to maturing awareness about the role of trace encoding in process mining pipelines and sheds light on issues, concerns, and future research directions regarding the use of encoding methods to bridge the gap between machine learning models and process mining.

1. Introduction

Encoding methods are responsible for transforming complex information into a representative feature space, i.e., mapping data from one space to another. In process mining (PM), several tasks (e.g., predictive process monitoring, trace clustering, anomaly detection, etc.) must encode data as a step of the pipeline before feeding algorithms. This step is crucial to correctly account for the goals of a user. For instance, if a problem demands a solution where interpretability and explainability are needed, the data should be encoded by methods that tend to accomplish those objectives. On the other hand, if the most essential requirements are space or time complexity, the user should agree to lose part of the previous benefits to match these ones.

In PM literature, most of the efforts have been dedicated to designing new algorithms and analytical methods but little attention has been given to the impact of encoding methods across the existing tasks. For instance, in predictive process monitoring Evermann et al. (2016) used the *word embedding* method to map the cases of an event log into real-valued vectors, whereas Tax et al. (2017) used *one-hot* encoding. A custom function is adopted by Hompes et al. (2015), whereas the *count2vec* (occurrence frequencies of activities) is employed by Appice and Malerba (2016). Thus, a researcher interested in comparing the results of these works is in front of an uncontrolled factor, as the impact of encoding is not documented and the methods used are different. Moreover, in this work, we emphasize that very few alternative encoding methods have been employed by the community and demonstrate that arbitrarily encoding data might bring suboptimal results and misalignment with the user's goals. We believe that a better understanding of the effect of encoding methods, according to the datasets' characteristics, is decisive in developing more interpretable, explainable, robust, and accurate PM solutions.

Using anomaly detection as a case study, we extend the results of our previous paper (Barbon Junior et al., 2020) by considering several aspects. Further, the encoding procedure in this work is always performed in an unsupervised fashion (i.e. each trace is encoded regardless of label availability), though it can be employed either for supervised or unsupervised tasks in process mining. Thus, we first increase the number of encoding methods with respect to our previous work and classify them according to their underlying characteristics. Second, we include more datasets, considering more types of anomalies,

* Corresponding author.
E-mail addresses: tavares@dbs.ifi.lmu.de (G.M. Tavares), rafael.oyamada@unimi.it (R.S. Oyamada), sylvio.barbonjunior@units.it (S. Barbon Junior), paolo.ceravolo@unimi.it (P. Ceravolo).

https://doi.org/10.1016/j.engappai.2023.107028
Received 31 December 2022; Received in revised form 17 July 2023; Accepted 22 August 2023
Available online 19 September 2023

in order to increase the space of characteristics and achieve a better understanding of how each encoding method behaves according to the data properties. Third, we employ evaluation criteria that are valuable for PM practitioners and can support the choice of a suitable encoding method according to their goals. Lastly, we provide a review of encoding methods from different literature and discuss how they can be employed across popular PM tasks: predictive monitoring, trace clustering, and anomaly detection.

More specifically, we first highlight how difficult it is not just to choose a suitable encoding method but also to configure its parameters. Subsequently, we perform an extensive experimental evaluation of 27 encoding methods with different parameters over 420 synthetic event logs. We also discuss how current PM literature is limiting their experiments by not considering the impact that encoding methods have in any problem domain regarding different process mining tasks. Thus, we discuss our results and focus the contribution of our work on answering the following research questions:

1. How expressive is an encoding method for separating the problems' classes?
2. What is the demand of time and memory to reach a suitable encoding method?
3. Is there any correlation between the encoding method and the performance achieved by algorithms in PM tasks?
4. How generic encoding methods are, i.e., can an encoding method be applied to different PM tasks?

We answer these questions by applying specific evaluation metrics according to different criteria. Through an in-depth analysis, we consider the criteria *expressivity*, which measures how effective an encoding method is at mapping data patterns from the event data nature to a new *n*-dimensional space by minimizing the information loss and preserving the underlying event log properties; *scalability*, which measures computational costs in terms of elapsed time and memory usage of encoding methods; *correlation power*, which maps how impactful an encoding method is with respect to the final performance of a given algorithm that is being employed to handle a specific PM task; and the *domain agnosticism*, which considers if the encoding method depends on or not the PM task. We demonstrate through our extensive experimental evaluation how difficult it might be to choose a suitable encoding method since each of the evaluated metrics has a different best-performing method. Thus, the main contributions of this work include:

- A review of encoding methods in PM and encoding methods from different areas of the literature that can be employed in PM tasks.
- The application of new combinations of evaluation metrics to measure the quality of encoding methods in PM tasks according to different criteria.
- A deep experimental evaluation of several encoding methods never employed before in PM.
- A discussion of insights into future research on encoding for PM.

We organize the presentation of our work as follows. First, in Section 2 we define the problem of choosing the right encoding method and its parameters. Section 3 discusses how related works in PM have been employing encoding methods to develop their solutions. In Section 4 we provide the necessary background to understand this work. In Section 5 we first discuss how encoding methods were applied for different PM tasks in the literature and situate encoding methods from different areas of the literature. Subsequently, we organize the encoding methods found by families of algorithms and describe each method. Section 6 describes the employed methodology to implement our experimental evaluation and Section 7 presents the carried experiments and results. Then, in Section 8 we develop a discussion focused on parameter impact and possible aggregation methods. In Section 9 we discuss the main insights obtained in this work and provide future directions. We conclude our discussion in Section 10.

2. Problem definition

In this section, we address the problem of how arbitrarily employing encoding methods in PM tasks leads to sub-optimal performance and results in unfair evaluations. Due to the wide range of encoding methods available nowadays, choosing a suitable one given a problem is challenging. This can be seen in the current literature, across different research fields, with several automated solutions that have been proposed to decrease human intervention in the design of algorithms and data science pipelines (Olson and Moore, 2016; Kim and Teh, 2018; Feurer et al., 2019). In PM, we believe this is even more challenging due to the nature of event logs, where events can be described by both numerical and categorical attributes, are aggregated by cases, and are constrained by the control-flow of the process. For example, the availability of a given amount of resources may be a precondition to observe an event (e.g., the execution of an activity) with dependencies to other preceding or concurrent events. Condensing all this information into a single encoding method is difficult, and, in practical terms, each method can only capture some aspects.

Usually, encoding methods for PM are adapted from other research fields. Simple techniques are often considered, for instance, the *one-hot* encoding scheme (Tax et al., 2017) or frequency-based encoding methods (Francescomarino et al., 2019). To capture the sequential nature of event logs, methods originally proposed in the Natural Language Processing (NLP) community have been employed (Koninck et al., 2018; Tavares and Barbon, 2020). However, while we can take into consideration the similarity between the sequential nature of traces and natural language sentences, there are also differences that must be discussed. For instance, NLP tasks usually handle a very large vocabulary, i.e., the set of unique words or tokens, whereas processes are usually represented by considerably small vocabularies (e.g., the business process activities). As an attempt of capturing additional complexity, graph neural networks have been recently studied in the literature (Venugopal et al., 2021). Convolutional neural networks have also been used for feature extraction (Mehdiyev et al., 2022). Image-like data engineering methods have been introduced by Pasquadibisceglie et al. (2019), Senderovich et al. (2019) and Venugopal et al. (2021). More recently, several pipelines have approached domain-specific encoding methods, which we will further describe in Section 3, that exploit derived features, such as the resource pool discovery algorithm used to encode event resources by Camargo et al. (2019).

In the context of our work, we stress that adopting the right encoding method and selecting optimal parameters can directly impact the final performance of a given task. Moreover, evaluating a new algorithm, e.g., a trace clustering algorithm, by comparing it with other solutions but employing different data inputs (i.e., different encoding steps preceding the clustering), produces an unfair evaluation. Rama-Maneiro et al. (2021) emphasize this problem, highlighting that a given model cannot be compared with another if their implementations consider different feature spaces. A brief example illustrating this issue can be found in Camargo et al. (2019), where the authors are approaching the problem of predictive monitoring. In their evaluation, the authors employ baselines to compare their proposal with existing LSTM architectures, each one based on a different preprocessing procedure. Regarding other predictive monitoring work, word embedding is employed in Camargo et al. (2019), Al-Jebrni et al. (2018) and Taymouri et al. (2021) whereas a traditional *one-hot* encoding is used in Polato et al. (2018), Mauro et al. (2019) and Kratsch et al. (2021), preventing the comparison between these studies. This problem is exacerbated by the fact that the PM community lacks shared benchmarks to be used in algorithm evaluation and comparison.

In order to briefly illustrate the impact of arbitrarily encoding an event log, we demonstrate in Fig. 1 the following scenario. We compare two encoding algorithms, vary the parameter of vector dimensionality, apply them to two datasets with different characteristics, and measure the accuracy achieved by a Random Forest classifier regarding

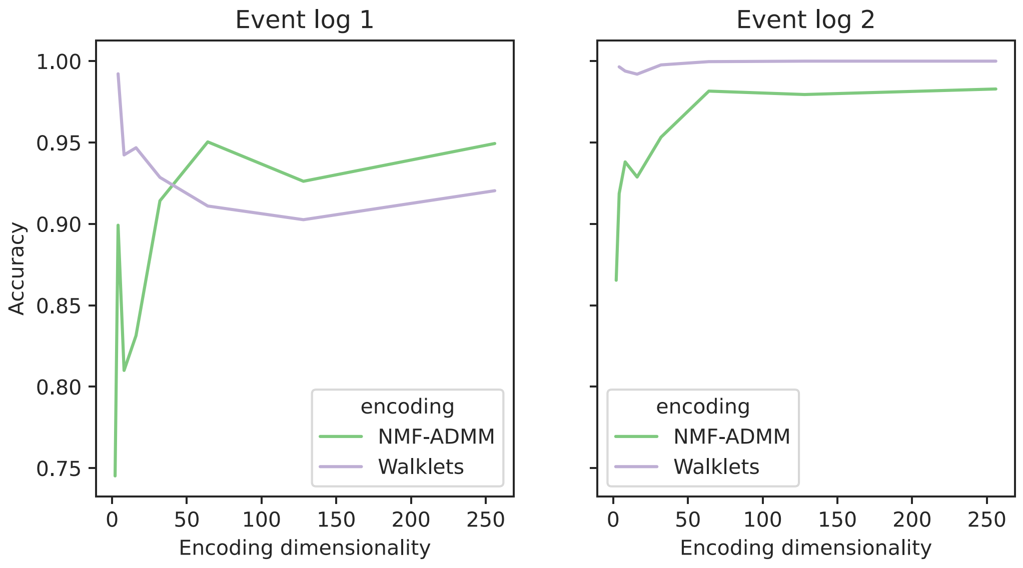


Fig. 1. A brief example of performances achieved for two different datasets regarding the anomaly detection problem. For both datasets, we fixed two graph-based encoding methods with different parametrizations regarding dimensionality.

the anomaly detection problem.[1] The event logs have different cardinalities, different types of anomalies, and different rates of anomaly injection. The first one has $10k$ traces, $444k$ events, and a 20% rate of *insertion* anomaly (a random activity is inserted in the trace), whereas the second has $5k$ traces, $221k$ events, and a 15% rate of *rework* anomaly (an activity is doubled in the trace). As we can see in Fig. 1, for the first event log (Event log 1), encoding the data employing the *Walklets* method performed better than using the *NMF-ADMM* with low dimensionality and worse for medium and high dimensionality. In addition, the latter method presented a high accuracy variation for different dimensionalities. For the second event log (Event log 2), the *Walklets* encoding presented a more stable accuracy, while the *NMF-ADMM* achieved higher accuracy w.r.t. the other dataset but always performed worse than *Walklets*. This is just a brief example to demonstrate that there is no best encoding method for every dataset or default parameterization to apply.

In this work, we attempt to demonstrate in detail how different methods perform on several datasets with distinct characteristics and properties. We first demonstrate through extensive experimental evaluation that each algorithm has distinct performances across different event logs and pipelines. We employ several metrics in order to summarize the overall behavior of each method and focus the general evaluation on expressivity, scalability, correlation, and domain agnosticism, which will be further detailed in Section 6.

3. Related works

Insufficient attention has been given to encoding methods in predictive monitoring tasks, trace clustering, and anomaly detection, despite their significant impact on algorithm performance. Surveys and benchmarks are being conducted in other problem domains to standardize and investigate encoding methods' behaviors in accordance with problem characteristics. For instance, Goyal and Ferrara (2018) surveyed several graph-based embedding methods on different datasets and discussed the main challenges for future research in the field. Regardless of the approached task (e.g. link prediction, node classification, etc.), the authors demonstrate the difficulty of choosing not only the right algorithm but also the right set of parameters (mainly the dimensionality). Several trade-offs must always be taken into consideration, for instance increasing the memory usage to achieve more precision or decreasing the dimensionality to decrease the computation time. Goldberg (2016) covered a wide range of methods to encode textual information. The work focuses on methods based on encoding methods to feed neural network architectures regarding different tasks and also provides historical notes for each category of task.

The aforementioned works usually focus on representational learning, which employs neural networks to learn a high-quality representation (encoding) of data. In the natural language literature, the *word2vec* (Mikolov et al., 2013a,b) can be seen as one of the most important methods for this purpose. From this perspective, several methods derived from it, for instance, Le and Mikolov (2014) and Koninck et al. (2018). The resulting feature vectors representing the original data are also called *embeddings*.

Recently, representational learning has been applied in PM as well. Koninck et al. (2018) propose *act2vec*, *trace2vec*, *log2vec*, and *model2vec*. Each approach derives from existing encoding methods in the literature and leverages the previous level information to enrich the learning. That is, the first level is *act2vec*, which extends the *word2vec* architecture to learn the representation of activities. Subsequently, the *trace2vec* adopts the *doc2vec* concept and jointly learns the representation of activities and traces. The *log2vec* architecture derives from the same idea as *trace2vec* where the log representation is included in the architecture to be jointly learned. Finally, for *model2vec*, the authors extend graph representation learning techniques to represent a process model discovered from the event log. The final architecture also includes all the previous representations to be learned jointly.

In the literature, we also find "*hand-crafted*" methods, which are usually developed by following some expertise domain knowledge. Venugopal et al. (2021) propose the use of graph convolutional networks for predictive monitoring. In their approach, the authors first perform a feature engineering step to handle time features and then transforms each activity in an event log into a matrix $num\_unique\_activities \times num\_time\_features$. Camargo et al. (2019) employ a PM algorithm to encode resources in event logs. In a nutshell, the applied algorithm is able to automatically discover resource pools and, hence, reduce the dimensionality of categorical values by grouping them. In Pasquadibisceglie et al. (2019), the authors propose the use of convolutional neural networks to perform predictive monitoring. Thus, they transform the data into an image-like structure in order to be able to train the neural network. Chiorrini et al. (2022) present a method for feature extraction that can be seen as an encoding method, where

[1] A detailed description of the material and methods is provided in Section 7

seven different features are extracted from each activity given a Petri net. These features aim at capturing local information for the activity with respect to its current case.

Although recent works in predictive process monitoring have explored more alternatives, it is noticeable that works in PM often use a minimal variety of encoding methods. Most papers use naive techniques like *one-hot* encoding. Moreover, other encoding approaches are usually combined with results from feature engineer procedures that handle numerical and time-related information. In recent works, the most common encoding methods for different tasks include the *one-hot* (Tax et al., 2017; Kratsch et al., 2021; Francescomarino et al., 2017; Mauro et al., 2019; Taymouri et al., 2021), counting the frequencies of categorical data (Francescomarino et al., 2019), some type of embedding network (Al-Jebrni et al., 2018; Lin et al., 2019; Camargo et al., 2019; Taymouri et al., 2021), or hand-crafted representations (Camargo et al., 2019; Venugopal et al., 2021; Pasquadibisceglie et al., 2019). Therefore, we motivate our work in order to fulfill this limitation by exploring a wider range of encoding methods.

4. Background notions

PM can be defined as a set of techniques to extract knowledge from event logs (van der Aalst, 2016). The goal is to provide analysis that uses event data to extract process-related insights, i.e., creating solutions that are specifically tailored for business processes and their stakeholders.

Thus, let us first consider Σ a universe of events, i.e., the set of all possible event identifiers. Σ^* denotes the set of all sequences over Σ.

Definition 4.1 (*Event, Attribute*). Events may have various *attributes*, such as `timestamp`, `activity`, `resource`, and others. Let $\mathcal{AN}$ be the set of attribute names. For any event $e \in \Sigma$ and an attribute $n \in \mathcal{AN}$, then $\#_n(e)$ is the value of attribute n for event e. Typically, values are restricted to a domain. For example, $\#_{activity} \in A$, where A is the universe of the legal activities of a business process, e.g. $\{a, b, c, d, e\}$.

With abuse of notation, we refer to the name of the activity of an event $\#_{activity}(e)$ as the event itself. Thus $\langle a, b, d\rangle$ denotes a trace of three subsequent events. An event can also be denoted by its position in the sequence as e_i with e_n the last event of this sequence. A sequence of events composes a *trace* $t \in \Sigma^*$ and it can be defined as follows.

Definition 4.2 (*Trace, Subtrace*). In a *trace* each event appears only once and time is non-decreasing, i.e. for $1 \le i < j \le |t| : t(i) \le t(j)$. A trace can also be denoted as a function generating the corresponding event for each position of its sequence: $t(i \to n) \mapsto \langle e_i, \dots, e_n\rangle$. A subtrace is a sequence $t(i \to j)$ where $0 < i \le j < n$.

Now let C be the *case universe*, that is, the set of all possible identifiers of a business case execution. C is the domain of an attribute $case \in \mathcal{AN}$.

Definition 4.3 (*Case, Event Log*). We denote a case $c_i \in C$ as $\langle a, b, d\rangle_{c_i}$, meaning that all events share the same case identifier. For example, for c_i we have $\#_{case}(e_1) = \#_{case}(e_2) = \#_{case}(e_3)$. An *event log* L is a set of cases $\mathbf{L} \subseteq \Sigma^*$ where each event appears only once in the log, i.e., for any two different cases the intersection of their events is empty.

Furthermore, we leverage the formal definition from Teinemaa et al. (2019) to define how to transform a trace from its process nature to a numerical feature vector. Moreover, in this work, we employ the *aggregation* trace abstraction method described by the authors to average the encoded values from (sub)traces.

Definition 4.4 (*(sub)Trace Encoder*). An encoder $f : t(i \to j) \mapsto \mathcal{X}_1 \times \cdots \times \mathcal{X}_p$ is a function that takes as input a (sub)trace $t(i \to j)$, where $0 < i \le j < n$, and transforms it into a p-dimensional array $\mathcal{X} \subseteq \mathbb{R}^p$.

In the context of this work, we approach the anomaly detection problem to benchmark encoding methods. In general, anomalies are behaviors in data that are different from the usual or normal patterns. Anomaly detection can be approached as a supervised learning problem, where the goal is to train a model that can accurately classify data points as normal or anomalous based on a labeled dataset. Given an event log L consisting of C cases, each case c_i is associated with a binary label y_i, where $y_i = 1$ if c_i is anomalous and $y_i = 0$ if c_i is normal. The goal is to learn a function $f(c)$ that can accurately predict the label y of a new case c based on its attributes.

5. Encoding methods

We perform two reviews: first, on the PM literature by taking into consideration which tasks use encoding methods at some point in their proposals; second, we review the literature on encoding methods in general, classify the found methods into families, and provide a technical description of each of them.

5.1. Brief literature review on encoding in process mining

We perform a brief review aiming for a more generic and intuitive idea of how many PM tasks are solved by employing an encoding method at any point of their proposed pipelines or solutions.

Thus, the online repositories employed for this review were the ACM Digital Library,[2] the IEEE Xplore,[3] and the Scopus.[4] We did not include Google Scholar in order to narrow our search since it usually captures the same papers as the other repositories, plus papers from unknown databases. Moreover, we searched only for works from the last 10 years with respect to the date time this review was performed, i.e., from 2012 to 2022. A base query was defined and it was partially modified according to each PM task: *"process mining" AND ("encoding" OR "encode") AND < task >*, where the keyword *task* might be *"clustering"*, *("predictive monitoring" OR "process monitoring")*, *("anomaly detection" OR "conformance-checking")*. Notice that we are including the terms conformance-checking and anomaly detection interchangeably since anomaly detection can be seen as a sub-task of conformance-checking.

After filtering by including only conference and journal papers, and dropping duplicates, we examined the abstracts of each retrieved document to eliminate irrelevant papers. We achieved a total of 149 papers, where 74 are included as clustering (CLUS), 55 as predictive process monitoring (PPM), and 20 as anomaly detection (AD). We illustrate the number of publications for each task and per year in Fig. 2a and the total publications for each task in Fig. 2b.

Furthermore, at this level, there might be specific targets when encoding data. For instance, event attributes might need to be encoded individually. This is a common setting in predictive process monitoring, where each activity is encoded in order to predict the next one. On the other hand, certain applications (e.g., clustering tasks) might need to encode the complete trace. In this scenario, the trace can be encoded in a straightforward fashion by the employed algorithm or it can be encoded by aggregating the individual event attributes.

Subsequently, we present our second review, this time regarding encoding methods from different literature. We organize the found methods into three different families that are based on: process mining, words (text), and graphs. To the best of our knowledge, most of the methods employed have never been used in PM tasks. Furthermore, to motivate researchers and practitioners to consider these alternative methods more often, we only include in this study methods that have open-source implementations.

[2] https://dl.acm.org/
[3] https://ieeexplore.ieee.org/Xplore/home.jsp
[4] https://www.scopus.com/search

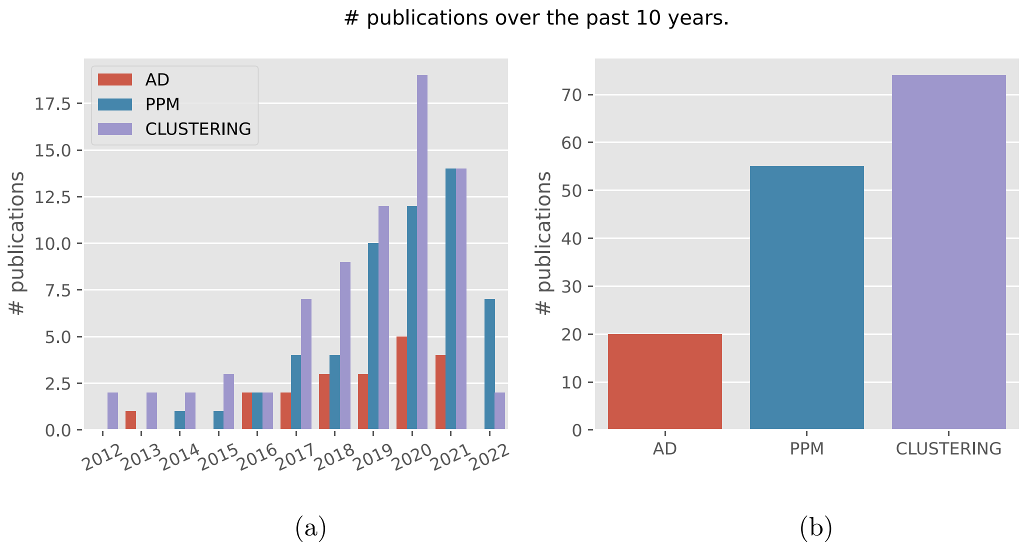


Fig. 2. (a) Number of publications each year. (b) The total number of publications over the past ten years.

5.2. PM-based encoding

Given an event log, we retrieve its respective process model and perform conformance-checking techniques to measure its adherence to the model. Each trace in the event log is evaluated. The results produced are employed as the encoded representation of the trace. The methods considered in our survey are illustrated below.

Token-replay (Berti and van der Aalst, 2019): given a process model, traces are replayed in it to obtain values that measure its conformance. More specifically, the values accumulated at each step are the number of tokens correctly consumed (c), the number of tokens correctly produced (p), the number of missing tokens to execute the event in the next step (m), and the number of unconsumed tokens after the last event execution (r). Thus, the final measure defined by the token-replay method is given by $fitness = \frac{1}{2}(1 - \frac{m}{c}) + \frac{1}{2}(1 - \frac{r}{p})$. All the values produced, $\langle c, p, m, r, fitness \rangle$, are used as the feature vector of a given trace.

Alignment (Carmona et al., 2018): performs a comparison between the process model and a trace and relates the trace to valid execution sequences, i.e., allowed by the model. An alignment is a sequence of moves that can be synchronous, model-dependent, or log-dependent. It is also important to note that more than one alignment between the log and model is possible, and techniques aim at finding the optimal one. The final feature vector is composed of the cost of the alignment, the number of visited states, the number of queued states, the number of traversed arcs, and the fitness value produced.

Log skeleton (Verbeek and de Carvalho, 2018): this technique aims at summarizing activity traces by capturing a set of constraints that apply to activities throughout the log. For example, the R_L^{eq} captures the equivalence relation between two activities, which exists if both activities have the same frequency of occurrence in every trace. On the other hand, the C_L^{df} counts the number of directly-follows occurrences for every pair of activities. Other examples of measures to capture relations include the always-after and never-together; examples of countermeasures include the sum of occurrences of a given activity in the entire log and the min and max numbers of occurrences of an activity in any trace. In the implementation used for this paper, six different constraints are used.

Position profile (Ceravolo et al., 2017): this technique represents an event log through a matrix, where each position refers to the $activity \times position$ regarding all traces. It can be formally defined as a triple $apf = (a, p, f) \in E$, where a is the activity, p is the position of the activity, f is the frequency of occurrence of the given activity, and E is the universe of events.

5.3. Text-inspired encoding

Many solutions used for trace encoding in PM are adapted from methods used in NLP. Exploiting the fact that words in sentences are ordered in sequence and are constrained by dependencies, encoding methods applied to text capture that information. Because traces are composed of sequences of activities the same information appears relevant to characterize them. In particular, in our survey, we consider the following methods.

N-grams (Gasparetto et al., 2022): this method represents a given sequence of elements through sub-sequences of n items. Thus, considering a sequence $\mathbf{s} = \{s_1, \dots, s_i\}$, the *n-grams* representation of these sequences is given by $n - grams = \{(s_1, \dots, s_n), (s_2, \dots, s_{n+1}), \dots (s_{i-n}, \dots, s_i)\}$.

One-hot (Weiss et al., 2015): given a variable containing n different values, the variable is transformed into an array where each unique value is represented as a binary vector with the $i - th$ position set to one and the rest set to zero. Clearly, the dimension of the vector depends on the size n of the unique values in the vector space, easily reaching high dimensional spaces.

CountVectorizer (count2vec) (Weiss et al., 2015): given a collection of categorical documents, this method produces a matrix of token occurrences, where each line in the matrix represents a document and each column a token. The size of the vector space depends on the n unique values in the vector space.

HashVectorizer (hash2vec) (Weiss et al., 2015): it does the same as *count2vec*. However, instead of storing tokens, it directly maps each token to a column position in the matrix of occurrences. It is mainly useful for large datasets and unlike *one-hot* and *count2vec*, which have the same dimensionality as the vocabulary length, this method has the flexibility to hash tokens in any dimensionality.

TF-IDF (Luhn, 1958): the term frequency (TF) captures the frequency of a particular token w.r.t. to a given document, whereas the inverse document frequency (IDF) measures how common the token is in the corpus. TF can be simply the number of times the token appears and the IDF is calculated as follows: $idf(t, D) = log(\frac{N}{count(d \in D : t \in d)})$, where t is the token and N is the number of documents d in the corpus D. Thus, the *TF-IDF* is obtained by multiplying both TF-IDF$(t, d, D) = tf(t, d) \times idf(t, D)$.

Word2vec (Mikolov et al., 2013a,b): the main contribution behind *word2vec* was learning distributed representations of words and reducing the computational cost compared to the state-of-the-art at the time. Although there are two original model architectures for learning the word vectors, Continuous Bag-of-Words ($CBOW$) and Continuous

Skip-gram Model (*skip-gram*), the core characteristic of *word2vec* is the removal of the hidden layer of a simple Neural Net Language Model. *CBOW* predicts the current word based on the t words around it, i.e., it predicts w_t given $(w_{t-i}, \dots, w_t - 1, w_{t+1}, \dots, w_{t+i})$. On the other hand, given w_t, the *skip-gram* predicts the surrounding words $(w_{t-i}, \dots, w_t - 1, w_{t+1}, \dots, w_{t+i})$. The parameter i in both cases is a parameter representing a range surrounding the current word w_t.

Doc2vec (Le and Mikolov, 2014): this algorithm is an extension of *word2vec* and learns the embeddings of documents (sentence, paragraph, essay, etc.). The difference w.r.t. *word2vec* is given by the learning which is performed via the distributed memory and distributed bag of words models and by adding another vector (document ID) to the input.

GloVe (Pennington et al., 2014): this is an unsupervised learning algorithm for obtaining vector representations for words. The main intuition behind this model is the capturing ratios of word-word co-occurrence probabilities in order to capture both local and global dependencies. This is expressed by $F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$, where P_{ik} and P_{jk} are the probabilities that the word k appears in the context of words i and j respectively.

5.4. Graph-based encoding

Process models are based on graphs, therefore, representing event data as graphs where nodes are activities and edges are control-flow relationships becomes natural. Considering that graph embedding methods offer promising encoding capabilities, it is of interest to study how such algorithms behave in the PM domain (Barbon Junior et al., 2020). Furthermore, graph embeddings open new possibilities for PM analysis, such as capturing graph structures and finding similarities across different process models. The intuition behind graph embedding methods is to represent nodes of a graph as low dimensional vectors, where such vectors are representative enough to keep its original relations (edges) intact. We can formally define the general idea as follows. A graph can be described as $G = (V, E)$, where $V = \{v_1, \dots, v_n\}$ is a set of vertices (nodes) and E is a set of edges $e = (u, v)$ that connect a pair of vertices $u, v \in V$. Given a graph G, a graph embedding is a mapping function $f : v_i \rightarrow y_i \in R^d$, such that $d \ll |v|$ and f preserves the original structure of their local neighborhood and minimizes the information loss. In this section, we describe graph embedding methods for event log encoding.

DeepWalk (Perozzi et al., 2014): it can be seen as a two-stage algorithm. First, a discovery of the local structure is performed through random walks. There are two parameters here, the number of random walks α and the number of vertices to visit t for each random walk. Second, similar to the *word2vec*, the *skip-gram* is performed to learn the embeddings. The intuition behind this algorithm is learning embeddings close to each other if they often occur in a similar structural context.

Node2vec (Grover and Leskovec, 2016): this algorithm is similar to *DeepWalk*, where the difference is a biased-random walk that aims at employing a trade-off between breadth-first and depth-first searches. In practice, such balance is capable of providing more informative embeddings than *DeepWalk*.

Walklets (Perozzi et al., 2017): while *DeepWalk* and *node2vec* implicitly capture a certain level of local dependencies by generating multiple random walks from a starting point node, this algorithm combines factorization approaches with random walks to capture both local and global information. It preserves dependencies by sub-sampling short random walks on the vertices and by skipping over steps in each random walk. This results in paths of fixed lengths composing sets of pairs of vertices. Thus, these sets are used to learn the latent representations.

role2vec (Ahmed et al., 2022): this is a framework that uses random walks to approximate the pointwise mutual information matrix, which is obtained by multiplying a matrix of structural features with the pooled adjacency power matrix.

Laplacian Eigenmaps (Belkin and Niyogi, 2001): the algorithm aims to find a low-dimensional this algorithm intuitively keeps the embedding of two nodes close when the weight W_{ij} is high. Given a graph G, this algorithm computes eigenvalues and eigenvectors $Ly = \lambda Dy$, where D is a diagonal weight matrix $D_{ii} = \sum_j W_{ji}$, and W is the weight matrix. Thus, $L = D - W$ is the Laplacian matrix that can be used to minimize the function $\rho(Y) = \frac{1}{2} \sum |Y_i - Y_j|^2 W_{ij} = tr(Y^T LY)$, where $Y \in \mathbf{R}^{N \times \hat{d}}$ is a matrix of $\hat{d}$ eigenvectors (desired dimensionality of the low-dimensional space) associated with the smallest eigenvalues.

GraRep (Cao et al., 2015): it aims to learn node embeddings by utilizing higher-order structural information in a graph. It computes k-step transition matrices, performs Singular Value Decomposition (SVD) on these matrices, concatenates the resulting singular vector matrices, and normalizes the concatenated matrix to obtain the final node embeddings.

Hope (Ou et al., 2016): this embedding algorithm is similar to *GraRep*, but instead of using the transition probability matrix, it employs a similarity matrix S. Thus, S can be obtained by using different similarity measures and consequently preserves higher-order dependencies.

BoostNE (Li et al., 2019): this algorithm performs a non-negative matrix factorization to calculate the residuals generated by previous embedding models. It assumes the same idea as the gradient boosting method in ensemble learning, where multiple weak learners lead to a better one when aggregated. Given a connectivity matrix obtained through the adjacency matrix of the graph, the algorithm calculates k residual matrices and uses each one as input to the next one using the following equation:

$$R_i = \begin{cases} X, & \text{if } i = 1 \\ max(R_{i-1} - U_{i-1} V_{i-1}, 0), & \text{if } i \geq 2 \end{cases} \tag{1}$$

where $U_i \in R_+^{n \times d_s}$ and $V_i \in R_+^{n \times d_s}$ intuitively act like the embedding representation of the center node and the context node in the ith level, respectively. Assuming the defined residual matrix, the embedding representation at the ith level is obtained by minimizing the loss function $L = \min_{U_i, V_i \geq 0} \|R_i - U_i V_i\|_F^2$, for $1 <= i <= k$.

Diff2vec (Rozemberczki and Sarkar, 2020): the overall idea of this algorithm is sub-sampling diffusion graphs for each node in a graph and generating sequences of vertices through an Euler tour. Given a graph G, a graph G' of l vertices is sub-sampled in a diffusion-like random process. Then, from G', sequences of vertices are generated by performing an Euler walk. In this process, G' is first converted to a multi-graph by doubling each edge. Thus, the Euler walk is employed instead of the random walk since this algorithm can capture a more complete view in graphs with this characteristic. The generated sequences of vertices are then used to create the graph embedding.

GLEE (Torres et al., 2020): unlike most graph embedding algorithms that expect similar nodes to have their embeddings close to each other, this algorithm uses the Laplacian matrix of a given graph to find an embedding with geometric properties. Examples of such properties are dot product (angle), length (area or volume) of a line segment (or polygon), the convex hull of a set of vectors, etc. Thus, given a graph G and its Laplacian matrix L, this procedure extracts eigenvectors corresponding to the largest eigenvalues in L. These vectors are used as node embeddings.

NetMF (Qiu et al., 2018): this method is built on a theoretical analysis that shows the equivalence of different graph embedding algorithms based on *DeepWalk*. In the original paper, the authors show that methods that use negative samplings, such as *DeepWalk* and *node2vec*, implicitly perform matrix factorization. Thus, the framework *NetMF* is proposed to unify existing methods and perform an explicit factorization.

NMF-ADMM (Sun and Févotte, 2014): given an adjacency matrix, the *NMF-ADMM* algorithm learns the embeddings by using the alternating direction method of multipliers to solve the negative matrix factorization problem.

GraphWave (Donnat et al., 2018): given an undirected graph $G = (V, E)$, an adjacency matrix A (binary or weighted), and a diagonal matrix $D_{ii} = \sum_j A_{ij}$ representing the degree of node i, this method learns a structural embedding of every vertex $v \in V$. The resulting GraphWave represents the compact node embeddings, where each row corresponds to a node in the graph and each column represents a dimension in the embedding space. These embeddings encode the structural similarities and local connectivity patterns of nodes.

NodeSketch (Yang et al., 2019): this method recursively generates k-order node embeddings in a recursive manner. These embeddings are categorized into low-order ($k = 2$) and high-order ($k > 2$). At each step, k, a Self-Loop-Augmented (SLA) adjacency matrix is generated to obtain the embeddings. Low-order SLA is obtained by simply adding the identity matrix to the original adjacency matrix $M' = M + I$. On the other hand, high-order embeddings first sketch an approximate k-order SLA adjacency of the current nodes and merge it with the $(k-1)$-order SLA adjacency matrix in a weighted manner.

6. Methodology

This section describes the experimental analysis carried out to evaluate encoding methods. We provide details on the software and materials and on the metrics used in order to assess the quality of the surveyed encoding methods.

6.1. Implementation overview

Our implementation can be organized into three steps: (i) dataset preparation, (ii) encoding generation, and (iii) evaluation of the encoding methods from multiple perspectives. The source code is available online in this repository.[5]

First, we generated synthetic logs using the PLG2 tool (Burattin, 2015). Subsequently, the encoding of the generated logs was performed using open-source libraries in Python as described in Table 1, which include Sklearn,[6] Karate Club,[7] PM4PY,[8] NLTK,[9] Gensim,[10] GloVe,[11] and the *position profile* implementation on github.[12] We organize each method according to families and provide the respective references for original papers and online implementations. Moreover, we set as baselines the methods *count2vec*, *one-hot*, *n-grams*, which implement the most simple transformations and are commonly employed in process mining papers. In the case of event-level encoding, the procedure was first performed at the activity level and then the results were aggregated to obtain the trace representation (trace-level encoding). This aggregation takes the resulting encoded information of each activity and averages it into a unique feature vector. For graph-based methods, this aggregation was obtained in two different ways: from edges or from nodes.

[5] https://github.com/gbrltv/business_process_encoding

[6] https://github.com/scikit-learn/scikit-learn

[7] https://github.com/benedekrozemberczki/karateclub

[8] https://github.com/pm4py/pm4py-core

[9] https://github.com/nltk/nltk

[10] https://github.com/RaRe-Technologies/gensim

[11] https://github.com/maciejkula/glove-python

[12] https://github.com/gbrltv/meta_trace_clustering/blob/main/clustering.py#L64

6.2. Evaluation metrics

Assuming encoding methods are used to map the original problem space into a different vector space, we observed the quality of the new space based on several criteria. Moreover, each encoding method has particularities regarding performance delivered, descriptive capability, computational cost, and complexity of parameter space. Thus, to be effective, an encoding method should meet the following criteria:

- Expressivity: the relative capacity of an encoding method to map data from the event nature to a new n-dimensional space by preserving its intrinsic behaviors and by minimizing the information loss. In other words, it measures how well the method converts event data from one representation to another since the goal of encoding is to preserve the original meaning.
- Scalability: the property related to increasing or decreasing the computational cost regarding the elapsed time and the memory usage of encoding methods. The encoding method should be able to map the event log quickly, without compromising the PM pipeline run time.
- Correlation power: the capacity of an encoding method to improve the original problem space. The new feature vector needs to be highly correlated to the PM task goal, i.e., the encoded feature vector should enhance the performance of PM tasks.
- Domain agnosticism: refers to how well a given encoding method maps data from different domains. Encoding methods that are non-agnostic can be used only in specific applications.

There are different strategies and metrics to assess encoding methods considering the presented criteria. In this work, we exploit the followings. We exploited Principal Component Analysis (PCA) (Daffertshofer et al., 2004) to verify how well a vector space can be compressed. Classification complexity metrics (Lorena et al., 2019) to measure how well samples, i.e., encoded traces, are distributed within classes. The F1-score (Sasaki et al., 2007) to observe the impact of encoding methods on accuracy. Time and space complexity to assess the computational performances. Although individual metrics might present trade-offs, e.g., PCA leads to information loss at the cost of better interpretability, the combination of them to assess different criteria is self-complementary. Furthermore, by jointly assessing these metrics we are also capable of estimating an assumption of process complexity. For instance, small memory usage and low execution time paired with a positive correlation power might be intuitively interpreted as a less complex problem. On the other hand, a complex problem that presents low expressivity tends to perform poorly in the final PM task and hence leading to a low correlation power. Table 2 summarizes each metric we exploited.

6.3. Experimental design

Our experimental design relies on labeled data for ground truth evaluation of the compared encoding methods, as an extension of Barbon Junior et al. (2020). Synthetic event logs were generated based on standard PM research practices and anomalies were injected into the generated traces, representing an anomaly detection PM task. Afterward, traces were labeled as anomalous or normal, making our data set suitable for supervised learning. Our dataset was made more realistic by adding heterogeneous behaviors to the event logs.

PLG2 (Burattin, 2015) was used to create five different process models by performing a random generation of a process capable of capturing several behaviors, such as sequential, parallel, and iterative control-flow. The rationale of PLG2 is based on the combination of basic control-flow patterns (Russell et al., 2006), e.g., sequence, parallel split, and synchronization. In order to simulate real-world scenarios, the patterns are progressively combined according to predetermined rules. Each of the five generated process models defines five scenarios of

Table 1
Encoding methods and related details.

Algorithm	Year	Family	Implementation
n-grams (Gasparetto et al., 2022)	–	Baseline	NLTK
one-hot (Weiss et al., 2015)	–	Baseline	Sklearn
count2vec (Weiss et al., 2015)	–	Baseline	Sklearn
token-replay (van der Aalst, 2016)	2016	PM	PM4PY
alignment (van der Aalst, 2016)	2016	PM	PM4PY
Log skeleton (Verbeek and de Carvalho, 2018)	2018	PM	PM4PY
position profile (Ceravolo et al., 2017)	2017	PM	GitHub
hash2vec (Weiss et al., 2015)	–	Text	Sklearn
TF-IDF (Luhn, 1958)	1958	Text	Sklearn
word2vec (CBOW) (Mikolov et al., 2013b)	2013	Text	Gensim
word2vec (skip-gram) (Mikolov et al., 2013a)	2013	Text	Gensim
doc2vec (Le and Mikolov, 2014)	2014	Text	Gensim
GloVe (Pennington et al., 2014)	2014	Text	GloVe
DeepWalk (Perozzi et al., 2014)	2014	Graph	Karate Club
node2vec (Grover and Leskovec, 2016)	2016	Graph	Karate Club
Walklets (Perozzi et al., 2017)	2017	Graph	Karate Club
role2vec (Ahmed et al., 2022)	2018	Graph	Karate Club
Laplacian Eigenmaps (Belkin and Niyogi, 2001)	2001	Graph	Karate Club
GraRep (Cao et al., 2015)	2015	Graph	Karate Club
Hope (Ou et al., 2016)	2016	Graph	Karate Club
BoostNE (Li et al., 2019)	2019	Graph	Karate Club
diff2vec (Rozemberczki and Sarkar, 2020)	2018	Graph	Karate Club
GLEE (Torres et al., 2020)	2020	Graph	Karate Club
NetMF (Qiu et al., 2018)	2018	Graph	Karate Club
NMF-ADMM (Sun and Févotte, 2014)	2014	Graph	Karate Club
GraphWave (Donnat et al., 2018)	2018	Graph	Karate Club
NodeSketch (Yang et al., 2019)	2019	Graph	Karate Club

Table 2
List of criteria, strategies, and metrics to evaluate encoding methods in process mining.

Criteria	Analysis	Acronym	Description
Expressivity	Principal Component Analysis	PCA	Using the 2D projection of a PCA space it is possible to observe patterns across scenarios from different complexities, ranging from very low, low, average, high and very high expressivity.
	Ratio of the PCA dimension to the original dimension	T4	This measure is related to the proportion of relevant dimensions that the coded feature vector is composed of. A larger T4 value means more encoded features are needed to describe data variability.
Scalability	Encoding Time	Time	Accumulated time in seconds during the encoding task.
	Encoding Memory	Mem	Accumulated memory in kilobytes during the encoding task.
Correlation power	Ratio of intra/extra class near neighbor distance	N2	This measure is sensitive to how data are distributed within classes and labeling noise in the data. Low values are indicative of simple problems.
	F1-score	F1	Average of F1-score obtained from the anomaly detection task, representing the predictive performance delivered by an encoding method.
Domain agnosticism	General usage of algorithm	DA	This is a binary evaluation (Agnostic or Non-Agnostic) considering agnosticism regarding the PM domain.

Table 3
Anomalies used to simulate the real-life event logs.

Anomaly	Description
skip	A sequence of 3 or fewer necessary events are skipped
insert	3 or fewer random activities are inserted in the case
rework	A sequence of 3 or fewer necessary events are executed twice
early	A sequence of 2 or fewer events executed too early, which is then skipped later in the case
late	A sequence of 2 or fewer events executed too late
all	Scenario where the event log is affected by all anomalies listed above

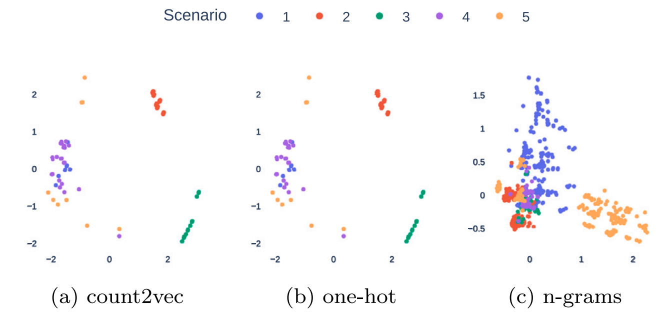


Fig. 3. 2D projections based on PCA transformation of the feature vectors of Baseline Family encoding methods of event logs from five different scenarios (1, 2, 3, 4, and 5). Each projection regards an encoding method, each point an encoded trace and each color represents a scenario.

different complexities based on the number of activities and gateways included in the scenario. An overview is presented in Table 4.

Creating the log requires simulating the process model. For that, we applied the ProM plug-in[13] for the simulation of a stochastic Petri net. We went through 10 thousand simulated cases and kept the default values of the other parameters. We injected anomalies, following Bezerra and Wainer (2013), by perturbing regular traces as proposed by Nolle et al. (2019), as in Table 3.

For each scenario, we injected different percentages of anomalies (5%, 10%, 15%, and 20%) by replacing normal traces. A total of 420 event logs were generated given five process models, six types of anomalies, and four anomaly percentages using labels and descriptions as additional attributes. Labels regard a normal execution or an anomalous one. The description attribute describes the anomaly and its impact on the case. It is important to note the different scenarios were created with increasing trace lengths and log sizes (1k, 5k, and 10k cases). We intuitively assume the complexity of event logs increases from scenario 1 to scenario 5 since we slowly increase these model properties such as the number of gateways and activities.

7. Benchmarking process mining encoding

In this section, we report on the results achieved in our experiments for each family of encoding methods (Baseline, Process Mining, Text, and Graph).

7.1. Expressivity

In this work, the expressivity of encoding methods is based on PCA and T4 analysis. PCA models were calculated using the encoded vector of all event logs for each encoding method, using a 2D sub-space projection of the first and the second principal component to identify how complex is the mapped space. Since the original problem (i.e., the event logs) is the same, differences in the distribution and density of the encoded logs can lead to interpretations about the mapping quality offered by each encoding method. In the PCA, each point represents an event log, and each color represents a scenario. The depiction highlights the encoding capacity of generating feature vectors preserving inter- and intra-traces similarities. This is demonstrated by the co-location of samples, i.e., encoded logs, of similar scenarios, i.e., the same color points near to same color points in each 2D projection. A high level of expressivity relies on non-overlapped clusters of samples from the same scenario with clusters sorted by scenarios' complexity occupying the whole sub-space projected. A low level of expressivity is associated with occluded samples with mixed sparse distributions or dense overlapped allocation. An average expressivity is identified when the projected distribution matches partially high and low characteristics. Very high and very low expressivity are obtained when an encoding method completely matches the mentioned characteristics, positively or negatively.

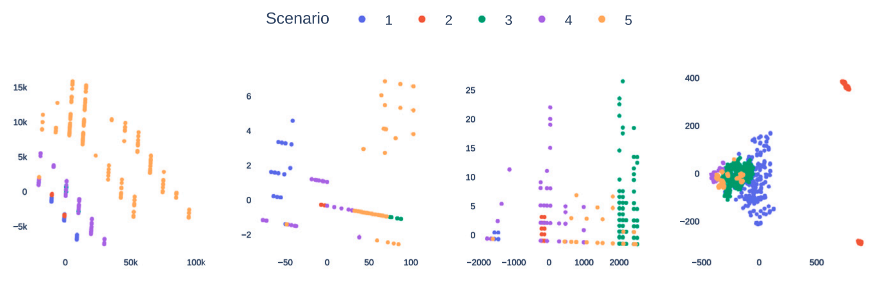
Fig. 3 shows the PCA projections of Baseline methods (*count2vec*, *one-hot*, and *n-grams*). Fig. 3.a and Fig. 3.b, regarding *count2vec* and *one-hot*, look similar since they separate the same classes, but keep uncovered a significant part of the space. Their analogous behavior comes from the fact that both techniques are very similar, with the difference that *count2vec* accounts for frequencies. However, as event data vocabulary is small and the number of repetitive activities within traces is not highly considerable, these methods present the same levels of expressivity. On the other hand, *n-grams* (Fig. 3.c) have an average expressivity as different scenarios overlap in the same areas.

Fig. 4 shows the projections of PM-based encoding methods. They can be assessed to very high and high expressivity levels, respectively *alignment* (Fig. 4.a) with the highest level, followed by *token-replay* (Fig. 4.b) and *Log skeleton* (Fig. 4.c). It is worth observing that the distribution in the *alignment* projection follows the complexity of scenarios. Furthermore, the *position profile* (Fig. 4.d) presented the worst expressivity, i.e., low.

The Text encoding family presented average, low, and very low levels of expressivity, as shown in Fig. 5. *GloVe* and *hash2vec* are from average level, as observed in Fig. 5.a and Fig. 5.b, due to the unsupervised nature of both methods. Low levels of expressivity were obtained by *TF-IDF* (Fig. 5.c), *CBOW* (Fig. 5.d) and *skip-gram* (Fig. 5.e). This similar behavior makes sense since the latter two ones are variations of the same algorithm (*word2vec*). The worst expressivity level, very low, was obtained by *doc2vec* (Fig. 5.f). Since this method is an extension of the *word2vec*, if the original algorithm performed poorly it is natural the extended version performs worse.

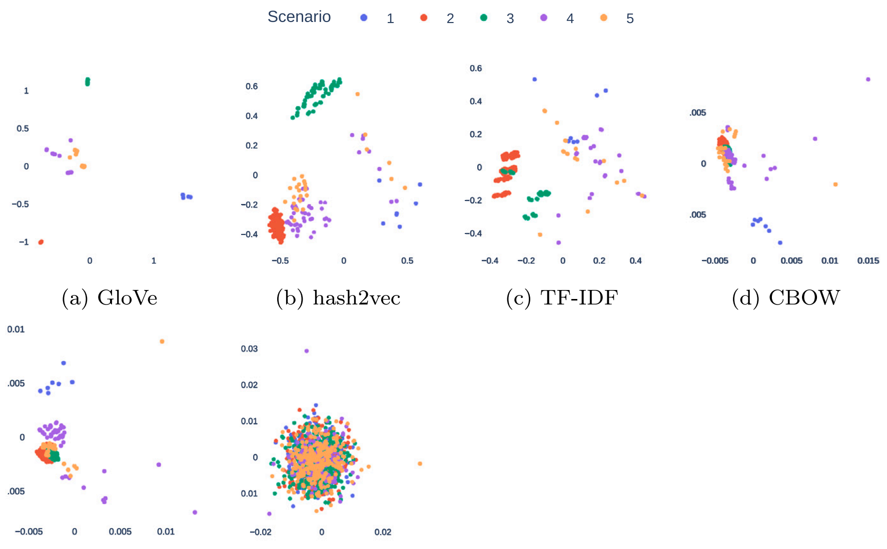
Very high, high, and average were the levels observed when using the Graph encoding family (Fig. 6). Average was obtained by *BoostNE* (Fig. 6.a) and *role2vec* (Fig. 6.m). PCA projections that represented high expressivity were computed from encoded vectors of *DeepWalk*, *diff2vec*, *GLEE*, *GraRep*, *Hope*, *Laplacian Eigenmaps*, *NetMF*, *NMF-ADMM*, *node2vec*, NodeSketch and *Walklets*, respectively, Fig. 6.b, Fig. 6.c, Fig. 6.d, Fig. 6.f, Fig. 6.g, Fig. 6.h, Fig. 6.i, Fig. 6.j, Fig. 6.k, Fig. 6.l, and Fig. 6.n. Very high expressivity was observed using *GraphWave* (Fig. 6.e), where it is possible to observe an organized gradient by scenario complexity, all different event logs are identified spread in the 2D projection. The fact most methods have achieved high expressivity indicates that graph-based methods in general are suitable for accomplishing expressivity for a process mining task. This result is confirmed by the measurements obtained with the T4 measure.

[13] http://www.promtools.org/doku.php



(a) alignment (b) token-replay (c) log skeleton (d) position profile

Fig. 4. 2D projections based on PCA transformation of the feature vectors of Process Mining encoding family methods of event logs from five different scenarios (1, 2, 3, 4, and 5). Each projection regards an encoding method, each point an encoded event log and each color represents a scenario.



(a) GloVe (b) hash2vec (c) TF-IDF (d) CBOW

(e) skip-gram (f) doc2vec

Fig. 5. The figure illustrates the 2D projections based on PCA transformation of the feature vectors of Text encoding family methods of event logs from five different scenarios (1, 2, 3, 4, and 5). Each projection regards an encoding method, each point an encoded event log and each color represents a scenario.

Table 4
Overview of five process models. For each scenario, we generated event logs by combining three different cardinalities, injecting seven different anomalies, at four different rates of injection (5%, 10%, 15%, and 20%). This resulted in 84 event logs for each scenario and 420 event logs in total. *gw*, *acts*, *evts*, and *vars* stand for the number of gateways, activities, events, and variants, respectively.

Log	#gw	trace size	#acts	#cases (10^3)	#evts (10^3)	#vars (10^3)
				1	75 ± 1	1 ± 0
scenario 1	8	9-13	22	5	378 ± 5	2 ± 0
				10	755 ± 9	2 ± 1
				1	186 ± 1	1 ± 0
scenario 2	12	26-30	41	5	929 ± 5	5 ± 0
				10	1857 ± 10	10 ± 0
				1	308 ± 1	1 ± 0
scenario 3	22	42-50	64	5	1538 ± 5	5 ± 0
				10	3077 ± 10	10 ± 0
				1	89 ± 3	0 ± 0
scenario 4	30	3-30	83	5	439 ± 6	2 ± 0
				10	879 ± 12	4 ± 0
				1	133 ± 3	1 ± 0
scenario 5	34	4-37	103	5	659 ± 7	3 ± 0
				10	1318 ± 13	6 ± 0

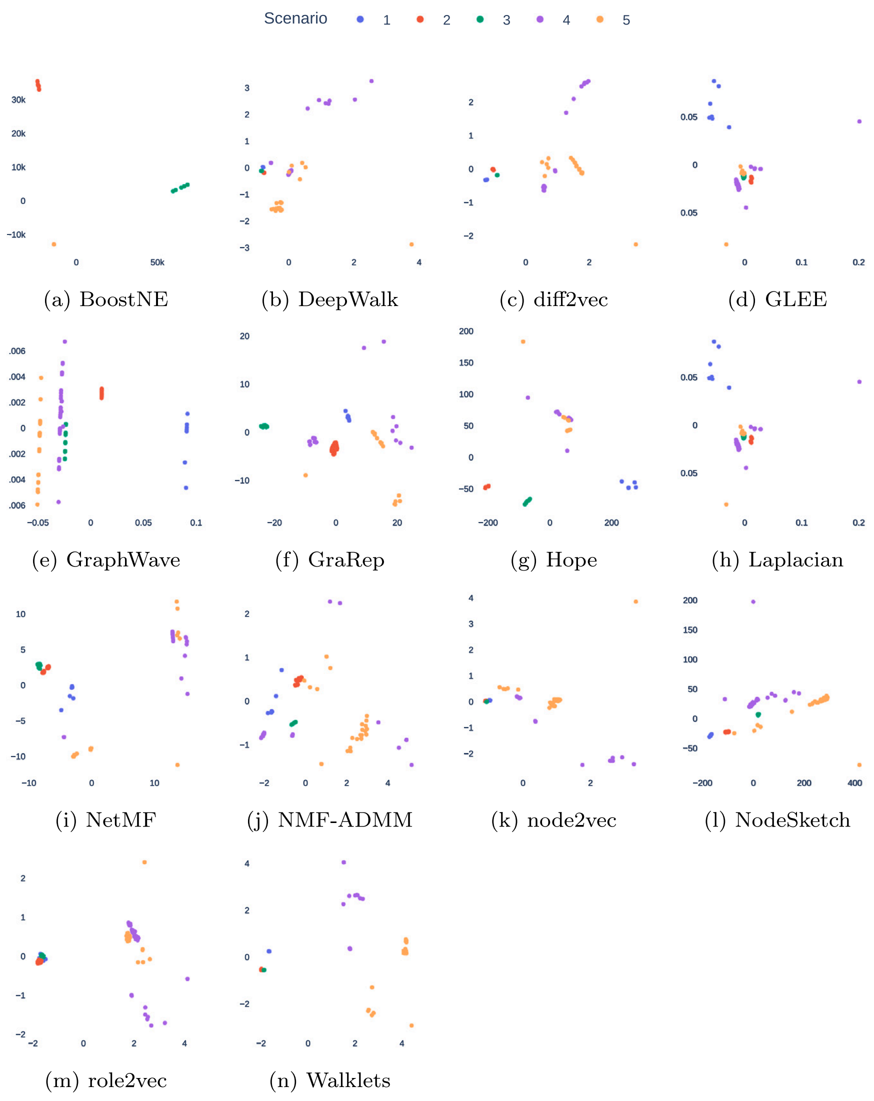


Fig. 6. 2D projections based on PCA transformation of the feature vectors of Graph encoding family methods of event logs from five different scenarios (1, 2, 3, 4, and 5). Each projection regards an encoding method, each point an encoded event log and each color represents a scenario.

T4 gives a rough measure, from 0 to 1, of the proportion of relevant dimensions used by the encoding vector to map the event log (Barbon Junior et al., 2020). Relevance is determined according to the PCA criterion, which strives to describe most of the variability in the data with uncorrelated linear functions of the features (Lorena et al., 2019). A higher T4 value indicates a more complex relationship between the input variables, indicating a larger number of original features are required to describe the data variability. Graph-based methods obtained the best expressivity, followed by the Baseline family, as shown in Fig. 7. In terms of T4 value, *doc2vec* reached the lowest expressivity level, 0.92.

7.2. Scalability

We drive the discussion of scalability considering the time (seconds) and memory (KB) consumption accumulated during the whole encoding process. Since costly methods are prohibitive to real-life event logs with huge volumes of data, their time and memory costs can directly influence the choice of an encoding method. In our experiments, we considered the time and memory consumed only during the encoding task. By comparing three different versions of event log size (1k, 5k, and 10k), we can observe how the costs are affected when encoding the same group of problems using different methods. Figs. 8, 9, 10 and 11 were used to demonstrate the scalability of time and memory, limiting the y-axis according to the higher observed value (time on left-side and memory on right-side) over all experiments of each encoding method.

Baseline family analyses are supported by Fig. 8. In terms of memory cost, the Baseline encoding family showed that *n-grams* was the most expensive method with a high space complexity. On the other hand, *position profile* was the worst method in terms of time complexity. *One-hot* demonstrated that memory costs increase more than time as the problem is scaled. In terms of scalability, *count2vec* presented the best scalability results of the Baseline family.

The PM encoding family presented high memory costs and average time costs, but with good scalability in both measures, as visible when the dataset increased and the performances slightly grew, as in Fig. 9. *Alignment* method was the cheapest one in terms of memory, *token-replay* presented a good balance in terms of time, and *Log skeleton* the most costly in both, memory and time.

The Text encoding family presented the fastest methods with low memory consumption. However, presented time scalability issues by the majority of methods (*GloVe*, *hash2vec*, *CBOW*, *skip-gram* and *doc2vec*), as represented by Fig. 10. The least scalable was *doc2vec*. A notable exception was *TF-IDF*, which presented reduced memory usage even with increasing problem size. Note that the scalability was evaluated considering the ability to not suffer from data growth, and

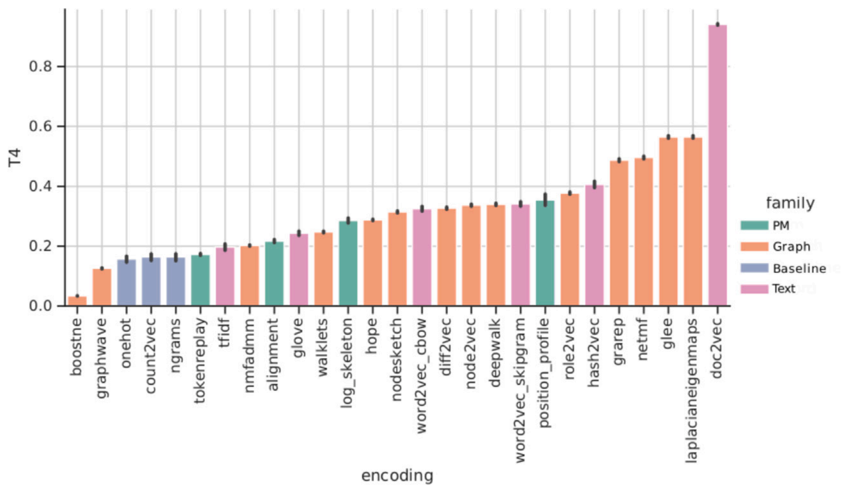


Fig. 7. T4 value obtained from encoding methods for expressivity evaluation. Low T4 values are correlated to good expressivity and high T4 values indicate poor expressivity.

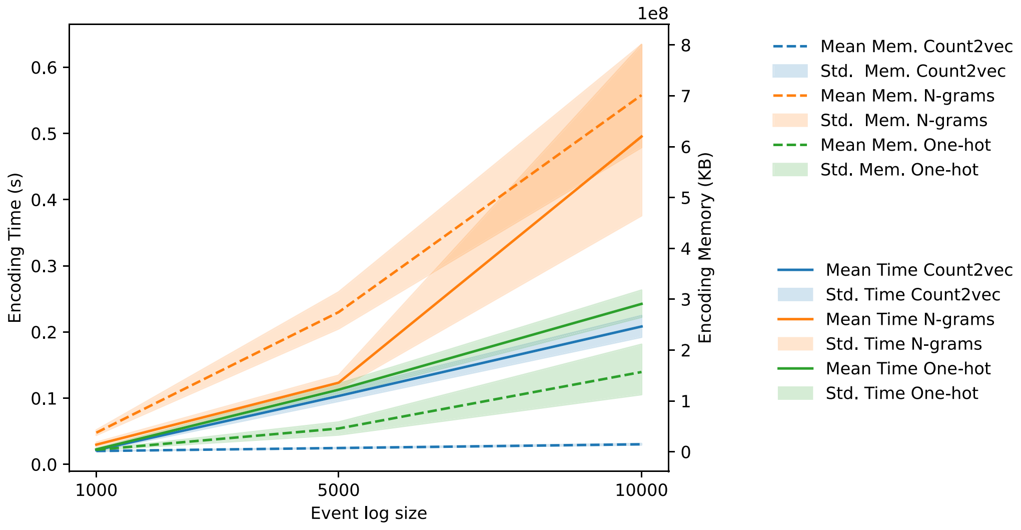


Fig. 8. Time and memory costs across different event log sizes (1k, 5k and 10k) when using Baseline encoding family.

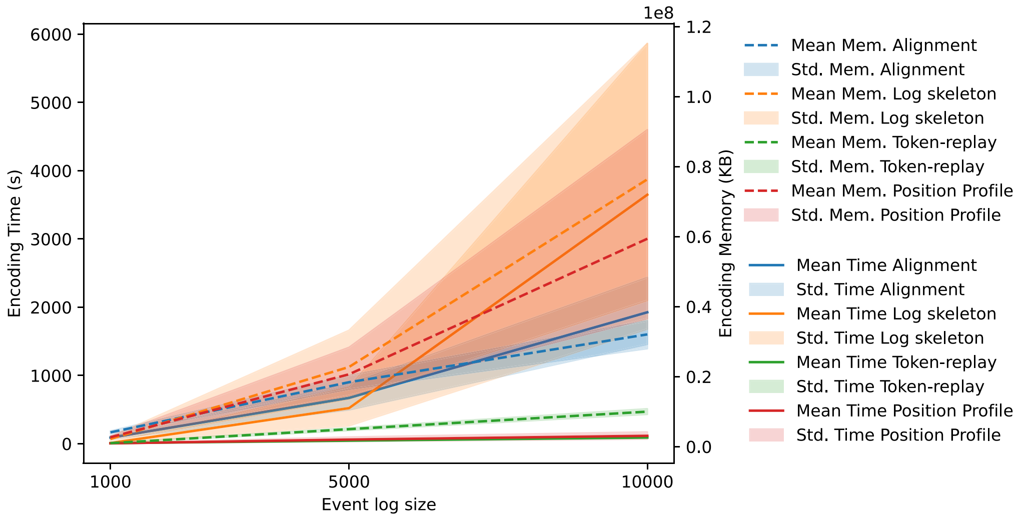


Fig. 9. Time and memory costs across different event log sizes (1k, 5k and 10k) when using Process Mining encoding family.

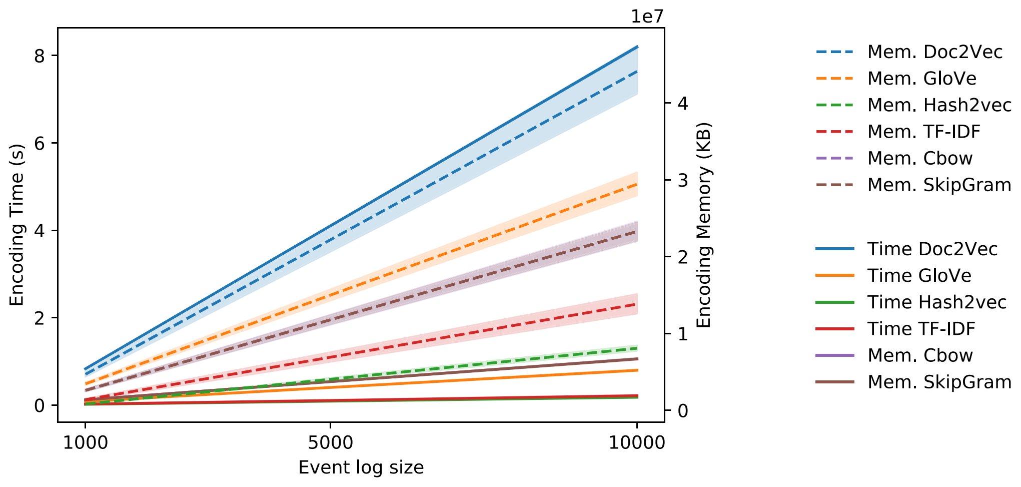


Fig. 10. Time and memory costs across different event log sizes (1k, 5k and 10k) when using Text encoding family.

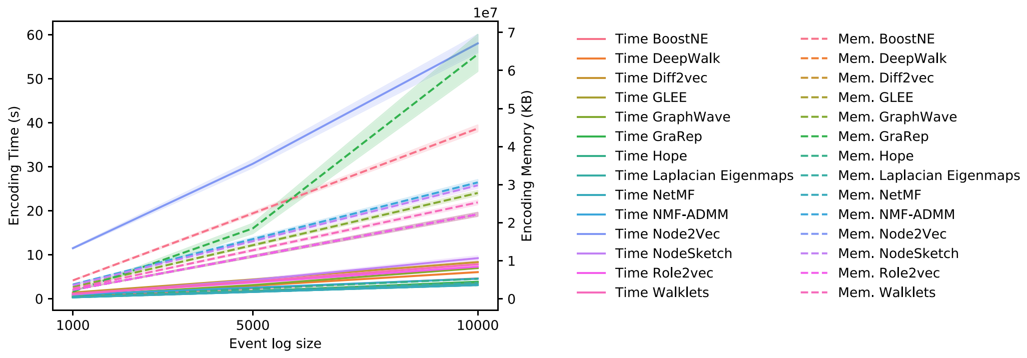


Fig. 11. Time and memory costs across different event log sizes (1k, 5k and 10k) when using Graph encoding family.

the average usage of time and memory in the Text encoding family is the lowest.

The Graph encoding family presented a heterogeneous usage of memory and an average time cost, as Fig. 11 shows. *GLEE*, *Hope* and *NetMF* were the fastest methods of this family. These methods presented average scalability. The best scalability was demonstrated by *NodeSketch*. The higher memory cost from the graph encoding family was achieved by *GraRep* with a cost comparable to *token-replay* (PM family) but with average scalability regarding time. The slowest method was *node2vec*, using the smallest event log it presented 4 times the average of the other methods of the same family. When dealing with the larger event log the time difference reached 7 times the other methods.

We organized the results of scalability and consumption of both time and memory analysis as a heat map (Fig. 12). In the figure, it is possible to observe that general low memory and little time consumption *count2vec*, do not reflect the scalability, i.e., increasing event log sizes, some methods compromise their costs with quadratic complexity costs of time and memory. Alternatively, *Log skeleton*, *token-replay*, and *NodeSketch* are very scalable but have a high memory and time cost. When considering raw time consumption, it is very evident how the PM family contains the slowest methods, being positioned in the last three points.

7.3. Correlation power

Correlation is an important analysis perspective since it reflects how correlated an encoding method is to the executed task in terms of performance. In other words, how the encoding positively contributed to the final performance. In our benchmark, we evaluated the correlation based on an anomalous trace detection task. In particular, we evaluated the F1-score obtained to detect anomalous behavior and the N2 from the mapped space. N2 is a ratio that computes distances between an example and its closest neighbor within a particular class (intra-class) and between an example and its closest neighbor from a different class (extra-class). The N2 value, which ranges from 0 to 1, is low when there is a greater distance between examples of different classes than between examples from the same class. Thus, a mapped space with a lower N2 refers to a representation that is better equipped to distinguish classes and support supervised learning. In our paper, we followed the N2 calculation as described by Lorena et al. (2019).

Fig. 13 represents the obtained N2 values from encoded space from all encoding methods. The figure presents each family with a particular color and results are sorted by N2, from the best one to the worst N2 value. *Position profile* achieved the best N2 values, mean of 0.48, and was the only method with less than 0.50 in terms of this measure. The top 5 N2 values were obtained by Graph and Text families. The PM encoding family, particularly *alignment* and *token-replay* reached high N2 values, superior to 0.85. In contrast, *Log skeleton* obtained less than 0.70, ranking in the bottom 10 encoding methods, but supported the best F1-score in the anomaly detection task.

When evaluating the F1-score, *Log skeleton* provided the best performance (mean of 0.942), followed by *position profile* (mean of 0.935) and all encoding from the Graph encoding family (above 0.915). Text encoding family provided results above 0.903, except *doc2vec* that lead to an average F1-score of about 0.845. Surprisingly, the most traditional encoding used in PM, *one-hot*, obtained the worst results, i.e., inferior to 0.840 of the F1-score. The obtained F1-scores are sorted by the performance from the best to the worst one in Fig. 14.

In order to provide a fair and statistically grounded comparison in terms of predictive performance, we used Friedman's statistical test and

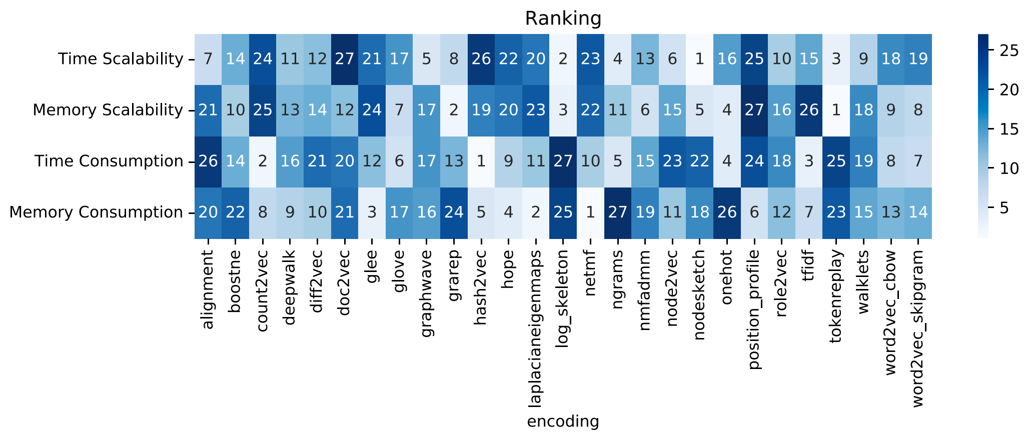


Fig. 12. Ranking of Time Scalability, Memory Scalability, Time Consumption, and Memory Consumption. The better approaches are positioned in the first ranking position, colored by white. The most costly and less scalable are the last potions with dark colors.

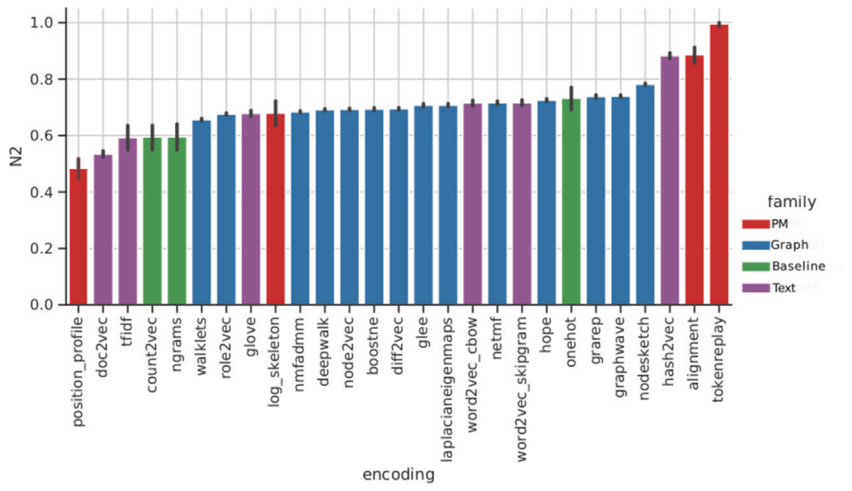


Fig. 13. Ratio of Intra/extra class nearest neighbor distance (N2), provided by the same group of tasks considering 27 different encoding methods. The bars are sorted from the best score (left) to the worst one (right). Each bar is colored according to the coding family.

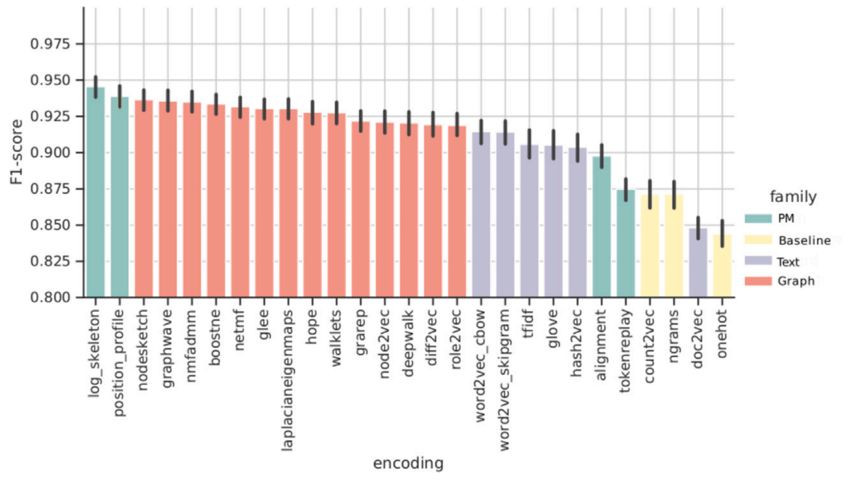


Fig. 14. F1-score obtained by several anomaly detection tasks performed using a Random Forest algorithm, considering 27 different encoding tasks. The bars are sorted from the best score (left) to the worst one (right). Each bar is colored according to the coding family.

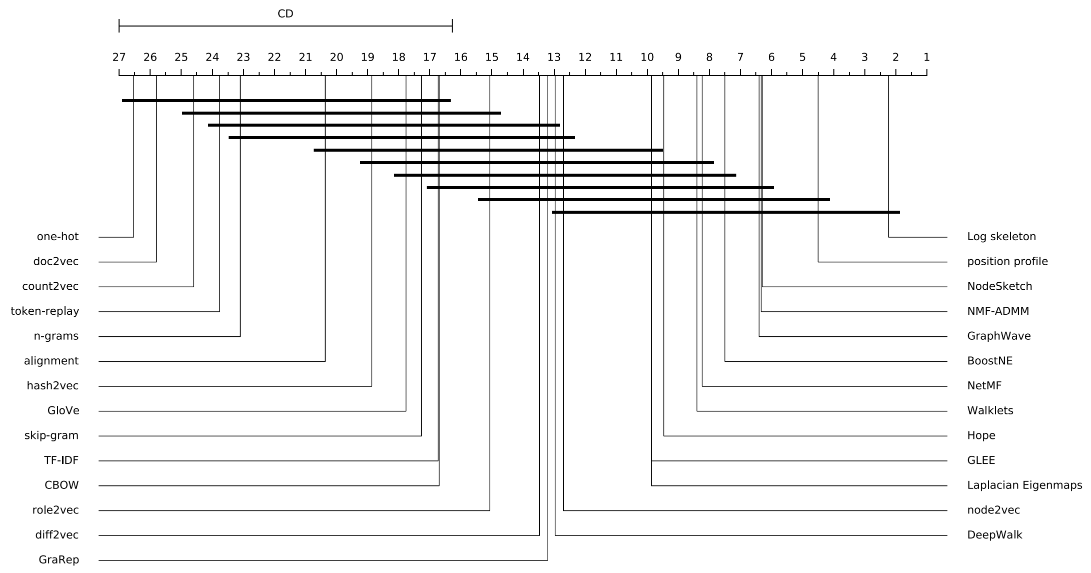


Fig. 15. Nemenyi post-hoc test (significance of $\alpha = 0.05$ and critical distance of 10.71) considering the **F1-score** (predictive performance) accuracy obtained when performing the anomaly detection task using all encoding methods over all scenarios (1k, 5k and 10k traces from five different scenarios). Statically similar methods are linked by the solid line.

the post-hoc test of Nemenyi. Both tests were employed to verify the statistically significant difference between the performance of the F1-score of each encoding method. The result of the statistical comparison can be observed as a Critical Difference chart, as illustrated in Fig. 15. Critical Difference test allows checking when there were statistical differences between the segmenters, each diagram and method's average ranks are placed on the horizontal axis, with the best ranked to the right. The solid line connects encoding methods with no significant performance difference. Thus, Fig. 15 demonstrates no statistical difference among *Log skeleton*, *position profile*, *NodeSketch*, *NMF-ADMM*, *GraphWave*, *BoostNE*, *NetMF*, *Walklets*, *Hope*, *GLEE*, *Laplacian Eigenmaps*, *node2vec*, all of them supporting high predictive performance. On the other hand, there is no statistical difference as encoding methods that provided lesser predictive models for *one-hot*, *doc2vec*, *n-grams*, *count2vec*, *token-replay*, *alignment*, *hash2vec*, *GloVe*, *skip-gram*, *CBOW* and *TF-IDF*.

7.4. Domain agnosticism

The last comparison criterion regards Domain Agnosticism. We consider this criterion as important as Expressivity, Scalability, and Correlation to comprehend the obtained results. Since a method valid for multiple applications is instrumental to the construction of adaptive data science pipelines.

The methods comprising the baseline family are traditionally used in different PM tasks. They perform straightforward transformations mapping traces into feature vectors. Their usage was not limited to PM pipelines, indeed, they were created in other data mining areas. For these reasons, we consider the Baseline family as domain agnostic. As well, the Text and Graph families have been used for a wide range of purposes. These last families are good examples of domain agnosticism and have been used with simple encoding scenarios as well as with complex and highly structured representations.

Considering our criteria, PM-based solutions are not domain agnostic. This family of encoding methods was conceived to represent event logs and has been used exclusively for this purpose. It should not be viewed as a limitation, but rather as a characteristic of specialized methods demonstrating a high correlation power, even at the cost of high computational costs, e.g., *Log skeleton* provided encoded spaces to induce models that obtained high F1-scores.

An overview of domain agnosticism and some implications of encoding method performance and resource consumption could be observed in Fig. 16. The most notable observation is that non-agnostic methods (*token-replay*, *alignment*, and *Log skeleton*) share high N2 values and high costs of space and time complexity. Among the agnostic methods, the best N2 and F1 performances are obtained with average space and time complexity, e.g., with *GraphWave* or *BoostNE*.

8. Encoding method selection

After providing a general discussion focused on the employed metrics, in this section, we dive into the algorithms' behaviors according to their parameters and the event logs' characteristics. We first present the impact of parameters for each family of algorithms. Subsequently, we evaluate the algorithms' behaviors for each type of anomaly injected in the event logs. Lastly, we present a similar evaluation but consider the event logs' properties. f

8.1. Parameter impact

In this section, we present how parameters might present different behaviors for encoding methods according to each event log. Employing synthetic data is beneficial for this evaluation since it allows us to have more control over different properties and obtain insights regarding which algorithm family and configurations can achieve better results according to the user's preferences. Thus, we discuss the employed configurations for parametric methods (i.e., graph- and word-based methods, see Table 5) across the employed scenarios. Furthermore, all the results in this subsection are filtered and only the performance values lying between the first and third quartiles are included to avoid outliers and bad visualizations.

Table 5 presents two important configurations that guide embedding generation: feature vector size and aggregation methods. A key aspect that we aimed to capture with experiments is scalability. Moreover, by analyzing different vector sizes we can assess how well the encoding method distributes the generated embeddings. This way, we fixed feature vector sizes to $\{2^n\}_{n=1}^{8}$, i.e., ranging from vectors of size 2 to 256. Due to vocabulary size, some embeddings were limited in terms of possible size configurations (e.g., *Laplacian Eigenmaps*, *GraRep*, *Walklets*, *NodeSketch*, *GLEE*, and *Hope*). *BoostNE* required a particular configuration that only allowed multiples of 17, this way, we tried to obtain vector sizes as close as possible to powers of 2. It is important to note that word and graph embeddings create an embedding representation for words and nodes, respectively. Therefore,

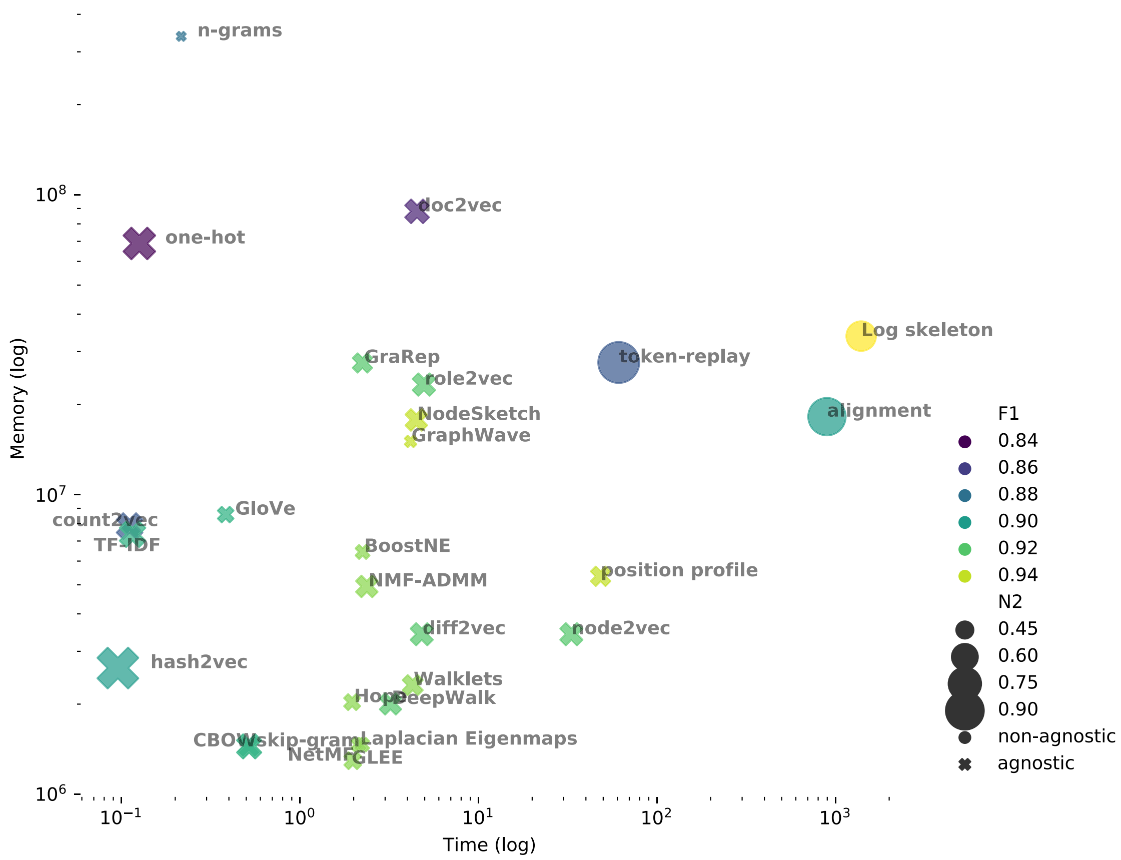


Fig. 16. Mean values of time, memory, and predictive performance (F1) projected by time and memory in a logarithmic scale. The marker size represents the N2 and their color gradient performance in terms of F1. The marker symbol represents the domain agnosticism evaluation.

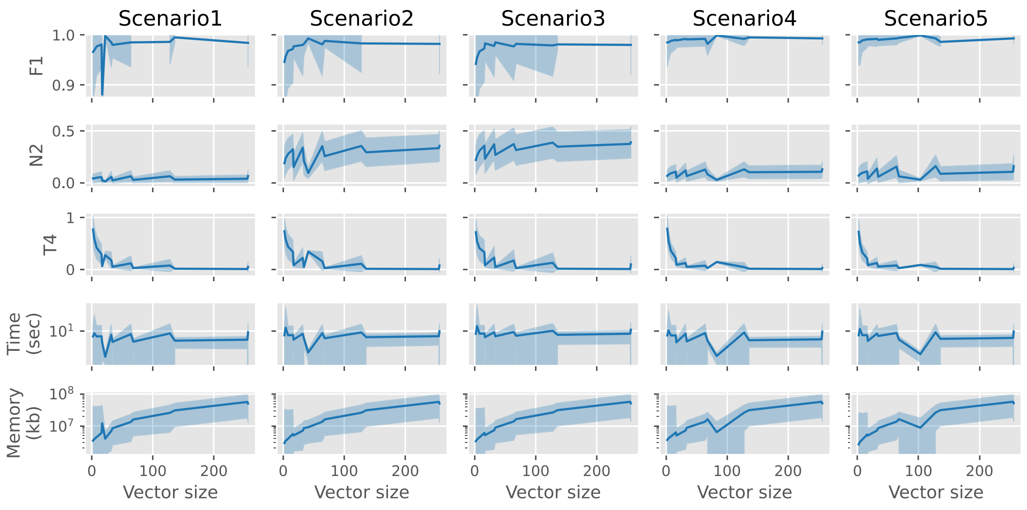


Fig. 17. Performance metrics (one for row) for different vector sizes throughout all the employed scenarios (one for column).

aggregation techniques are required to obtain a trace representation, i.e., an aggregation of the word or node representations. As for the trace, we selected two options: average and max pooling. For the graph embeddings, we can obtain representations for both the nodes and edges, increasing the possible aggregation options. Considering the proposed edge aggregations in Grover and Leskovec (2016), the edges embedding can be obtained following four binary operators: Average, Hadamard, Weighted L1, and Weighted L2. Given $f(x)$ as the function that returns the node embedding and u and v as two nodes belonging to a graph, the edge embedding is computed as described in Eqs. (2), (3), (4), (5).

$$Average = \frac{f(u) + f(v)}{2} \tag{2}$$

$$Hadamard = f(u) \times f(v) \tag{3}$$

$$Weighted\,L1 = |f(u) - f(v)| \tag{4}$$

$$Weighted\,L2 = |f(u) - f(v)|^2 \tag{5}$$

Fig. 17 shows the behaviors of different parameter vector sizes (see Table 5) across the five employed scenarios. We can see that for T4 and

Table 5
Employed configurations for word- and graph-based encoding methods.

Encoding	Vector size	Trace Agg.	Graph Agg.	Edge Agg.
GraphWave	$\{2^n\}_{n=1}^{8}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
Laplacian Eigenmaps	$\{2^n\}_{n=1}^{4}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
NMF-ADMM	$\{2^n\}_{n=1}^{8}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
DeepWalk	$\{2^n\}_{n=1}^{8}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
GraRep	$\{2^n\}_{n=1}^{4}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
node2vec	$\{2^n\}_{n=1}^{8}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
Walklets	$\{2^n\}_{n=2}^{8}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
role2vec	$\{2^n\}_{n=1}^{8}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
NetMF	$\{2^n\}_{n=1}^{8}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
NodeSketch	$\{2^n\}_{n=2}^{8}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
BoostNE	$\{17, 34, 68, 136, 255\}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
GLEE	$\{2^n\}_{n=1}^{4}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
Hope	$\{2^n\}_{n=1}^{5}$	{avg, max}	{node, edge}	{avg, had, w1, w2}
diff2vec	$\{2^n\}_{n=1}^{8}$	{avg, max}	–	–
word2vec	$\{2^n\}_{n=1}^{8}$	{avg, max}	–	–
hash2vec	$\{2^n\}_{n=1}^{8}$	{avg, max}	–	–
GloVe	$\{2^n\}_{n=1}^{8}$	{avg, max}	–	–
doc2vec	$\{2^n\}_{n=1}^{8}$	{avg, max}	–	–

memory usage, although the parameters affect the metrics, they behave similarly across all the scenarios. Both behaviors of this parameter for the mentioned metrics are intuitive, since T4 measures the ratio of the PCA dimension to the original data dimension, whereas the memory usage naturally increases by increasing the dimensionality of the encoded data. Furthermore, the elapsed time for encoding methods across scenarios does not change and we can see in the figure a similar behavior for them. Thus, if these are the metrics of interest of a user, the choices are intuitive and straightforward.

On the other hand, regarding the classification complexity metrics F1 and N2, the parameter choice depends on the characteristics of the event log. For example, for scenarios 1, 4, and 5, the N2 has slight differences, whereas it considerably changes for scenarios 2 and 3. Considering the F1 metric, although the average behavior is similar across the scenarios, we can see a high variation of performances in the first three scenarios for small vector sizes.

This evaluation highlights the difficulty of choosing the right parameter values according to the user's requirements. For instance, we can see that high F1 scores are achievable by employing high and low vector sizes. However, if only a limited amount of memory is available, an algorithm selection procedure is needed to find the right algorithm that achieves the desired score of F1 and fits the available storage. Nevertheless, if the user faces characteristics similar to scenarios 2 and 3, other parameters and algorithms (e.g. PM-based methods, see Fig. 13) should be evaluated to optimize the N2 metric.

Another parameter to be evaluated is the aggregation strategy for algorithms from the graph and word families. Fig. 18 shows the overall distributions of the employed methods for each type of aggregation. First, Fig. 18(a) illustrates a high-level overview regarding the aggregation of edges and nodes all at once. We can see that, in general, aggregating nodes for the final encoded trace significantly overcome the edge aggregation regarding the time and N2 metrics, whereas for the remaining ones it slightly overcomes or performs equally. This is due to the fact the full information of a trace is retained in the node representation and not in the edges that connect the encoded trace to similar or related (depending on the graph algorithm) neighbors. Second, the overall aggregations for edges and nodes according to each strategy (average or max) is presented in Fig. 18(b). We can see that for edges the overall behavior is the same across all metrics except for N2, which performs slightly better in general for average aggregation. Moreover, there is no best aggregation regarding nodes, since for the last three metrics the behavior is quite similar, whereas for F1 the average is slightly better (higher average score and lower variation), and for N2 is the opposite. Lastly, in Fig. 18(c) we also show the individual aggregation strategies for the edges, although there is no significant difference in performances among these ones.

Finally, the aggregations for methods from the word family are illustrated in Fig. 19. Similarly to Figs. 17 and 18, the metrics T4, elapsed time, and memory usage present similar behaviors. In this case, however, the F1 and N2 scores also behave similarly in general, although the average aggregation presents a slightly lower variation for the former one.

Despite the parameters evaluated in this section presenting similar behaviors for the metrics of T4, elapsed time, and memory usage, it is evident the wide distribution of possible performances regarding all metrics that might be achieved by the same parameter value across different event logs. Always selecting the same feature vector size and aggregation strategy might lead to drastic differences among metrics. Thus, in the next sections we evaluate which underlying characteristics in the event logs might affect the performances.

8.2. Anomaly analysis

In this section, we evaluate how each anomaly type affects the encoding methods in general. The performances regarding each type of anomaly are aggregated throughout all the families to visualize the general behaviors in Fig. 20. We employ the violin plots for this visualization since it better highlights the density curves and allows us to deeply see where the main differences among the anomalies lie. For instance, similar F1 scores are achieved by all methods and parameters regarding the *all* anomaly. We can assume the *attribute* anomaly as the easiest one for F1 since most performances tend to be almost 1, whereas, for the remaining anomalies, we see a small concentration close to 1 although the overall behavior is the same. On the other hand, the N2 metric is significantly affected by the different anomalies. Regarding the *all*, *attribute*, and *late* anomalies, some extra effort should be considered to find the right algorithm and its parameters since the values of the performance are more concentrated above their averages. Finally, the remaining metrics behave similarly for this evaluation as well.

In order to go deeper into the anomaly analysis, we present in Fig. 21 the same idea but this time separated by families and via bars. Since we are including all the results without excluding outliers as in the previous discussion, we can highlight overall differences among the families. Regarding the F1 metric we can notice a significant poor performance by the PM encoding methods, except for the *attribute*

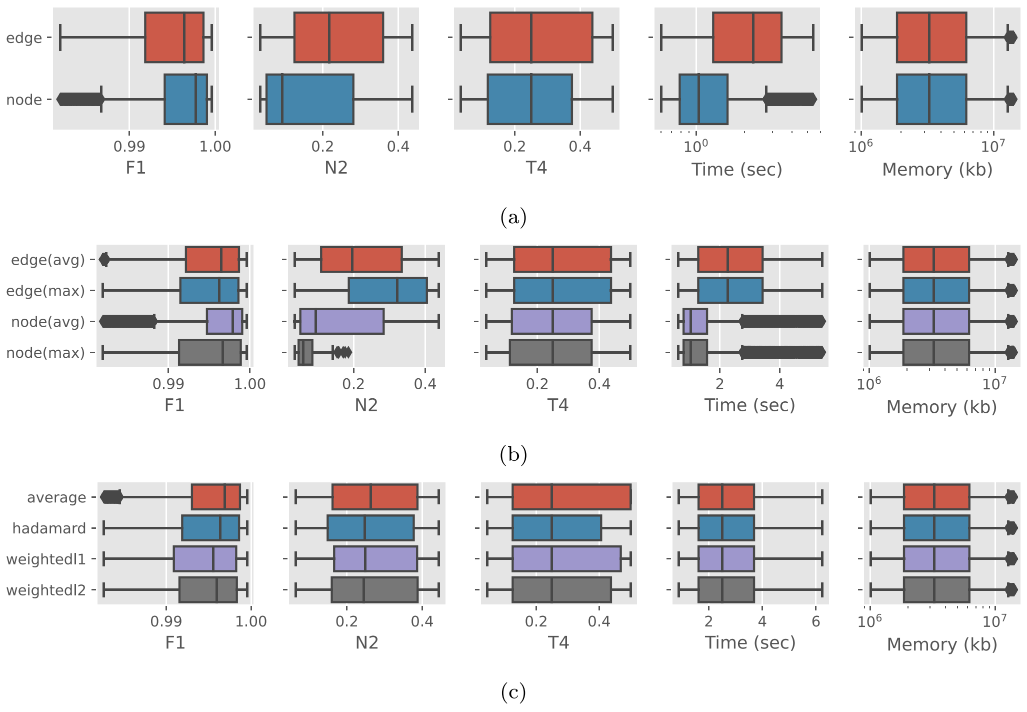


Fig. 18. (a) Overall aggregation of edges and nodes. (b) Specific aggregation of edges and nodes. (c) Aggregation strategies for edges.

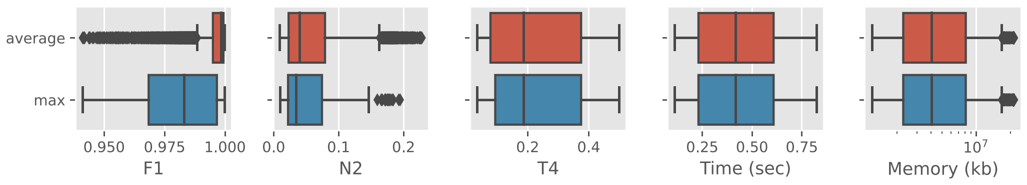


Fig. 19. Distribution of metrics for aggregations of word-based encoding methods.

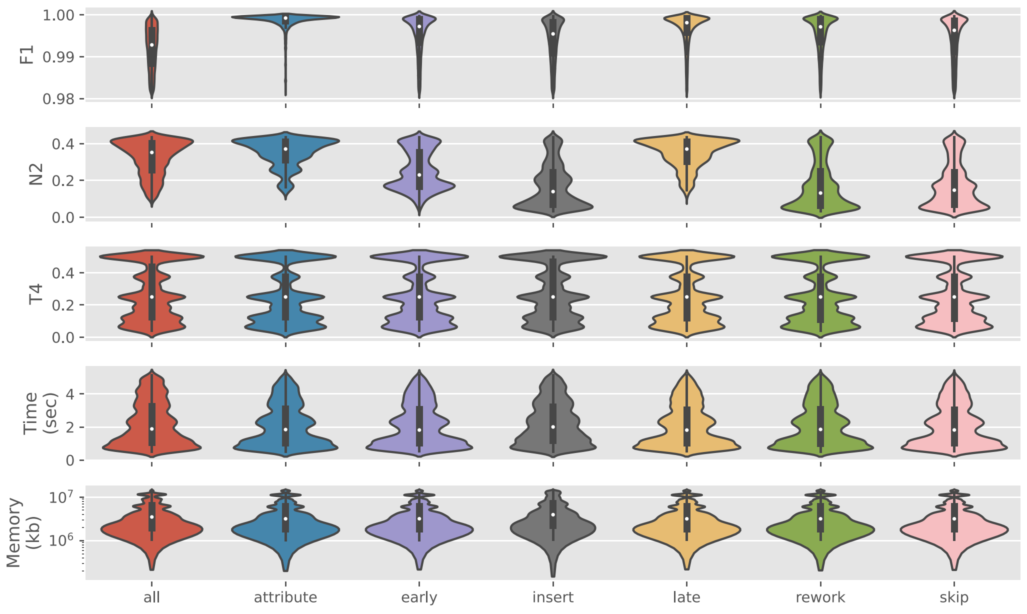


Fig. 20. Distribution of performances for each anomaly type.

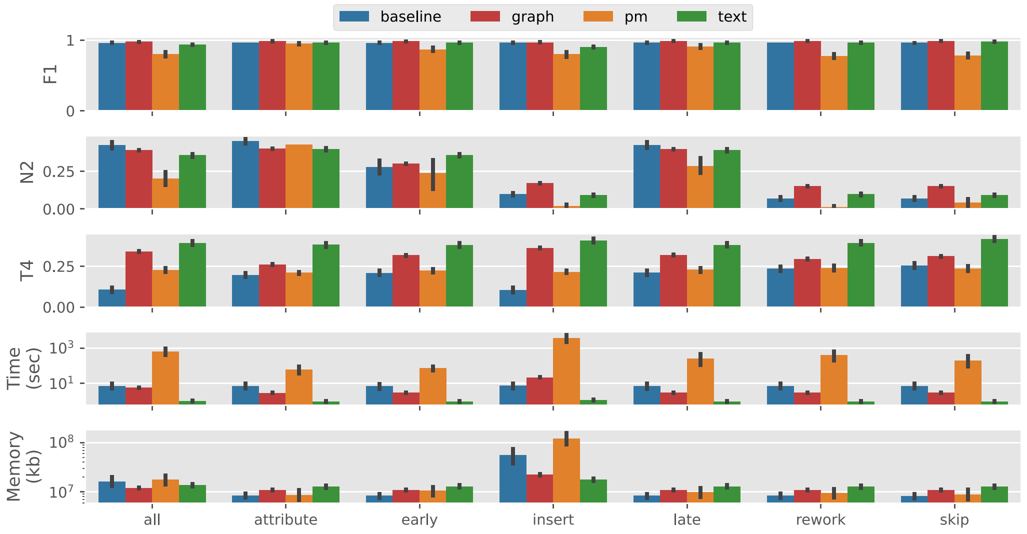


Fig. 21. Overall performances for each anomaly type and grouped by algorithm families.

anomaly. We can also notice slight variations in performances among anomalies with respect to the text-based methods, whereas this does not occur for the graph-based methods. Contradictorily, we can see a better performance of PM methods regarding the N2 metric, which emphasizes the difficulty of selecting a suitable algorithm for each task. Furthermore, the graph-based methods were overcome by the other ones in this metric.

More specifically, methods from the baseline and PM families present the lowest T4 metrics since they are non-parametric, which means they have the same encoded dimension regardless of the event log. Nevertheless, they are more time-consuming, especially the encoding methods based on PM techniques, due to the expensive nature of the algorithms and the lack of efficiency-oriented tools in the literature. Furthermore, the PM methods are drastically affected by the anomalies because some anomalies are easier to detect (e.g., early) than others (e.g., rework). Encoding methods from the graph and word families are not affected by the anomalies and they present the same patterns in general.

8.3. Log behavior

Besides the anomaly type affecting the PM family algorithms, in this section, we will analyze further properties of event logs that might also affect the performance metrics.

In Fig. 22, we show the overall performances achieved by each family of algorithms over the scenarios. Regarding the F1 metric, the first three scenarios present a slightly higher variation in performance, suggesting more difficulty in finding the right algorithms and parameters. However, this variation is due to the fact that simple data (i.e., a small number of activities, short traces, etc.) do not require exploring such a wide range of parameters, and the higher the parameter values (e.g., vector size) the higher the information loss. Regarding the remaining metrics, we can see some disturbance with respect to scenarios 2 and 3. There is a high variation and high average values for N2, which emphasizes that the event properties affect the algorithms' performances. The PM family performs better for T4 but it is outperformed in the sense of scalability by all the other ones. Overall, the graph methods perform better or are similar with respect to most of the other ones across all scenarios.

This analysis indicates that event log properties do not drastically affect the employed encoding methods, except for the PM family. This is intuitive since this family has an underlying bias to handle this specific data domain, i.e., process data.

Since the specific event properties do not present many behavior changes, in Fig. 23 we perform the last evaluation regarding the log characteristics but with respect to their sizes for each performance metric. For F1, we can notice that the log size affects only the PM methods, indicating they are not scalable. Again, scenarios 2 and 3 seem problematic since the families' behaviors significantly change for the N2 metric although the remaining behavior remains the same as discussed in the previous figure. The T4 has the expected behavior since this metric should not be affected by the event log sizes. Similarly, the scalability metrics regarding the elapsed time and memory usage are also intuitive since they increase as the event log size increases as well.

9. Issues, discussion, and future directions

Research comparisons about encoding methods focused on PM are still embryonic. We classified encoding methods from different areas in Section 5, and we observed that works in PM do not investigate the strong and weak points of each method. Thus, we have proposed a study involving a large set of methods from different families over several event logs.

Investigating the pros and cons of encoding methods based on *expressivity*, *scalability*, *correlation power*, and *domain agnosticism* over different encoding families and several event logs with various complexities, we were able to gain insights and share some assumptions for future directions and possible novel PM encoding methods. We believe the criteria employed in this work to assess the effectiveness are the most important aspects to take into consideration in order to achieve significant accomplishments for any PM task that needs to encode event logs. In addition, the deep evaluation regarding the impact that parameters and data characteristics have on the performance metrics will serve as guidelines for practitioners and researchers when choosing a suitable algorithm and set of parameters.

9.1. Overview of experiments

The combination of metrics to evaluate different criteria can serve as user requirements when deciding which encoding method should be employed for the specific problem. The *expressivity* of an encoding method can measure how effectively the data original event data is

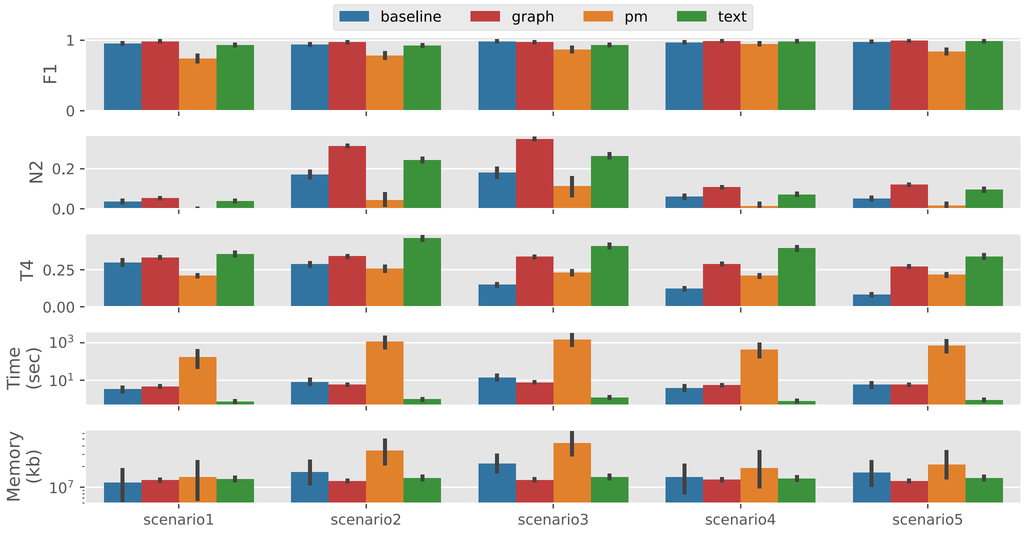


Fig. 22. Performances by scenario.

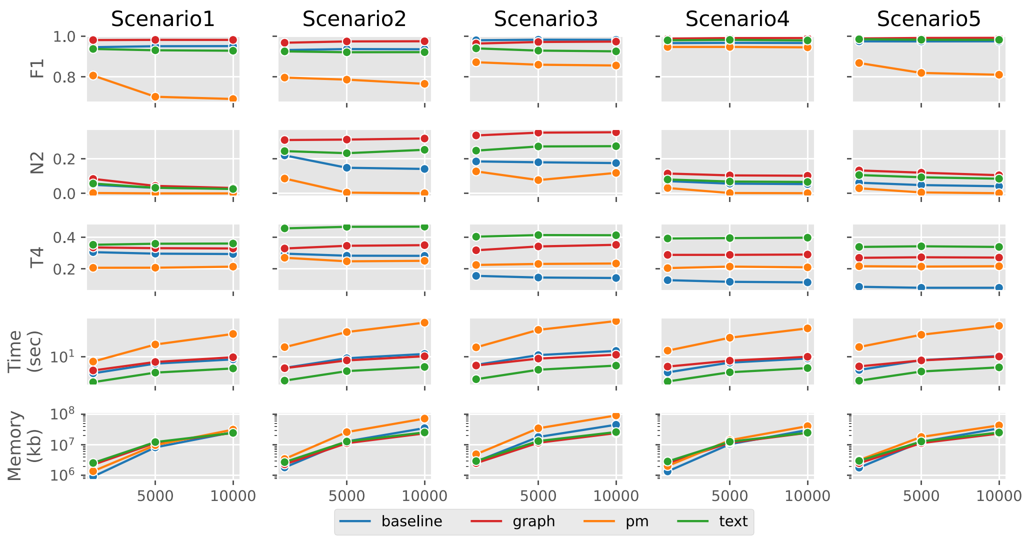


Fig. 23. Performances (rows) for each scenario (columns) according to different event log sizes.

mapped into a new feature space by preserving the underlying process structure and minimizing the information loss. The *scalability* is also an important concern since real-life event logs consist of a large volume of data which can lead to high computational costs regarding elapsed time and memory usage. Understanding the *correlation power* between the nature of an encoding method and the performance of the executed task is also a relevant analysis that allows practitioners to estimate the complexity of analyzing a specific event log. Lastly, the *domain agnosticism* is a novel and important discussion introduced in this work to consider if encoding methods can be adapted for different problem domains.

From the extensive evaluation in Section 7, we can conclude that methods from graph and word families can be very expressive or not in terms of T4 (see Fig. 7). Thus, from the presented results practitioners could leverage the insights to select the right algorithm in case this is a relevant or priority metric for a particular task. Visually, according to the PCA figures, the graph-based methods are better since they present fewer overlaps among the scenarios in general, although word-based methods like *hash2vec* and *GloVe* also present good expressivity in terms of visualization. On the other hand, regarding the scalability, we present a generic analysis in Section 7 and provide a more deep discussion in Section 8. In the latter section, it is clearly illustrated that PM encoding methods suffer at scaling when the event log size increases, regardless of other data characteristics like the number of activities or trace sizes.

In terms of *correlation power*, the PM methods present some stren gths depending on the type of anomaly present in the event log. Although this family performed poorly in general regarding the F1 metric, it outperformed the other ones for N2. However, these methods are extremely sensitive to anomalies in general. Furthermore, the graph-based methods present stability since it does not drastically change across the different anomalies and scenarios employed in this work. The same behavior can be seen for word-based methods, although they performed slightly worse than graph-based methods in the sense of correlation power. Nevertheless, it presents better scalability in terms of elapsed time, memory usage, and also event log sizes. Lastly, the scenarios do not seem to affect the F1 metric in general, except for PM methods, but they affect the N2 across the scenarios.

9.2. Challenges and future directions

Addressing further issues on encoding methods in PM, online PM may introduce new challenges for the current encoding methods. As emphasized by Ceravolo et al. (2020), measures such as accuracy and memory consumption need to drive the creation of methods to match online PM goals, as the encoding methods used in such solutions. The concern of limitations posed by an online PM task was also highlighted by Tavares et al. (2019), mentioning the demand for adapting when dealing with concept drifts and focusing on inter-activity time implications. STARDUST (Pasquadibisceglie et al., 2022) is an example of online PM, particularly on trace streams. The authors discussed approaches to handling traces recorded without the final activity. It is therefore essential to investigate encoding methods that support online PM tasks coping with new challenges, such as reduced memory consumption and the ability to map partial traces. In our experiments, we used *scalability* measure to support insights and discussions regarding this topic.

Currently, the predictive process monitoring problem also faces this issue regarding the lack of encoding methods specific to PM. Representation learning or feature learning is a learning paradigm that has been recently introduced in the community by Koninck et al. (2018). We believe this is a promising path for improving the encoding procedure in process mining tasks. In the mentioned work, the authors derived their new proposals from the *word2vec*. However, we believe PM requires a specialized method or a sufficiently generic one regardless of the problem domain since the event data might be considered more complex than sequential text. We claim that since the nature of event data, in general, contains several multi-perspective constraints, such as sequential rules, relational information, concurrency of resources, parallel activities, etc.

Encoding regards mapping data into another representation for different goals. The new space could not allow interpretations and explainability as previously supported by the original data. Moreover, practitioners need to trust the generated mapped space, as mentioned by Elkhawaga et al. (2022). In particular, the predictive model presented in a great part of predictive monitoring tasks does not explain why it provided wrong predictions, so the reason why a prediction model made a mistake cannot be understood. Shedding some light on this topic, Rizzi et al. (2020) presented post-hoc explainers and different encoding methods for identifying the important features. On a general note, our experiments using *expressivity* and *domain agnosticism* confirmed the wide range of representations provided by different encoding methods, even from the same family. Regarding this point of eXplainable Artificial Intelligence (XAI), as a tendency, encoding methods able to provide high explainability levels should be integrated into the PM pipeline as a key component, not a separate step follow-up effort for particular applications.

The significant number of encoding methods and reduced availability of experts pose an additional challenge to selecting and properly setting the encoding method. Strategies focused on accuracy or time performance are applied when selecting a method, but the cost of testing different setups and costly tuning strategies could impact the PM pipeline conception. This problem has been addressed by promising strategies based on meta-learning (Tavares and Junior, 2021; Tavares et al., 2022a; Tavares et al., 2022b), but the current solutions require creating a meta-database containing the history of possible solutions. Also, the criteria to recommend a particular algorithm are still limited to simple performance functions. Therefore, Automatic Machine Learning (AutoML) (Olson and Moore, 2016; Feurer et al., 2019) proves to be an important research area that impacts the aforementioned concerns regarding encoding methods used in PM tasks. Alternatively, another learning paradigm that could be explored for this nature of data is self-supervised learning (SSL). The general idea behind SSL is learning a set of possible outcomes given an input, instead of predicting a unique value as traditional methods. For instance, the *data2vec* was recently presented by Baevski et al. (2022), where the authors propose a generic framework for encoding any type of data, although only the domains of image, speech, and language have been considered. Intuitively, this might be interesting for capturing mutual dependencies in event data.

Finally, we also highlight the need for an automated method for selecting a suitable encoding method and its parameters according to the event log properties and user requirements. Our extensive experimental evaluation showed that although all algorithm families are capable of performing well for each metric, sometimes it is hard to find the right algorithm configuration among a wide range of options.

10. Conclusion

The main contributions presented in this work include a review of process mining tasks using encoding methods, a review and detailed description of encoding methods from other areas never employed in PM before that were categorized into families, and an extensive experimental evaluation and benchmark assessing relevant evaluation metrics to measure the effectiveness of an encoding method with respect to different parameters and data characteristics. We believe this work can support researchers and practitioners to achieve significant accomplishments in different application areas in PM. Furthermore, we stress current challenges and issues in the literature regarding the difficulty of choosing the right algorithm and its parameters. We also discuss how arbitrarily selecting algorithms leads to unfair evaluation and sub-optimal solutions. This is the first work that focuses on a detailed analysis for preprocessing event logs instead of focusing only on the task algorithm itself (i.e. a clustering or learning algorithm).

We also highlighted the need for a better understanding of how each method behaves according to different scenarios of event logs and different PM tasks. Thus, to fill this gap we simulated such scenarios by employing the PLG2 tool to generate synthetic processes with distinct properties and presented the results as a benchmark. In total, 27 encoding methods were evaluated throughout 420 different event logs containing different anomalies. We considered four different evaluation criteria to measure the effectiveness of encoding methods for process mining tasks. We limited our evaluation to only one task, anomaly detection, but the analysis and insights presented in this work can be leveraged for other applications, such as predictive monitoring and clustering.

We conclude this work by stressing the difficulty of choosing suitable algorithms and their parameters according to the user's preferences since each pipeline setting performs differently according to the event log characteristics. This might be a direction to novel automated solutions whether for entire pipelines or preprocessing steps only. We also believe that an encoding method that handles explicitly the nature of processes is essential for advancing the state-of-the-art. This is claimed by considering that most methods are adapted or adopted from other areas. Therefore, this research line is promising and has several opportunities for work to be developed.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Ahmed, N.K., Rossi, R.A., Lee, J.B., Willke, T.L., Zhou, R., Kong, X., Eldardiry, H., 2022. Role-based graph embeddings. IEEE Trans. Knowl. Data Eng. 34 (5), 2401–2415. http://dx.doi.org/10.1109/TKDE.2020.3006475.

Al-Jebrni, A., Cai, H., Jiang, L., 2018. Predicting the next process event using convolutional neural networks. In: International Conference on Progress in Informatics and Computing. (PIC), pp. 332–338. http://dx.doi.org/10.1109/PIC.2018.8706282.

Appice, A., Malerba, D., 2016. A co-training strategy for multiple view clustering in process mining. IEEE Trans. Serv. Comput. 9 (6), 832–845. http://dx.doi.org/10.1109/TSC.2015.2430327.

Baevski, A., Hsu, W., Xu, Q., Babu, A., Gu, J., Auli, M., 2022. Data2vec: A general framework for self-supervised learning in speech, vision and language. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., Sabato, S. (Eds.), International Conference on Machine Learning. (ICML), In: Proceedings of Machine Learning Research, vol. 162, PMLR, pp. 1298–1312.

Barbon Junior, S., Ceravolo, P., Damiani, E., Marques Tavares, G., 2020. Evaluating trace encoding methods in process mining. In: International Symposium: From Data to Models and Back. Springer, pp. 174–189.

Belkin, M., Niyogi, P., 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In: International Conference on Neural Information Processing Systems: Natural and Synthetic (NIPS'01). NIPS '01, MIT Press, Cambridge, MA, USA, pp. 585–591.

Berti, A., van der Aalst, W.M., 2019. Reviving token-based replay: Increasing speed while improving diagnostics. In: ATAED@ Petri Nets/ACSD. pp. 87–103.

Bezerra, F., Wainer, J., 2013. Algorithms for anomaly detection of traces in logs of process aware information systems. Inf. Syst. 38 (1), 33–44.

Burattin, A., 2015. PLG2: Multiperspective processes randomization and simulation for online and offline settings. arXiv:1506.08415.

Camargo, M., Dumas, M., Rojas, O.G., 2019. Learning accurate LSTM models of business processes. In: Hildebrandt, T.T., van Dongen, B.F., Röglinger, M., Mendling, J. (Eds.), Business Process Management. (BPM), In: Lecture Notes in Computer Science, vol. 11675, Springer, pp. 286–302. http://dx.doi.org/10.1007/978-3-030-26619-6_19.

Cao, S., Lu, W., Xu, Q., 2015. GraRep: Learning graph representations with global structural information. In: International on Conference on Information and Knowledge Management (CIKM). CIKM '15, Association for Computing Machinery, New York, NY, USA, pp. 891–900. http://dx.doi.org/10.1145/2806416.2806512.

Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M., 2018. Conformance Checking - Relating Processes and Models. Springer, pp. 1–263. http://dx.doi.org/10.1007/978-3-319-99414-7.

Ceravolo, P., Damiani, E., Torabi, M., Barbon Junior, S., 2017. Toward a new generation of log pre-processing methods for process mining. In: Carmona, J., Engels, G., Kumar, A. (Eds.), Business Process Management Forum. (BPM), In: Lecture Notes in Business Information Processing, vol. 297, Springer, pp. 55–70. http://dx.doi.org/10.1007/978-3-319-65015-9_4.

Ceravolo, P., Tavares, G.M., Junior, S.B., Damiani, E., 2020. Evaluation goals for online process mining: A concept drift perspective. IEEE Trans. Serv. Comput..

Chiorrini, A., Diamantini, C., Genga, L., Pioli, M., Potena, D., 2022. Embedding process structure in activities for process mapping and comparison. In: Chiusano, S., Cerquitelli, T., Wrembel, R., rvåg, K.N., Catania, B., Vargas-Solar, G., Zumpano, E. (Eds.), New Trends in Database and Information Systems (ADBIS), Vol. 1652. Springer, pp. 119–129. http://dx.doi.org/10.1007/978-3-031-15743-1_12.

Daffertshofer, A., Lamoth, C.J., Meijer, O.G., Beek, P.J., 2004. PCA in studying coordination and variability: A tutorial. Clin. Biomech. 19 (4), 415–428.

Donnat, C., Zitnik, M., Hallac, D., Leskovec, J., 2018. Learning structural node embeddings via diffusion wavelets. In: International Conference on Knowledge Discovery and Data Mining (SIGKDD). KDD '18, Association for Computing Machinery, New York, NY, USA, pp. 1320–1329. http://dx.doi.org/10.1145/3219819.3220025.

Elkhawaga, G., Abu-Elkheir, M., Reichert, M., 2022. Explainability of predictive process monitoring results: Can you see my data issues? Appl. Sci. 12 (16), 8192.

Evermann, J., Rehse, J., Fettke, P., 2016. A deep learning approach for predicting process behaviour at runtime. In: Dumas, M., Fantinato, M. (Eds.), Business Process Management Workshops. (BPM), In: Lecture Notes in Business Information Processing, vol. 281, pp. 327–338. http://dx.doi.org/10.1007/978-3-319-58457-7_24.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F., 2019. Auto-sklearn: Efficient and robust automated machine learning. In: Hutter, F., Kotthoff, L., Vanschoren, J. (Eds.), Automated Machine Learning - Methods, Systems, Challenges. In: The Springer Series on Challenges in Machine Learning, Springer, pp. 113–134. http://dx.doi.org/10.1007/978-3-030-05318-5_6.

Francescomarino, C.D., Dumas, M., Maggi, F.M., Teinemaa, I., 2019. Clustering-based predictive process monitoring. IEEE Trans. Serv. Comput. 12 (6), 896–909. http://dx.doi.org/10.1109/TSC.2016.2645153.

Francescomarino, C.D., Ghidini, C., Maggi, F.M., Petrucci, G., Yeshchenko, A., 2017. An eye into the future: Leveraging A-priori knowledge in predictive business process monitoring. In: Carmona, J., Engels, G., Kumar, A. (Eds.), Business Process Management. (BPM), In: Lecture Notes in Computer Science, vol. 10445, Springer, pp. 252–268. http://dx.doi.org/10.1007/978-3-319-65000-5_15.

Gasparetto, A., Marcuzzo, M., Zangari, A., Albarelli, A., 2022. A survey on text classification algorithms: From text to predictions. Information 13 (2), 83. http://dx.doi.org/10.3390/info13020083.

Goldberg, Y., 2016. A primer on neural network models for natural language processing. J. Artificial Intelligence Res. 57, 345–420. http://dx.doi.org/10.1613/jair.4992.

Goyal, P., Ferrara, E., 2018. Graph embedding techniques, applications, and performance: A survey. Knowl.-Based Syst. 151, 78–94. http://dx.doi.org/10.1016/j.knosys.2018.03.022.

Grover, A., Leskovec, J., 2016. Node2vec: Scalable feature learning for networks. In: International Conference on Knowledge Discovery and Data Mining (SIGKDD). KDD '16, Association for Computing Machinery, New York, NY, USA, pp. 855–864. http://dx.doi.org/10.1145/2939672.2939754.

Hompes, B., Buijs, J., van der Aalst, W., Dixit, P., Buurman, J., 2015. Discovering deviating cases and process variants using trace clustering. In: Benelux Conference on Artificial Intelligence. (BNAIC).

Kim, H., Teh, Y.W., 2018. Scaling up the automatic statistician: Scalable structure discovery using Gaussian processes. In: Storkey, A.J., Pérez-Cruz, F. (Eds.), International Conference on Artificial Intelligence and Statistics. (AISTATS), In: Proceedings of Machine Learning Research, vol. 84, PMLR, pp. 575–584.

Koninck, P.D., vanden Broucke, S., Weerdt, J.D., 2018. Act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (Eds.), Business Process Management. (BPM), In: Lecture Notes in Computer Science, vol. 11080, Springer, pp. 305–321. http://dx.doi.org/10.1007/978-3-319-98648-7_18.

Kratsch, W., Manderscheid, J., Röglinger, M., Seyfried, J., 2021. Machine learning in business process monitoring: A comparison of deep learning and classical approaches used for outcome prediction. Bus. Inf. Syst. Eng. 63 (3), 261–276. http://dx.doi.org/10.1007/s12599-020-00645-0.

Le, Q., Mikolov, T., 2014. Distributed representations of sentences and documents. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Vol. 32. ICML '14, JMLR.org, pp. II–1188–II–1196.

Li, J., Wu, L., Guo, R., Liu, C., Liu, H., 2019. Multi-level network embedding with boosted low-rank matrix approximation. In: Spezzano, F., Chen, W., Xiao, X. (Eds.), International Conference on Advances in Social Networks Analysis and Mining. (ASONAM), ACM, pp. 49–56. http://dx.doi.org/10.1145/3341161.3342864.

Lin, L., Wen, L., Wang, J., 2019. MM-pred: A deep predictive model for multi-attribute event sequence. In: Berger-Wolf, T.Y., Chawla, N.V. (Eds.), International Conference on Data Mining (SDM), SDM 2019, Calgary, Alberta, Canada, May 2-4, 2019. SIAM, pp. 118–126. http://dx.doi.org/10.1137/1.9781611975673.14.

Lorena, A.C., Garcia, L.P., Lehmann, J., Souto, M.C., Ho, T.K., 2019. How complex is your classification problem? A survey on measuring classification complexity. ACM Comput. Surv. 52 (5), 1–34.

Luhn, H.P., 1958. The automatic creation of literature abstracts. IBM J. Res. Dev. 2 (2), 159–165. http://dx.doi.org/10.1147/rd.22.0159.

Mauro, N.D., Appice, A., Basile, T.M.A., 2019. Activity prediction of business process instances with inception CNN models. In: Alviano, M., Greco, G., Scarcello, F. (Eds.), Advances in Artificial Intelligence. (AI*IA), In: Lecture Notes in Computer Science, vol. 11946, Springer, pp. 348–361. http://dx.doi.org/10.1007/978-3-030-35166-3_25.

Mehdiyev, N., Mayer, L., Lahann, J., Fettke, P., 2022. Deep learning-based clustering of processes and their visual exploration: An industry 4.0 use case for small, medium-sized enterprises. Expert Syst. n/a (n/a), e13139. http://dx.doi.org/10.1111/exsy.13139.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013a. Efficient estimation of word representations in vector space. In: Bengio, Y., LeCun, Y. (Eds.), International Conference on Learning Representations. (ICLR).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013b. Distributed representations of words and phrases and their compositionality. In: Burges, C.J.C., Bottou, L., Ghahramani, Z., Weinberger, K.Q. (Eds.), Neural Information Processing Systems. (NIPS), pp. 3111–3119.

Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M., 2019. BINet: Multi-perspective business process anomaly classification. Inf. Syst. 101458.

Olson, R.S., Moore, J.H., 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In: Hutter, F., Kotthoff, L., Vanschoren, J. (Eds.), Workshop on Automatic Machine Learning (AutoML), Co-Located with ICML. In: JMLR Workshop and Conference Proceedings, vol. 64, JMLR.org, pp. 66–74.

Ou, M., Cui, P., Pei, J., Zhang, Z., Zhu, W., 2016. Asymmetric transitivity preserving graph embedding. In: Krishnapuram, B., Shah, M., Smola, A.J., Aggarwal, C.C., Shen, D., Rastogi, R. (Eds.), International Conference on Knowledge Discovery and Data Mining. (SIGKDD), ACM, pp. 1105–1114. http://dx.doi.org/10.1145/2939672.2939751.

Pasquadibisceglie, V., Appice, A., Castellano, G., Fiorentino, N., Malerba, D., 2022. STARDUST: A novel process mining approach to discover evolving models from trace streams. IEEE Trans. Serv. Comput. 1–14. http://dx.doi.org/10.1109/TSC.2022.3215502.

Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D., 2019. Using convolutional neural networks for predictive process analytics. In: International Conference on Process Mining. (ICPM), IEEE, pp. 129–136. http://dx.doi.org/10.1109/ICPM.2019.00028.

Pennington, J., Socher, R., Manning, C.D., 2014. Glove: Global vectors for word representation. In: Empirical Methods in Natural Language Processing. (EMNLP), pp. 1532–1543.

Perozzi, B., Al-Rfou, R., Skiena, S., 2014. DeepWalk: online learning of social representations. In: Macskassy, S.A., Perlich, C., Leskovec, J., Wang, W., Ghani, R. (Eds.), International Conference on Knowledge Discovery and Data Mining. (SIGKDD), ACM, pp. 701–710. http://dx.doi.org/10.1145/2623330.2623732.

Perozzi, B., Kulkarni, V., Chen, H., Skiena, S., 2017. Don't walk, skip! online learning of multi-scale network embeddings. In: International Conference on Advances in Social Networks Analysis and Mining (ASONAM). ASONAM '17, Association for Computing Machinery, New York, NY, USA, pp. 258–265. http://dx.doi.org/10.1145/3110025.3110086.

Polato, M., Sperduti, A., Burattin, A., de Leoni, M., 2018. Time and activity sequence prediction of business process instances. Computing 100 (9), 1005–1031. http://dx.doi.org/10.1007/s00607-018-0593-x.

Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., Tang, J., 2018. Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In: International Conference on Web Search and Data Mining (WSDM). WSDM '18, Association for Computing Machinery, New York, NY, USA, pp. 459–467. http://dx.doi.org/10.1145/3159652.3159706.

Rama-Maneiro, E., Vidal, J., Lama, M., 2021. Deep learning for predictive business process monitoring: Review and benchmark. IEEE Trans. Serv. Comput. 1. http://dx.doi.org/10.1109/TSC.2021.3139807.

Rizzi, W., Francescomarino, C.D., Maggi, F.M., 2020. Explainability in predictive process monitoring: When understanding helps improving. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (Eds.), Business Process Management Forum - BPM. In: Lecture Notes in Business Information Processing, vol. 392, Springer, pp. 141–158. http://dx.doi.org/10.1007/978-3-030-58638-6_9.

Rozemberczki, B., Sarkar, R., 2020. Fast sequence-based embedding with diffusion graphs. CoRR arXiv:2001.07463.

Russell, N., ter Ahm Arthur Hofstede, van der Aalst, W.M., Mulyar, N.N., 2006. Workflow Control-Flow Patterns: A Revised View.

Sasaki, Y., et al., 2007. The truth of the F-measure. Teach. Tutor. Mater. 1 (5), 1–5.

Senderovich, A., Francescomarino, C.D., Maggi, F.M., 2019. From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring. Inf. Syst. 84, 255–264. http://dx.doi.org/10.1016/j.is.2019.01.007.

Sun, D.L., Févotte, C., 2014. Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence. In: International Conference on Acoustics, Speech and Signal Processing. (ICASSP), pp. 6201–6205. http://dx.doi.org/10.1109/ICASSP.2014.6854796.

Tavares, G.M., Barbon, S., 2020. Analysis of language inspired trace representation for anomaly detection. In: Bellatreche, L., Bieliková, M., Boussaïd, O., Catania, B., Darmont, J., Demidova, E., Duchateau, F., Hall, M., Merčun, T., Novikov, B., Papatheodorou, C., Risse, T., Romero, O., Sautot, L., Talens, G., Wrembel, R., Žumer, M. (Eds.), ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium. Springer International Publishing, Cham, pp. 296–308.

Tavares, G.M., Barbon Junior, S., Damiani, E., Ceravolo, P., 2022a. Selecting optimal trace clustering pipelines with meta-learning. In: Brazilian Conference on Intelligent Systems. (BRACIS), Springer, pp. 150–164.

Tavares, G.M., Ceravolo, P., Da Costa, V.G.T., Damiani, E., Junior, S.B., 2019. Overlapping analytic stages in online process mining. In: International Conference on Services Computing. (SCC), IEEE, pp. 167–175.

Tavares, G.M., Junior, S.B., 2021. Process mining encoding via meta-learning for an enhanced anomaly detection. In: European Conference on Advances in Databases and Information Systems. Springer, pp. 157–168.

Tavares, G.M., Junior, S.B., Damiani, E., 2022b. Automating process discovery through meta-learning. In: Sellami, M., Ceravolo, P., Reijers, H.A., Gaaloul, W., Panetto, H. (Eds.), Cooperative Information Systems. (CoopIS), Springer International Publishing, Cham, pp. 205–222.

Tax, N., Verenich, I., Rosa, M.L., Dumas, M., 2017. Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (Eds.), Conference on Advanced Information Systems Engineering. (CAiSE), In: Lecture Notes in Computer Science, vol. 10253, Springer, pp. 477–492. http://dx.doi.org/10.1007/978-3-319-59536-8_30.

Taymouri, F., Rosa, M.L., Erfani, S.M., 2021. A deep adversarial model for suffix and remaining time prediction of event sequences. In: Demeniconi, C., Davidson, I. (Eds.), International Conference on Data Mining. (SDM), SIAM, pp. 522–530. http://dx.doi.org/10.1137/1.9781611976700.59.

Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M., 2019. Outcome-oriented predictive process monitoring: Review and benchmark. ACM Trans. Knowl. Discov. Data 13 (2), 17:1–17:57.

Torres, L., Chan, K.S., Eliassi-Rad, T., 2020. GLEE: geometric Laplacian eigenmap embedding. J. Complex Netw. 8 (2), http://dx.doi.org/10.1093/comnet/cnaa007.

van der Aalst, W.M.P., 2016. Process Mining - Data Science in Action, Second Edition. Springer, http://dx.doi.org/10.1007/978-3-662-49851-4.

Venugopal, I., Töllich, J., Fairbank, M., Scherp, A., 2021. A comparison of deep-learning methods for analysing and predicting business processes. In: International Joint Conference on Neural Networks. (IJCNN), IEEE, pp. 1–8. http://dx.doi.org/10.1109/IJCNN52387.2021.9533742.

Verbeek, H.M.W., de Carvalho, R.M., 2018. Log skeletons: A classification approach to process discovery. CoRR arXiv:1806.08247.

Weiss, S.M., Indurkhya, N., Zhang, T., 2015. Fundamentals of Predictive Text Mining, Second Edition In: Texts in Computer Science, Springer, http://dx.doi.org/10.1007/978-1-4471-6750-1.

Yang, D., Rosso, P., Li, B., Cudre-Mauroux, P., 2019. NodeSketch: Highly-efficient graph embeddings via recursive sketching. In: International Conference on Knowledge Discovery and Data Mining (SIGKDD). KDD '19, Association for Computing Machinery, New York, NY, USA, pp. 1162–1172. http://dx.doi.org/10.1145/3292500.3330951.