Evaluating Trace Encoding Methods in Process Mining

Sylvio Barbon Junior[1], Paolo Ceravolo[2], Ernesto Damiani[3], and Gabriel Marques Tavares[2(✉)]

[1] Londrina State University (UEL), Londrina, Brazil
barbon@uel.br
[2] Università degli Studi di Milano (UNIMI), Milan, Italy
{paolo.ceravolo,gabriel.tavares}@unimi.it
[3] Khalifa University (KUST), Abu Dhabi, UAE
ernesto.damiani@kustar.ac.ae

Abstract. Encoding methods affect the performance of process mining tasks but little work in the literature focused on quantifying their impact. In this paper, we compare 10 different encoding methods from three different families (trace replay and alignment, graph embeddings, and word embeddings) using measures to evaluate the overlaps in the feature space, the accuracy obtained, and the computational resources (time) consumed with a classification task. Across hundreds of event logs representing four variations of five scenarios and five anomalies, it was possible to identify the edge2vec method as the most accurate and effective in reducing class overlapping in the feature space.

Keywords: Trace encoding · Word embeddings · Graph embeddings · Classification · Process Mining

1 Introduction

Process Mining (PM) is aimed at extracting knowledge from business process event logs. Trace encoding, i.e. encoding the sequence of events in a case, is then a crucial stage for any PM task [8]. Event logs incorporate multiple information such as activity sequences, time spans, dependency between activities or attribute values, replaceability between activities or resources, concurrent or iterative behavior, and others [3,11,12,19] that can be hardly summarized in a single representation. Encoding transforms this information into a feature space enabling data processing. For this reason, the choice of the encoding method can drive the successful implementation of PM tasks. A bad encoding creates ambiguity, sparsity and complex separation boundaries [17,20]. A good encoding

This study was financed in part by Coordination for the National Council for Scientific and Technological Development (CNPq) of Brazil - Grant of Project 420562/2018-4 and Fundação Araucária (Paraná, Brazil). It was also partly supported by the program "Piano di sostegno alla ricerca 2019" funded by Università degli Studi di Milano.

boosts performances by correctly and effectively discriminating traces and the impact on computational costs.

The machine learning community has deeply discussed the relationship between data encoding and the complexity of classification tasks. For example, Ho and Basu, in [17], studied several properties, such as class ambiguity, data sparsity, non-discriminative features, and the intrinsic complexity of class separation boundaries. Lorena et al. [20], grouped some of these properties to support measures of complexity of the classification problems. The Maximum Fisher's Discriminant Ratio (F1) and the Volume of Overlapping Region (F2) were suggested to measure how effectively the feature vectors can separate classes. F1 uses the largest discriminating ratio among all the dimensions provided by the encoding method, indicating if the problem classes can be separable using this high-discriminant feature. F2 is related to the overlapping intervals between the problem classes [10]. The Average Number of Principal Component Analysis (PCA) dimensions compared to the original dimensions (T4) can be used for evaluating dimensionality [20].

Despite trace encoding is widely discussed in the PM community [3,11,12,19], to the best of our knowledge, we lack a study on the quality achieved by the different methods proposed in the literature. In this work, we compared 10 different encoding methods (alignment, trace replay, edge2vec, node2vec, fasttext, tfidf, count2vec, word2vec, one-hot, and hash2vec) representative of both traditional PM methods, such a trace replay and alignment, and methods producing highly informative but low-dimensional vectors such as graph embeddings and word embeddings. These methods are compared against a classification problem as it directly relates to well-known PM tasks such as trace clustering and anomaly detection. More specifically, the Random Forest algorithm was employed to measure accuracy and time for binary classification of event logs. The F1, F2, and T4 measures were used to assess the quality of the compared encoding methods. The classes to be identified were common PM anomalies (early, insert, late, rework, and skip) with four different shares across five different scenarios, analyzing a total of 100 event logs.

Our results bring important insights into the optimal representation of traces in PM. More specifically, the paper starts by presenting the relevance of trace encoding in the PM literature (Sect. 2). Then, we expand on the encoding methods evaluated (Sect. 3). Section 4 presents the event logs and experiments, along with an application scenario in the classification domain. The results are presented and discussed in Sect. 5. Final remarks and conclusions are presented in Sect. 6.

2 Related Work

As process data may be analyzed according to multiple perspectives, there exists a considerable amount of encoding techniques that could be applied to event logs. Traditional PM goals, such as discovering a process model, require some level of abstractions to represent information, which can be achieved by encodings. In process discovery techniques, relations from the log are extracted and

transformed into other forms of representations, such as directly-follows mappings [27]. Leontjeva et al. [19] presented a complex sequence encoder based on indexes and hidden Markov models encoding. A last state encoding method was proposed by Polato et al. [24] for time and activity prediction in processes. Inspired by natural language processing research, Koninck et al. [11] applied word2vec and doc2vec for representational learning in business processes. The authors adapted these encodings to work on several business process layers, activities, traces, logs, and models. Also using word embeddings to learn representations, Hake et al. [16] combined word2vec and recurrent neural networks to label nodes in business process models.

Recently, graph embedding techniques have been proposed to encode information structured as graphs [15]. These techniques are suitable in PM environments as graphs can represent business process models, where nodes and edges are activities and directly-follows relations. A graph representation of the business process can be compared to graph representations of the traces similarly to traditional PM conformance checking methods. Furthermore, graph embeddings open new possibilities for PM analysis, such as capturing graph structures and finding similarities across different process models.

Nolle et al. [22] used autoencoders for the assessment of anomalies. Autoencoder is a class of neural networks trained to copy its input to an output that preserves the input's probability density function [14]. For that, it learns the input-output mapping ignoring the noise. The authors' method consists of first transforming the event log using the one-hot encoding technique. Then, the autoencoder is trained with back-propagation using the event log both as input and output label. However, the vector size increases linearly with the number of activities, meaning that complex processes are encoded in huge dimensions. Also, autoencoders involve relevant computational costs that limit their application.

3 Encoding Methods

To limit the scope of our work, we decided to investigate the control-flow perspective, that is, the selected encoding methods are pertinent to the analysis of the sequence of executed activities. Moreover, autoencoders were not considered due to their extremely high computational costs, making applications in business processes difficult. The selected methods can be organized into three main groups: trace replay and alignment, word embeddings, and graph embeddings. Table 1 details the encoding methods we studied with their features, types, and ranges.

3.1 Trace Replay and Alignment

Most PM quality measures are based on conformance checking methods, which aim at comparing a process execution to a process model [25]. The measures produced by conformance checking techniques can be interpreted as features.

Table 1. Encoding characteristics, produced features and their possible values

Encoding	Family	Feature	Type	Range
trace replay	PM-based	trace is fit	Boolean	{True, False}
		fitness	Numeric	[0, 1]
		consumed tokens	Integer	[0, ∞[
		remaining tokens	Integer	[0, ∞[
		produced tokens	Integer	[0, ∞[
alignment	PM-based	cost	Integer	[0, ∞[
		visited states	Integer	[0, ∞[
		queued states	Integer	[0, ∞[
		traversed arcs	Integer	[0, ∞[
		fitness	Numeric	[0, 1]
word2vec	Text-based	n-dimensions*	Numeric	]−∞, ∞[
fasttext	Text-based	n-dimensions*	Numeric	]−∞, ∞[
count2vec	Text-based	n-dimensions**	Integer	[0, ∞[
one-hot	Text-based	n-dimensions**	Integer	{0, 1}
tfidf	Text-based	n-dimensions**	Numeric	[0, 1]
hash2vec	Text-based	n-dimensions*	Numeric	[−1, 1]
node2vec	Graph-based	n-dimensions*	Numeric	]−∞, ∞[
edge2vec	Graph-based	n-dimensions*	Numeric	]−∞, ∞[

* encoding vector size is determined by a parameter
** encoding vector size is determined by the vocabulary size

More specifically, we exploited two conformance checking algorithms to encode traces: trace replay and trace alignment.

Trace Replay. These techniques replay traces into a model trying to consume the executed activities according to the constraints imposed by the model. By counting the missing and remaining activities, a measure of the conformance is produced [6].

Trace Alignment. These techniques also perform a comparison between a model and a trace but directly relate a trace to the valid execution sequences, i.e. allowed by the model [6]. Ultimately, an alignment can be seen as a sequence of moves that can be synchronous if originated from both the model and the trace, model-dependent if originated from the model only, or log-dependent if originated from the trace only. It follows that more than one alignment is possible when comparing a trace to a model. Thus, the technique aims at finding an optimal alignment, minimizing the number of model- and log-moves, which are measured by a cost function.

3.2 Word Embeddings

Word embeddings are grounded in information retrieval and natural language processing. Neural network algorithms are exploited to create highly informative but low-dimensional vectors modeling the context in which words of a corpus are inserted. We applied the following text-based encodings: word2vec, fasttext, count2vec, one-hot, tfidf and hash2vec.

Word2vec. The word embeddings come from the weights of a two-layer neural network created to reconstruct the linguistic context of words in a corpus [21]. This way, words appearing in similar contexts generate more similar encodings than words present in different contexts. In the process domain, a trace can be described by its sequence of activities, which can be treated as words in a corpus. From this perspective, a trace is a sentence and a log is a text, i.e., a sequence of sentences. Consequently, the trace encoding is the aggregation of its activities encodings, which is obtained by their mean.

Fasttext. Fasttext represents each word as a bag of n-gram characters, trying to capture the morphemes of a corpus. The final vector representation of a word is, then, retrieved by the sum of its n-gram character representations [2]. Given this construction, the method performs well in the representation of rare words and can generate encodings for words that do not appear in the training data.

Count2vec. The count vectorizer is a simple way of encoding words by accounting for their frequencies in a text document. This tokenization process outputs a matrix of word counts. The length of the features is determined by the number of unique words in the document. For this method, the event log is interpreted as a document and the activity frequency regulates the resulting feature vector for each trace.

One-Hot. The one-hot encoding technique encodes categorical values in a binary representation. For that, it first maps words into integers and, then, transforms the generated integer values to a binary value. Like count2vec, the number of dimensions linearly increases with the vocabulary size, with the tendency of generating sparse features.

Tdidf. Term frequency-inverse document frequency (tdidf) is a traditional information retrieval metric aimed at capturing the importance of a word in a document given a collection of documents. The term frequency weights a term occurrence proportionally to its frequency in a document. The inverse document frequency quantifies the importance of a term as the inverse function of its occurrence across a collection of documents.

Hash2vec. Tdidf creates a dictionary of words, which increases linearly to the vocabulary size, often generating large and sparse representations. To overcome this issue, the hash2vec maps a feature into an index (word) using a hash function. Then, word frequencies are computed based on previously mapped indices. The technique allows a vector of a predetermined size, on the other hand, if a small vector size is used, hash collisions, where different words are represented by the same index, can take place [28].

3.3 Graph Embeddings

Graph embeddings emerged from the necessity of modeling more complex relations, such as entity links and long-term relations. Graphs are suitable for this task due to their data representation format, enabling exploration of nodes and edges. We applied two versions of node2vec: one encodes the nodes, while the other encodes the edges.

Node2vec. Built on top of word2vec, node2vec aims at encoding graph data while preserving neighborhoods and structures. The low-dimensional node representations are based on second-order random walks that propose a trade-off between breadth and width searches, exploring neighbors and neighborhoods. The flexibility of node exploration allows for a richer representation of diverse neighborhoods.

Edge2vec. Edge2vec captures the links (edges) that connect nodes. For our evaluation, this behavior is interesting as process models can be represented as graphs. This way, by grouping the edges representations, we can generate another encoding using the same method.

4 Materials and Methods

This section presents the event logs, the experimental setup, and the quality metrics used in our experimental analysis. Generated event logs and code for experiments are publicly available[1], following open-science principles.

4.1 Event Logs

Our experimental design implies relying on labeled data providing the ground truth for the evaluation of the compared methods. We then generated synthetic event logs following standard practices in PM research and injecting anomalies to the generated traces. This way we achieved two goals. Traces are labeled as anomalous or normal, making our data set suitable for supervised learning. Heterogeneous behaviors are introduced in the event logs, making our data set more realistic.

First, five different process models were generated using the PLG2 tool [5]. PLG2 performs a random generation of process models capable of representing several business process behaviors such as sequential, parallel, and iterative control flows. For that, the tool combines traditional control-flow patterns [26], e.g., sequence, parallel split and synchronization. The patterns are progressively combined, given a predetermined set of rules, to simulate real-world scenarios. The five generated process models define five different base scenarios differing because of the number of activities and gateways [9]. The next step was to simulate the process model to generate the log. For that, we applied the *Perform a simple*

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

simulation of a (stochastic) Petri net ProM plug-in[2]. The number of simulated cases was set to 1000, and the arrival rate of new cases was set to 30 min. The other hyperparameters were unchanged. As a post-processing step, we injected anomalies by perturbing regular traces. Injecting anomalies into event logs is a common practice in the literature [1]. For that, we applied the anomalies proposed by Nolle et al. [23]: 1. skip: a sequence of 3 or less necessary events is skipped;2. insert: 3 or less random activities are inserted in the case;3. rework: a sequence of 3 or less necessary events is executed twice;4. early: a sequence of 2 or fewer events executed too early, which is then skipped later in the case;5. late: a sequence of 2 or fewer events executed too late.

The anomalies were applied in normal traces, replacing their occurrence. Moreover, to analyze to which extent anomalies affect the encodings, we injected different percentages of anomalies for each scenario: 5%, 10%, 15%, and 20%. Given five scenarios (our base models), five anomalies, and four anomaly percentages, a total of 100 event logs were generated. To facilitate the interpretation of the logs, we added two additional attributes: label and description. Case labels represent if a case belongs to a normal execution or one of the anomalous types. Furthermore, the description is a natural language sentence describing the anomaly and its impact on the case. Descriptive statistics about the generated event logs are listed in Table 2. The different scenarios are of increasing complexity, scenario 3 contains the longest traces and, consequently, logs composed of more events.

Table 2. Event log statistics demonstrating different levels of complexity. 20 event logs with 1k cases were generated for each scenario

Log name	#gateways	#events	trace size	#activities
scenario 1	8	10k–11k	9–13	22
scenario 2	12	26k	26–30	41
scenario 3	22	43k–44k	42–50	64
scenario 4	30	11k–13k	3–30	83
scenario 5	34	18k–19k	4–37	103

4.2 Trace Encoding

Since trace replay and alignment require a process model, we generated a model using the Inductive Miner Directly Follows algorithm [18]. Process model and encodings were extracted using the PM4Py library[3]. For word embeddings, we used the Gensim[4] library to compute word2vec and fasttext and the Scikit-learn

[2] http://www.promtools.org/doku.php.
[3] https://pm4py.fit.fraunhofer.de/.
[4] https://radimrehurek.com/gensim/.

library[5] compute the remaining encodings. For the graph embeddings, a graph model is expected as input. Thus, we generated a directly-follows graph using the event log to capture node and edge frequency. The encodings were extracted with the node2vec[6] library. For all encoding methods, the recommended standard hyperparameters were used.

4.3 Feature Vector Measures and Classification Algorithm

In our experiments, we computed F1, F2 and T4 measures using the ECoL (*Extended Complexity Library*) R package, available at Github[7] and CRAN[8], using standard hyperparameters. Although multiple PM tasks could exploit trace encoding, we drove our evaluation using a binary classification task for anomaly detection. This is a basic supervised approach that can be easily evaluated and whose connections to other tasks are well known. We used the Random Forest classification algorithm [4] following the Scikit-learn implementation with standard parameters. The Random Forest was chosen due to its high predictive performance and wide use in related papers. The traditional holdout method was used to divide the data into train and test sets, with an 80%/20% proportion. Each classification was performed 30 times to compute a mean accuracy value, eliminating possible eccentric performances. Moreover, the meantime consumption for the performed executions was computed.

5 Results and Discussion

This section presents the results obtained in evaluating the impact of the studied encoding methods from several complementary perspectives.

5.1 Accuracy Performance

One of the main goals when choosing an encoding method is to support high predictive performance. Figure 1 presents the accuracy results (along with their standard deviation) aggregated over the event logs of all the scenarios presented in Table 2. *Trace replay* and *alignment* methods obtained very similar results, an average accuracy of 91.93% and 92.62%, respectively. The word embedding family obtained average accuracy varying from 88.72% (*one-hot*) to 94.16% (*tfidf*), with the latter being the best performing text-based encoding. The best performances were achieved with methods of the graph embedding family. *Node2vec* reached an average accuracy of 94.18% while *edge2vec* obtained 96.08%, the best overall performance.

Trace replay and *alignment* methods rely on the comparison of an event log to a model. Since the model is induced from the event log, anomalies may

[5] https://scikit-learn.org/stable/.
[6] https://github.com/eliorc/node2vec.
[7] https://github.com/lpfgarcia/ECoL.
[8] https://cran.r-project.org/package=ECoL.

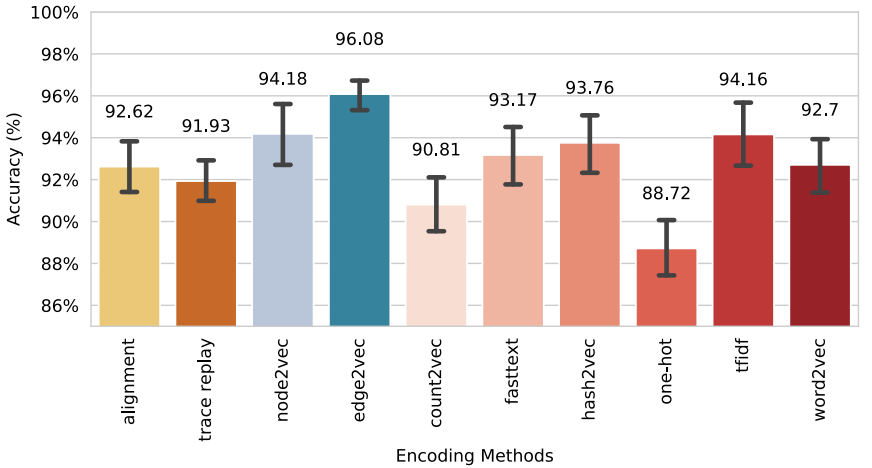

Fig. 1. Average accuracy obtained using all encoding methods across binary problems related to anomaly detection (early, insert, late, rework and skip) affected by four different levels of compromised samples (5%, 10%, 15% and 20%).

have been modeled as normal transitions. Ideally, a model could be constructed from a filtered event log, without anomalies. However, in reality, often event data is not labeled, and manually detecting anomalies is a resource-consuming task. Nonetheless, these methods produce the most interpretable features, easily understandable by process stakeholders. This characteristic has gained more attention in data mining research, as black-box models may not offer sufficient basis for their choices. Overall, the trade-off between performance and interpretability plays an important role when applying trace encodings.

Word embeddings present a wider range of performances. *One-hot* encoding and *count2vec* appear with the worst results. Both methods are grounded in word frequencies and fail to encode global information, such as accounting differences between traces in the same log. Besides, the ordering of the traces is lost by these methods. This way, counting frequencies demonstrate to be a shallow method that does not meet business process modeling requirements. *Word2vec* and *fasttext*, which are more recent advancements in text processing, capture activities context by considering their neighborhood. These methods allow for a better overall trace description and, consequently, higher accuracy values. Moreover, *fasttext* performs slightly better than *word2vec*, probably a result of its consideration of n-grams when encoding a word. *Hash2vec* and *tfidf* are the best performing methods within this family. Both methods propose a frequency analysis that also covers inter-trace behavior, i.e., global event log characteristics. Even though these encodings do not consider the ordering, their performance surpasses methods that capture context information. This implies that trace context, i.e., activities neighborhood, from a text analysis perspective, is not so determinant as a descriptor when compared to weighted frequencies. A possible

explanation for the inferior performance obtained by *word2vec* and *fasttext* is that these methods require a rich corpus for training their models. According to Table 2, the richest log, in the number of unique activities, only contains 103 words. This highly limits the capacity of capturing context information. In most cases, the set of activities in a business process is considerably smaller than the vocabulary of a document collection. This way, in the business process domain, modern word embedding techniques are not necessarily the best. Finally, the length of the event log also plays a role in this performance since a higher number of traces may increase the encoding quality of context-based methods.

The graph embedding family was the best performing for the classification task. *Node2vec* and *edge2vec* are built on top of word2vec, thus, the goal is also to capture context information. Further, the graph structure is capable of representing many complex behaviors. Therefore, their performance overcomes all other families. Within graph embeddings, *edge2vec* displays an accuracy considerably higher than *node2vec*. This happens because *node2vec* is limited to encode node (activity) behavior only. On the other hand, *edge2vec* encodes the connections within activities and long-term relations are captured.

5.2 Time Usage

Time costs of an encoding method can directly influence its selection since costly methods are prohibitive to real-life event logs with huge volume of data. In our experiments, we considered the time consumed during the classification task. Figure 2 presents the time variation among all methods. *Trace replay* and *alignment* obtained similar results, with trace replay (0.258 s) being the fastest and most stable method. Graph embeddings were the most time-costly, with an average of 0.349 s. The *edge2vec* method, which uses edge information, was the slowest, spending an average of 0.391 s. Text-encoding family reveals to be faster, except for the *word2vec* method that required an average of 0.341 s and resulted the most unstable, obtaining the highest standard deviation (0.07 s).

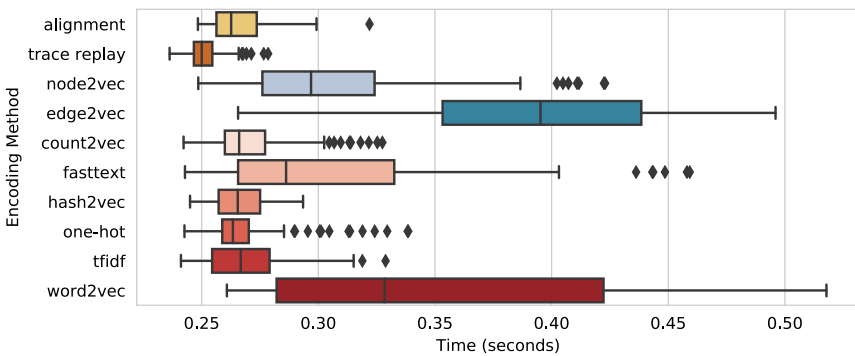

Fig. 2. Average time required in the classification task for each trace encoding across all scenarios.

It is important to note that concerning the time dispensed to perform the encoding procedure, our results confirmed what is well known in the literature. However, due to space limits, we do not report these results in detail in this paper. *Trace replay* and *alignment* are time costly, mainly alignment [13], followed by graph embeddings [15]. The high cost of alignment is related to the multi-step approximation required to find an optimal alignment. Moreover, the alignment procedure has a computational complexity that grows exponentially to the number of states and transitions, becoming impractical in most scenarios. On the other hand, when dealing with graph embeddings representation, *node2vec* generates random walks, which require several iterations (time perspective). Methods from the word embedding family, such as *one-hot* and *fasttext* are less costly and can be employed in tasks focused on light-weight processing, such as online PM [7].

5.3 Encoding Representativeness

The capacity to represent knowledge towards providing low ambiguity between classes and reduce the inherent complexity to the problem guides a high-quality encoding method. Moreover, this capacity is made by constructing a short and highly informative feature vector. In our experiments, we measured the Maximum Fisher's Discriminant Ratio (F1), the Volume of Overlapping Region (F2), and the Ratio of the PCA dimensions to the original dimensions (T4).

Using F1, we can assess how informative the encoding methods are to separate the classes of the analyzed problems. Figure 3 presents a scatter plot of the F1 values for all the encoding methods in the studied scenarios with the average value indicated by the cross. The values that represent good quality are the lower ones. From the F1 perspective, it is possible to observe *trace replay*, *node2vec*, *edge2vec* and *fasttext*, as the methods with low overlapping of a single feature in particular cases.

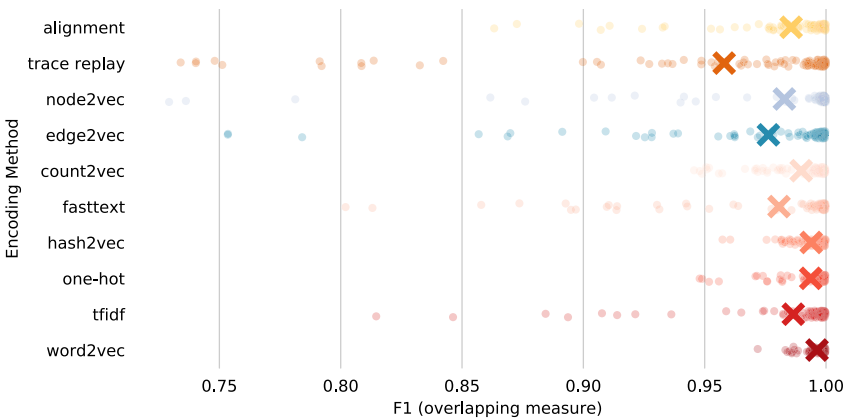

Fig. 3. Maximum Fisher's Discriminant Ratio (F1) values, for the studied scenarios. F1 measures the overlap in classes of the best-disjunct feature.

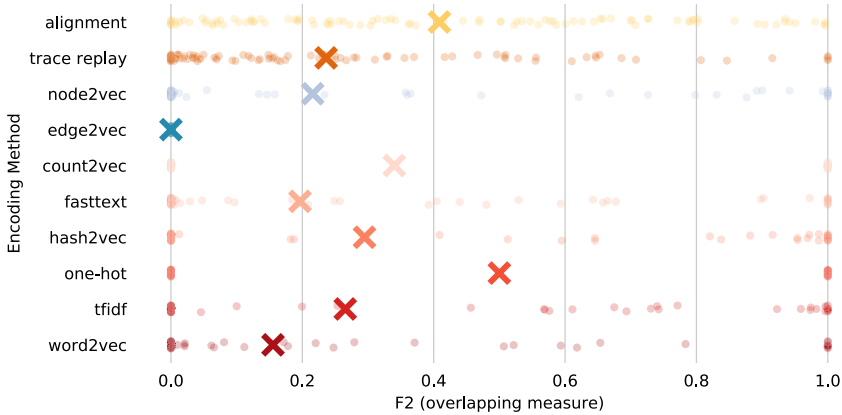

Fig. 4. Volume of Overlapping Region (F2) of the feature values distributions within the problem classes. Low F2 values implies low overlapping.

Encoding quality comparison can be performed by observing the overlapping of the feature values distribution produced by each encoding method. This evaluation is supported by F2, where higher values refer to higher overlap between the classes. As Fig. 4 shows, *edge2vec* achieved the lowest metric value, i.e., the most disjoint representation. On the other hand, *alignment* and *one-hot* encodings generated the most overlapping distributions.

Scenarios with high F1 and F2 values can lead to difficulties in choosing a proper classification algorithm and even demanding hyperparameter tuning to achieve accurate results in classification tasks. Conversely, low F1 and F2 values imply a broader set of algorithms and hyperparametrizations that can discriminate the classes of the problem.

Some encoding methods depend on hyperparameters to determine their feature vector dimension. Using default values, we compared the relevance of dimensions settled by each encoding method to describe most of the data variability through T4. T4 takes advantage of the PCA projection of principal components to identify the number of features capable of representing more than 95% of data variability. The higher the T4 value, the more the encoded features are needed to describe data variability, representing a concise problem description. Lower values represent a waste of features to explain data variability (Fig. 5).

Table 3 shows the T4 values of all encoding methods obtained in the different scenarios. Graph embeddings obtained the lowest T4 values. *Node2vec* obtained 0.02, delivering feature vectors capable of describing the data variability with few samples. On the other hand, *trace replay* and *alignment* presented vectors of dimension closer to the original samples. Several word embedding methods build feature vectors with adaptive sizes to better represent the problem. Among them, *tfidf* was able to obtain competitive results, reaching an average T4 of 0.21. In some scenarios, *tfidf* was superior to both *trace replay* and *alignment*.

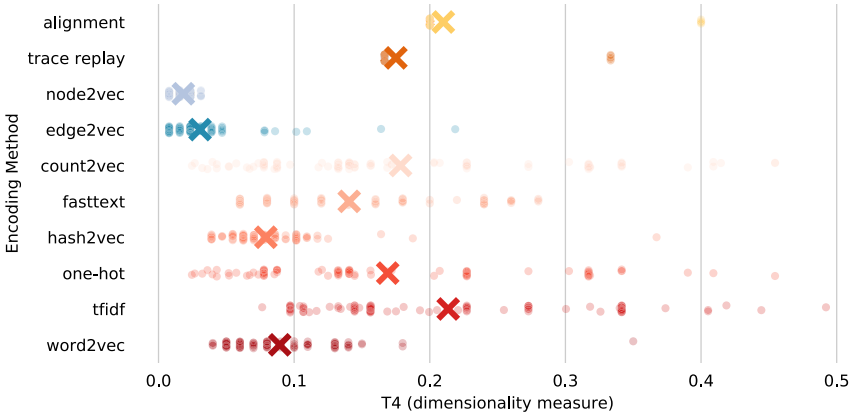

Fig. 5. Ratio of the PCA dimension to the original dimension (T4) of all encoding methods. High T4 means more original features are relevant.

Table 3. Encoding methods dimensionality and T4 values (mean and standard deviation)

Encoding method	T4	Dimensions
trace replay	0.18 (±0.04)	5
alignment	0.21 (±0.04)	5
word2vec	0.09 (±0.04)	100
fasttext	0.14 (±0.07)	50
count2vec	0.18 (±0.10)	22–103*
one-hot	0.17 (±0.10)	22–103*
tfidf	0.21 (±0.10)	22–103*
hash2vec	0.08 (±0.04)	128
node2vec	0.02 (±0.01)	128
edge2vec	0.03 (±0.03)	128

* encoding vector size is determined by the vocabulary size

Complex scenarios, such as Scenarios 4 and 5, required higher dimensionality in the feature space. When dealing with simple scenarios, e.g., Scenario 1, a small number of features is required for all encoding methods. Thus, the demand for dimensions, i.e., larger feature vectors, is strictly related to complex problems.

5.4 Encoding Ranking

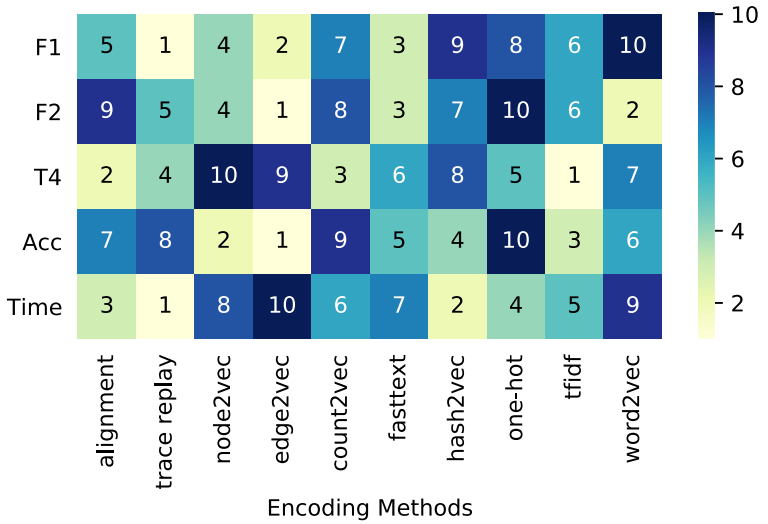

Fig. 6. Ranking of each metric across all encoding methods. The rank ranges from 1 to 10, where the best-ranked position is 1 and the worst-ranked is 10.

Figure 6 presents a heatmap created by ranking each encoding method across mean values of accuracy (Acc), time, F1, F2, and T4. For a concisely and resource-friendly encoding method considering just the classification task, we can take advantage of the *trace replay* and *alignment* methods. Regarding F1 and F2 metrics, the graph embeddings present high performance, mostly with *edge2vec* being the best and second-best in F1 and F2, respectively. This performance demonstrates a great encoding capability proposed in this method. Moreover, *fasttext* regularly ranks well in these two metrics, showing high informative and quality encodings. *Word2vec* has the second-best F2 (low overlap between features) while, at the same time, has the worst F1. This means that the encoding does not produce a unique, highly descriptive feature, and it depends on the conjunction of its created features to encode behavior. *Trace replay* being the best F1 demonstrates its capability of proposing quality encodings.

We need to emphasize that word and graph embedding families can reduce time considerably by performing a feature selection procedure. Also, *count2vec*, *tfidf* and *one-hot* can dynamically adapt the feature vector and *word2vec*, *fasttext*, *hash2vec*, *node2vec* and *edge2vec* have their feature vector size according to user definition. This means parameters for controlling the trade-off between computational time and accuracy are made available by most techniques. Additionally, the complexity of the encoding process needs to be considered. Encoding generation is more time and resource-consuming than the classification task. This way, methods such as *alignment*, which are known to be slow [13], have their applicability hindered in most real situations. At the same time, medium

performance methods, such as *fasttext* and *hash2vec*, demand less computational resources. This way, a trade-off between all the presented perspectives must be considered when choosing an encoding for business processes.

6 Conclusion

In this work, we compared ten trace encoding methods across 100 event logs depicting several scenarios with different levels of complexity. We assessed encodings in classification tasks towards collecting feature vector metrics such as overlapping (F1 and F2) and dimension (T4). Moreover, we considered accuracy and time outcomes to support a general comparison. Overall, the experiments show that encoding significantly contributes to the results of a classification algorithm. Also, a good encoding method can improve a wide range of algorithms without need of tuning. In fact, our experiments suggest that an improper trace encoding can bring additional complexity, obtaining a suboptimal classification performance. In future work, we expect to expand the encoding families, e.g., deep learning encoding approaches. Finally, it is important to study anomalies with more attention and spot their effect on the metrics and performance of different PM tasks.

References

1. Bezerra, F., Wainer, J.: Algorithms for anomaly detection of traces in logs of process aware information systems. Inf. Syst. **38**(1), 33–44 (2013)
2. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Trans. Assoc. Comput. Linguist. **5**, 135–146 (2017)
3. Bose, R.J.C., Van der Aalst, W.M.: Context aware trace clustering: towards improving process mining results. In: Proceedings of the 2009 SIAM International Conference on Data Mining, pp. 401–412 (2009)
4. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324
5. Burattin, A.: PLG2: multiperspective processes randomization and simulation for online and offline settings (2015)
6. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-99414-7
7. Ceravolo, P., Tavares, G.M., Junior, S.B., Damiani, E.: Evaluation goals for online process mining: a concept drift perspective. IEEE Trans. Serv. Comput. 1 (2020). https://ieeexplore.ieee.org/abstract/document/9124702
8. Ceravolo, P., Damiani, E., Torabi, M., Barbon, S.: Toward a new generation of log pre-processing methods for process mining. In: Carmona, J., Engels, G., Kumar, A. (eds.) BPM 2017. LNBIP, vol. 297, pp. 55–70. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65015-9_4
9. Chinosi, M., Trombetta, A.: BPMN: an introduction to the standard. Comput. Stand. Interfaces **34**(1), 124–134 (2012)
10. Cummins, L., Bridge, D.: On dataset complexity for case base maintenance. In: Ram, A., Wiratunga, N. (eds.) ICCBR 2011. LNCS (LNAI), vol. 6880, pp. 47–61. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23291-6_6

11. De Koninck, P., vanden Broucke, S., De Weerdt, J.: act2vec, trace2vec, log2vec, and model2vec: representation learning for business processes. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 305–321. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_18
12. Delias, P., Doumpos, M., Grigoroudis, E., Matsatsinis, N.: A non-compensatory approach for trace clustering. Int. Trans. Oper. Res. **26**(5), 1828–1846 (2019)
13. Fani Sani, M., van Zelst, S.J., van der Aalst, W.M.P.: Conformance checking approximation using subset selection and edit distance. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) CAiSE 2020. LNCS, vol. 12127, pp. 234–251. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49435-3_15
14. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
15. Goyal, P., Ferrara, E.: Graph embedding techniques, applications, and performance: a survey. Knowl.-Based Syst. **151**, 78–94 (2018)
16. Hake, P., Zapp, M., Fettke, P., Loos, P.: Supporting business process modeling using RNNs for label classification. In: Frasincar, F., Ittoo, A., Nguyen, L.M., Métais, E. (eds.) NLDB 2017. LNCS, vol. 10260, pp. 283–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59569-6_35
17. Ho, T.K., Basu, M.: Complexity measures of supervised classification problems. IEEE Trans. Pattern Anal. Mach. Intell. **24**, 289–300 (2002)
18. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery with guarantees. In: Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q. (eds.) CAISE 2015. LNBIP, vol. 214, pp. 85–101. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19237-6_6
19. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_21
20. Lorena, A.C., Garcia, L.P.F., Lehmann, J., Souto, M.C.P., Ho, T.K.: How complex is your classification problem? A survey on measuring classification complexity. ACM Comput. Surv. **52**(5), 1–34 (2019)
21. Mikolov, T., Chen, K., Corrado, G.S., Dean, J.: Efficient estimation of word representations in vector space. CoRR abs/1301.3781 (2013)
22. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: Analyzing business process anomalies using autoencoders. Mach. Learn. **107**(11), 1875–1893 (2018). https://doi.org/10.1007/s10994-018-5702-8
23. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: BINet: multi-perspective business process anomaly classification. Inf. Syst. 101458 (2019). https://www.sciencedirect.com/journal/information-systems/special-issue/10419P9FG88
24. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.D.: Time and activity sequence prediction of business process instances. Computing **100**(9), 1005–1031 (2018). https://doi.org/10.1007/s00607-018-0593-x
25. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
26. Russell, N., ter Hofstede, A., van der Aalst, W., Mulyar, N.: Workflow control-flow patterns: a revised view. BPM reports (2006)
27. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
28. Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, pp. 1113–1120. Association for Computing Machinery (2009)