Anomaly Detection on Event Logs with a Scarcity of Labels

Sylvio Barbon Junior*, Paolo Ceravolo†, Ernesto Damiani‡, Nicolas Jashchenko Omori* and Gabriel Marques Tavares†
*Londrina State University (UEL), Londrina, Brazil
Email: {barbon, omori}@uel.br
†Università degli Studi di Milano (UNIMI), Milan, Italy
Email: {paolo.ceravolo, gabriel.tavares}@unimi.it
‡Khalifa University (KUST), Abu Dhabi, UAE
Email: ernesto.damiani@kustar.ac.ae

***Abstract*—Assuring anomaly-free business process executions is a key challenge for many organizations. Traditional techniques address this challenge using prior knowledge about anomalous cases that is seldom available in real-life. In this work, we propose the usage of *word2vec* encoding and *One-Class Classification* algorithms to detect anomalies by relying on normal behavior only. We investigated 6 different types of anomalies over 38 real and synthetics event logs, comparing the predictive performance of Support Vector Machine, One-Class Support Vector Machine, and Local Outlier Factor. Results show that our technique is viable for real-life scenarios, overcoming traditional machine learning for a wide variety of settings where only the normal behavior can be labeled.**

***Keywords*-One Class Classification; anomaly detection; Local Outlier Factor; encoding; Support Vector Machine**

I. Introduction

In Business Process Management (BPM), anomaly detection has traditionally focused on deviations from a reference model [1]. The reference to a business process model is significant as organizations are driven by normative conformity and adherence to plans. In addition, a process model is an abstraction that can specify sequential, conditional, iterative, and concurrent behavior. This way, cases that significantly differ in the sequence or number of activities can still comply with the same model [2]. Thus, model-aware analytics, such as Process Mining (PM) [3], generalize better, in the BPM domain, than traditional statistics or data mining techniques.

The input of PM is an event log, composed of recorded events described by the *activity* executed at a certain time, the actors who executed the activity, the associated resources, and costs. A unique sequence of time-ordered events of a same case is called *trace*. The analysis of the conformance between a trace and a process model, known in PM as *conformance checking*, highlights control-flow and data-flow anomalies. Control-flow anomaly refers to activities executed in the wrong order. Data-flow anomaly refers to unexpected values in events' attributes, for instance, a user who executes an activity he does not have access to. Patterns in the event log may indicate the root cause of an anomaly, such as a system malfunctioning or a security violation [4].

However, a known problem of PM is that in real scenarios the process model may not be available, be inadequate or incorrect, or significant domain knowledge may be required to elicit it [5]. More recently, Machine Learning (ML) has been proposed as an alternative to process-aware methods, as it can learn anomalies directly from the event log, without needing a reference model [6]. The training stage of ML requires, however, to hold enough ground truth labels for the output classes to be learned. Hence, the need for significant domain knowledge comes back as a need for pre-labeled data.

To get rid of this issue, in this paper, we propose to combine vector space models and One-Class Classification (OCC) algorithms to run anomaly detection by relying solely on the normal behavior described in an event log. Inspired by natural language processing (NLP) we treat activities as words and traces as sentences and use the word2vec algorithm to map them into a vector space [7]. When traces are encoded in the vector space, we use OCC methods to detect anomalies.

To evaluate our approach, we compared OCC to supervised ML. In our experiments, we used both synthetic and real-world event logs, including six different types of anomalies. Results show that our semi-supervised approach handles anomaly detection with performances similar to supervised ML, or even better for a wide variety of anomalies.

The paper is organized as follows. Section II presents the current state of the art. Section III presents our methodology and the datasets used in the experiments. Section IV reports our results and evaluates them with a detailed analysis of the advantages of combining vector space encoding and OCC methods. Section V evaluates the overall implications of the results and places the method in the literature. Finally, we draw our conclusions in Section VI.

II. Related Works

Much BPM research relies on control-flow or data-flow verification for anomaly detection, using place/transition nets or constraint satisfaction [8], [9], [10]. Although various terms

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 Universita degli Studi di Milano.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00032

are used in the literature to frame these issues, PM has been largely adopted in the last years [1].

Bezerra et al. [11] define anomaly in PM using a set of assumptions: (i) the set of traces can be partitioned into a set of normal and a set of anomalous executions; (ii) each of the anomalous execution is infrequent compared to the set of all executions; (iii) the process model representing normal behavior should "make more sense" than the process model representing anomalous traces. More recently, an analysis of online anomaly detection in PM has also been proposed [12]. One of the earliest works for anomaly detection in PM uses a conformance checking pipeline [13], [11]. First, the method filters the dataset based on domain-dependent knowledge. From that, process discovery techniques are applied to the filtered log. Then, the most appropriate model (high fitness while structurally simple) is chosen as the process model. Finally, traces are classified depending on model fitting; that is, a non-fitting trace is classified as anomalous, whereas a fitting trace is classified as a normal execution. However, this approach depends on a clean event log for model creation and assumes that process discovery techniques might generate an ideal model, which is not necessarily true. Moreover, domain knowledge costs resources and is not always available, hence making this method impractical in most situations.

Likelihood graphs encoding the control-flow of a process can be exploited to identify traces with activities having direct follow relationships deviating from an observed probability [14]. However, parallel and recursive behavior cannot be correctly handled using these techniques. For this reason, many PM researchers adopt model-aware analytics where traces are replayed over the model. Each executed activity corresponds to consuming a token. Consumed, missed, and remaining tokens are counted to calculate conformance statistics [3]. The accuracy of this approach substantially depends on the quality of the process model but model discovery techniques are the result of a trade-off between precision and generality. Relying on automatically generated models for anomaly detection is then incongruous while cleansing these models by manual effort is costly.

The adoption of methods using ML for detecting anomalies in business processes has also risen in recent years. In [6], the authors use an autoencoder (a class of neural networks) to model process behavior and further detect anomalies. More recently, a recurrent neural network architecture was proposed [15], [16]. The method (BINet) handles both control-flow and the data perspective of a business process, detecting anomalies in both situations. In its core, BINet is a Gated Recurrent Unit trained to predict the next event. In the preprocessing stage, the logs are transformed into a numerical representation using integer encoding. Anomalies are detected by assuming that an anomalous attribute has a lower probability than a normal attribute, and regulate it by a heuristic threshold. The experiments show that BINet usually outperforms other methods. However, there is a very high computational cost given the deep learning architecture. Moreover, the authors note that BINet suffers from a forgetting problem when sequences of activities are repeated in a case.

A great part of ML techniques relies on supervised strategies, depending on labels and specialist intervention as a preliminary step. In real life, the true class label of a case is not immediately available (or will never be available), and manually labeling might be labor-intensive. Thus, anomaly detection on event logs has to cope with label scarcity. A potential alternative to supervised ML is working with unsupervised learning, particularly clustering methods. Cluster labels have been used as automatically computed labels. Following this idea, in [17], authors developed a framework aimed at bridging the gap between the abstraction levels of row data logs and process models. However, the results of clustering solutions have a high level of variability between different randomized executions and parameters [18].

One-Class Classification algorithms emerged as a suitable solution to traditional supervised learning, as well as being able to address the main drawbacks of the clustering algorithm. OCC is computationally more competitive because only the common behavior samples are required in the training stage [19]. In the PM scenario, these can be achieved from the main business process model and its variants automatically generated. OCC can be performed by either a density or a boundary strategy. When defining the classification boundary around the positive class towards accepting as many samples as possible, it minimizes the chance of accepting non-positive (i.e. outlier) samples [19], [20]. Thus, OCC is a suitable and straightforward solution to address scenarios with a scarcity of labels.

Our approach overcomes the mentioned gaps with: (i) usage of a better encoding technique; (ii) ability to work with a scarcity of labels; (iii) no need of a process model; (iv) no required expert knowledge about the business process.

III. Methodology

This section presents the event logs, the encoding strategy, the algorithms, and the evaluation metrics we used for comparing OCC to supervised ML.

A. Event logs

A controlled scenario with synthetic event logs and known labels is ideal to evaluate the performance of our method.

We generated event logs following the procedure proposed by Nolle et al. [16] that allows creating events logs from process models using a seed state, where the same seed guarantees the creation of identical event logs. A likelihood graph [14] models the dependencies within events and between events and attributes. This implies that the probability distributions in the control-flow are constrained. For example, activity A has a 60% probability of being followed by activity B. The constraints also bind events and attributes, e.g., an activity A has an 80% probability of being followed by activity B when A is executed on Mondays. This modeling allows introducing long-term control-flow dependencies, typical of real-world scenarios. Event logs can be generated by merely performing random walks in the likelihood graph, respecting

its transition probabilities. We decided to replicate previously proposed datasets to create a common ground for comparison between different methods. The steps followed to replicate this synthetic dataset are accessible in the code repository linked to [16]. The models we employed are a subset of those used in [16] but cover the entire range of those publicly available. With the use of the PLG2 tool [21], six random process models were created. They vary in complexity, i.e., number of activities, breadth, and width. A handmade procurement process model (P2P) from [15] was also added. In addition to these synthetic event logs, we used real-life event logs from previous Business Process Intelligence Challenges (BPIC)[1]: BPIC12, BPIC13, BPIC15, and BPIC17.

After the event logs were produced, the next step was to inject anomalies into them. Following [11], [14], [16], we applied six different anomaly types. (i) *Early*: a sequence of 2 or fewer events executed too early, which is then skipped later in the case. (ii) *Late*: a sequence of 2 or fewer events executed too late. (iii) *Insert*: 3 or less random activities inserted in the case. (iv) *Skip*: a sequence of 3 or less necessary events is skipped. (v) *Rework*: a sequence of 3 or less necessary events is executed twice. (vi) *Attribute*: an incorrect attribute value is set in 3 or fewer events.

These anomalies were injected in 30% of all cases from each event log, including those from real-life scenarios (BPICs). Note that the anomalies are on the event level, but they can be converted to the case level easily: a case is anomalous if any of its event attributes is anomalous. For each synthetic process model, four likelihood graphs were created with different transition probabilities, dependencies, and the number of attributes. In total, we created 28 synthetic event logs that, together with the BPICs, form a total of 38 event logs used in our experimental evaluation. Table I shows the detailed statistics of these event logs.

Name	#Logs	#Activities	#Cases	#Events	#Attrs.	#Attr. values
P2P	4	27	5k	48k-53k	1-4	13-386
Small	4	41	5k	53k-57k	1-4	13-360
Medium	4	65	5k	39k-42k	1-4	13-398
Large	4	85	5k	61k-68k	1-4	13-398
Huge	4	109	5k	47k-53k	1-4	13-420
Gigantic	4	154-157	5k	38k-42k	1-4	13-409
Wide	4	68-69	5k	39k-42k	1-4	13-382
BPIC12	1	73	13k	290k	0	0
BPIC13	3	11-27	0.8k-7.5k	4k-81k	2-4	23-1.8k
BPIC15	5	422-486	0.8k-1.4k	46k-62k	2-3	23-481
BPIC17	1	53	31k	1.2M	1	299

TABLE I: Detailed event logs statistics

B. Encoding strategy

Traditionally, ML and data mining techniques cannot be directly applied to PM event logs due to a mismatch at the representation level [22]. ML operates at the event level, where each event is an instance, PM at the business case level, where a group of events represents an instance of the process. This way, embedding techniques are necessary to overcome this gap [23].

There are a few attempts to perform event log encoding in the literature. In [6], the authors use one-hot encoding to transform the log into a numerical representation before feeding an autoencoder. Similarly, in [16], [15], the authors use integer encoding to map all log attributes into integers and then use the resulting vectors as the input for a deep learning algorithm. Though both encoding approaches are traditional in ML literature, there are better-performing ones proposed recently. In [24], the authors represent a trace using the *doc2vec* approach [25], which extends the Continuous Bag of Words architecture with a paragraph vector for encoding traces. Inspired by this perspective, we employed the widely known word2vec algorithm [7]. Word2vec encodes words by training a neural network to reconstruct its linguistic context in a corpus. Word vectors embed the weights learned by the neural network capturing, this way, semantic and syntactic similarities in vectors. Our method interprets each activity as a word. Hence, the corpus is composed of the set of unique activities in the business process. The vectors we obtain can then be aggregated to get trace-level representations. In our case, we use the element-wise mean, i.e., the trace representation is the mean of its activities weights.

C. Classification algorithms

We used three different algorithms for classifying the traces into normal and anomalous, Support Vector Machines (SVM) [26], One-Class Support Vector Machines (OCSVM) proposed by Tax et al. [27] and Local Outlier Factor (LOF) proposed by Breunig et al. [28]. All three algorithms are available on the Scikit-learn Python package [29]. Several hyperparameter values were combined as a tuning strategy, as illustrated in Table II. The first technique is supervised, meaning that it depends on labels for its training. The other two algorithms require only the common behavior to induce a model able to classify unlabeled data. The SVM is used as a baseline for comparison purposes, aiming to check if the techniques that do not depend on completely labeled data can produce comparable results.

TABLE II: Collection of combined hyperparameter values

Method	Hyperparameter	Values
SVM	*c*	[0.1, 1, 10, 100, 1000, 10000, 100000]
	kernel	[polynomial, rbf, sigmoid]
	gamma	[auto, scale]
OCSVM	*nu*	[0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4]
	kernel	[polynomial, rbf, sigmoid]
	gamma	[auto, scale]
LOF	*k*	[1, 10, 25, 50, 100, 250]
	contamination	[0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, auto]

SVM considers each n-dimensional input vector as a point in an n-dimensional space. Then, the algorithm searches for an n-1 dimensional hyperplane that can correctly separate points. To find this hyperplane, the algorithm tries to maximize the

[1]https://www.tf-pm.org/resources/logs

distance of the hyperplane to the closest boundary points, called Support Vectors. In this work we used the sigmoid, polynomial and RBF kernels for the tests, varying the c and $gamma$ hyperparameters.

OCSVM is a one-class version of SVM. Similarly, it considers the input vectors as points in an n-dimensional space but, instead of creating a hyperplane to separate the data into two classes, it creates an n-dimensional hypersphere that can represent the input data such as the points that are inside the hypersphere are considered the normal behavior and the points that are outside are considered anomalies. The algorithm aims at maximizing the number of points inside the hypersphere while minimizing the empty space inside it. Different values for the parameter nu are used to assume different levels of generalization by the algorithm, with higher values standing for higher generalization. The sigmoid, polynomial and RBF kernels were considered in our experiments.

LOF detects outliers given a set of vectors, which are also treated as points in an n-dimensional space. The main idea behind LOF is to compare the local point density to its k nearest neighbors densities. Thus, points with lower density than its neighbors are considered outliers. The experiment used different numbers of k nearest neighbors as well as different values for *contamination*, a hyperparameter that defines the density threshold for anomaly detection. For the evaluation of the algorithms, we chose the F-score metric [30].

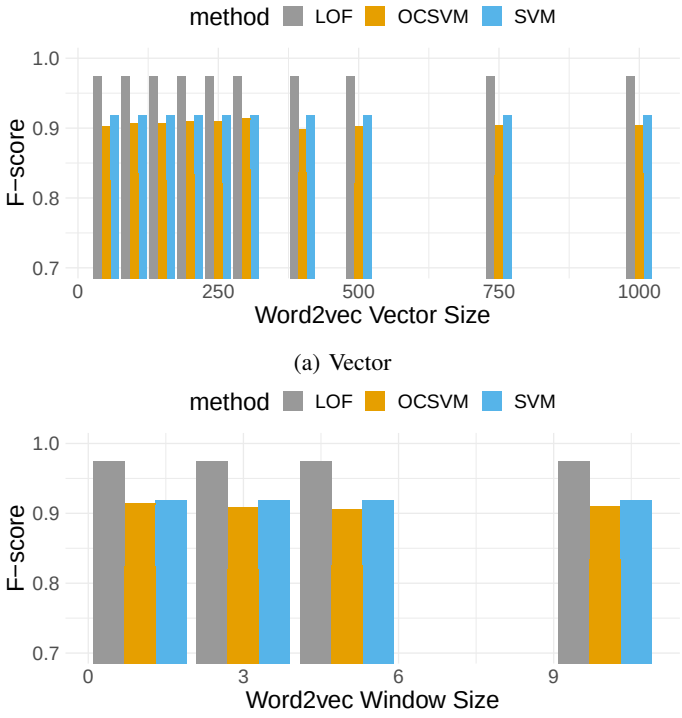
IV. Results

This section presents the results obtained from several complementary perspectives, evaluating their impact on the overall performance. To follow the open science principles, we made the experiments and event logs available[2].

A. Word2vec descriptive performance

The initial experiment aims at evaluating how the proposed encoding influences the results. Regarding word2vec hyperparameters, we explored the number of dimensions and window size, which are reportedly the most impacting hyperparameters. The implementation used the *gensim* package for Python[3] and the other hyperparameters were set to standard values. Figure 1a and 1b show the F-score values obtained.

According to Figure 1a, the impact of different vector sizes on the performance is minimal. For this task, we ranged values from 50 to 1000. In other domains, the number of dimensions highly affects the representational capacity of word2vec. A low number usually diminish encoding quality. However, traditional NLP tasks perform over text corpora containing large quantities of unique words. In business processes, the set of unique activities (which we consider as words) is of several orders of magnitude less. Moreover, the contexts that surround business process activities are often less heterogeneous. The same phenomenon is revealed in Figure 1b, where various window sizes are compared. The window size controls how much of the context surrounding each word is considered in the learning procedure. We expected longer windows to drive richer characterization of activities, but our experimental results show that it does not impact performances.



(a) Vector

(b) Window

Fig. 1: F-score obtained with different window and vector sizes of word2vec across all event logs

The main lesson learned from these experiments is the robustness of word2vec in representing the context of business activities. It can be adopted without the need to search for optimal hyperparametrization. Hence, in BPM applications, smaller vector and window sizes are advised as they consume less computational resources without losing representational capacity.

Figure 2 further demonstrates the effectiveness of word2vec embeddings. We applied this technique to one of the `small` event log using 200 dimensions and a window size of 1. Then, using the t-SNE dimensionality reduction technique (with standard hyperparameters from Scikit-learn package[4]), the number of dimensions was reduced to two. We can see how word2vec distributes the normal and anomalous classes in the feature space. Word2vec correctly interprets the activities contexts placing anomalous behavior apart from normal. Moreover, anomalous instances are usually near normal ones because they are a slight variation of them, according to the injecting procedure we followed.

[2]https://github.com/gbrltv/ProcessAnomalyDetector
[3]https://radimrehurek.com/gensim/models/word2vec.html
[4]https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

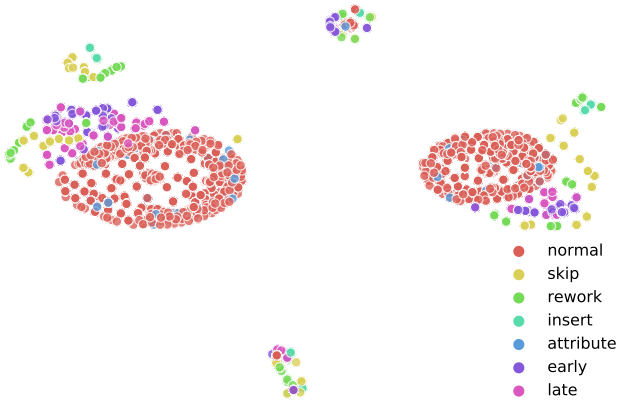
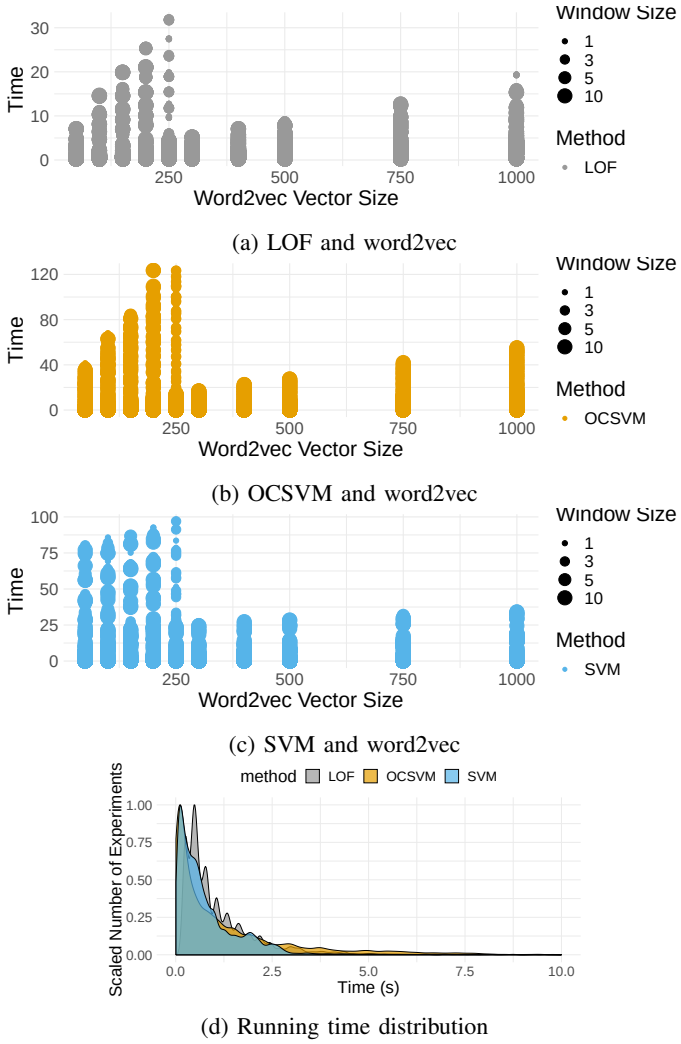


Fig. 2: `small` trace distribution. The experiment used word2vec to model the business process behavior. Then, the t-SNE dimensionality reduction technique was applied for visualization. It is notable how anomalous and normal behavior is quite separated in the feature space



(a) LOF and word2vec

(b) OCSVM and word2vec

(c) SVM and word2vec

(d) Running time distribution

Fig. 3: Comparing execution time, in seconds, with window and vector size per algorithm across all event logs. Quadrant (d) reports the likelihood of observing a specific execution time in the experiments we ran

B. Time analysis

We compared a total of 40 different word2vec configurations with several setups of LOF, OCSVM, and SVM. It is possible to observe that small vector sizes, i.e., less than 200 features, affect the execution time with small window sizes. Configurations of this kind demand more time due to more frequent sliding over the samples, as seen in figures 3a, 3b and 3c. When dealing with more than 250 features, time consumption is independent of window sizes, drastically reducing the number of outliers in the results.

SVM shows a slightly superior time performance, followed by LOF and OCSVM. However, the average time difference is less than one second, as observed in the running time distribution of Figure 3d. The size of the vector has, indeed, the most noticeable impact on execution time. We also compared the time of classification algorithms according to the Friedman and Nemenyi test [31]. Using $\tau = 0.05$ and critical distance of 0.53, it was possible to account a significantly different performance of SVM (1.37, ranked first), the fastest algorithm. LOF (2.13, ranked second) and OCSVM (2.50, ranked third) were not accounted as significantly different.

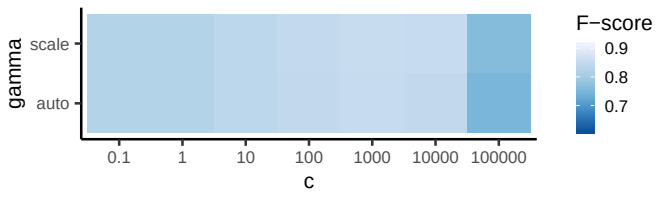
Considering the results of Section IV-A, we then suggest the adoption of small vector sizes, particularly 50 features, which provide fast processing and stability for most event logs with competitive predictive performance.

C. Hyperparameter selection

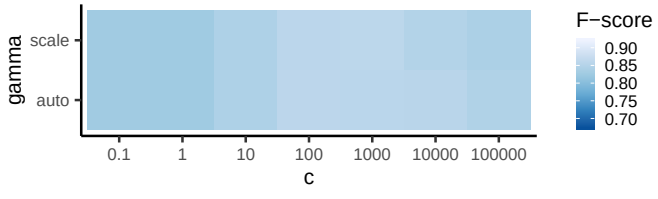
This experiment focuses on the tuning of hyperparameters to improve performances and support fair comparison among the methods. Regarding SVM, the best hyperparameter to deal with all event logs was c =1000 and $gamma$ as $scale$ using a $polynomial$ kernel. This algorithm was the most volatile to hyperparameter selection, e.g., some configurations of c using $sigmoid$ kernel reduced drastically the F-score, as shown in Figures 4a, 4b and 4c. For OCSVM, the nu hyperparameter has impacted the most on performance. Small nu values (best $nu = 0.01$) results in better predictive outcomes independent of $gamma$ or $kernel$. As illustrated in Figures 4d, 4e and 4f. Finally LOF, similarly to SVM, demands a combination of hyperparameters to achieve the optimal performance. Also, the $auto$ value for the $contamination$ parameter obtained an average good performance. For k (number of neighbors), smaller values (1, 10, 25 and 50) were the best. Using high $contamination$ (value of 0.4) we obtained the worst LOF performances, as visible in Figure 4g.

D. Classification performance

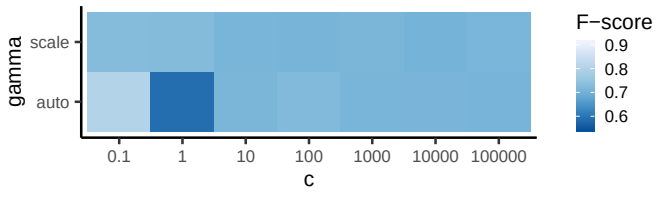
Figure 5 shows a boxplot of the best F-scores for each algorithm. LOF's entire first quartile is above 0.95 while its median is 0.96. The median F-scores for OCSVM and SVM were 0.87 and 0.9, respectively. Therefore, LOF outperforms



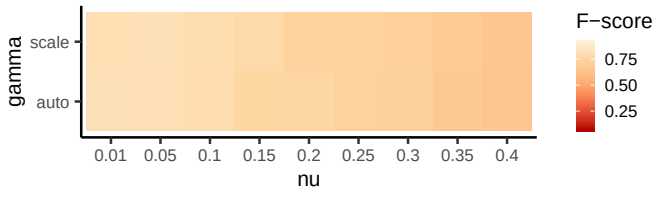
(a) SVM with poly kernel



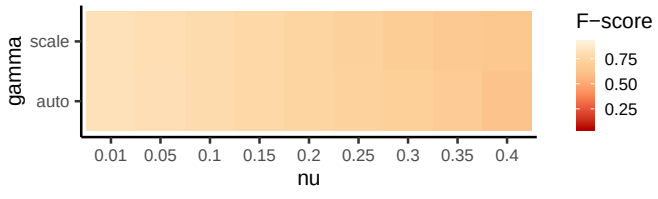
(b) SVM with rbf kernel



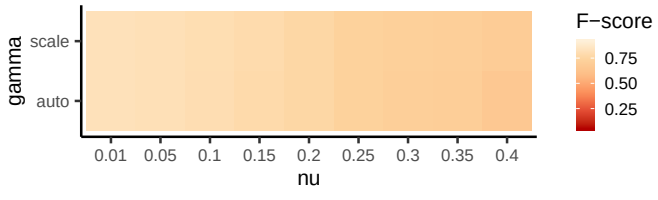
(c) SVM with sigmoid kernel



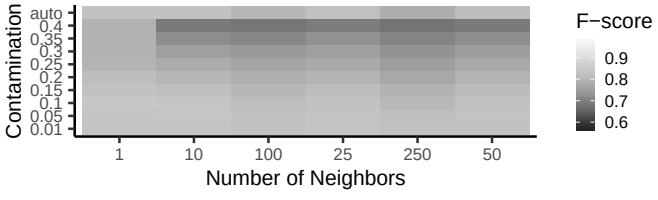
(d) OCSVM with poly kernel



(e) OCSVM with sigmoid kernel



(f) OCSVM with rbf kernel



(g) LOF

Fig. 4: Hyperparameter overview of SVM, OCSVM, and LOF, light regions means best predictive performance (F-score) across all event logs

both SVM and OCSVM. This result is valuable as LOF does not even need anomalous examples to induce its model, making it easier to prepare an event log for this method. That is, supervised methods do not necessarily imply better performance.

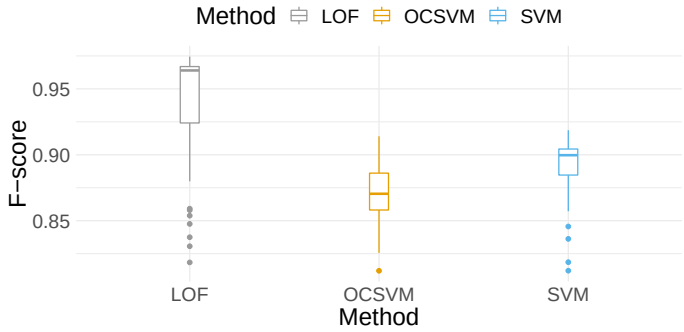


Fig. 5: Boxplot F-score results for each classification algorithm across all event logs

Figure 6 compares the performances over individual event logs. As corroborated by the previous analysis, LOF outperforms the other two algorithms in almost all event logs. This trend is even more explicit in synthetic event logs, where LOF reaches F-scores of over 0.95. In most datasets, SVM is better than OCSVM by a low margin, however, it required a longer tuning process for its hyperparameters. Real-life event logs have a lower F-score when compared to synthetics. The main reason is that BPIC event logs naturally contain not labeled anomalies, which incorrectly represent normal behavior.

F-score comparison among the classification algorithms, according to the Friedman and Nemenyi test using $\tau = 0.05$, showed significantly different performances. The critical distance of 0.53 attested the superiority of LOF (1.11, ranked first), followed by SVM (2.00, ranked second) and OCSVM (2.89, ranked third).

E. Performances by anomaly

As there are six anomaly types, we further experimented to analyze their differences. For this, we created six additional event logs using the `medium` behavior. For each anomaly type, we created an event log with 30% of anomalies using only the respective type. Figure 7 reports the results of the algorithms for each anomaly. The first important note is that having only one type of anomaly in the log makes the anomaly identification easier.

The *attribute* anomalies are the most difficult to detect as they do not insist on the control-flow perspective. In spite of it, the results obtained are not particularly severe. Regarding the other anomaly types, LOF can detect anomalous instances as they all affect the control-flow aspect, which is well inferred by word2vec. SVM presented good results for *insert*, *rework* and *skip* anomalies. These anomalies are easier to detect due to a higher impact on activities' contexts. For instance, *skip* makes a trace to miss a required activity execution. Then, when analyzing a trace with a skipped activity, word2vec modeling

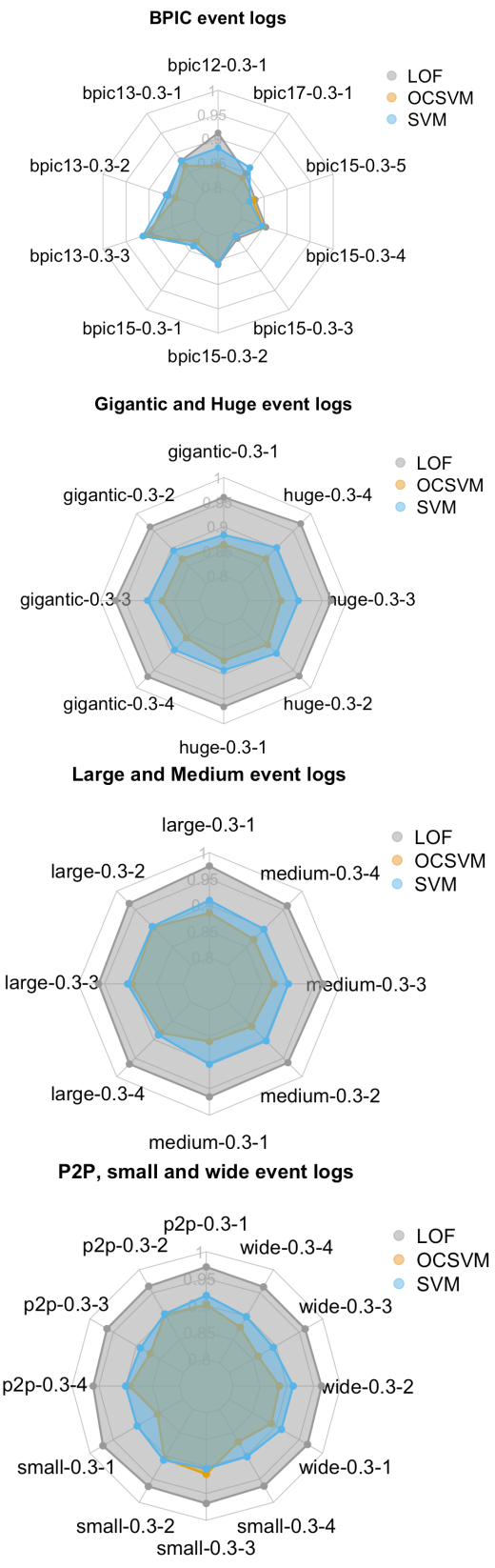


Fig. 6: F-score of all event logs grouped by size and behavior

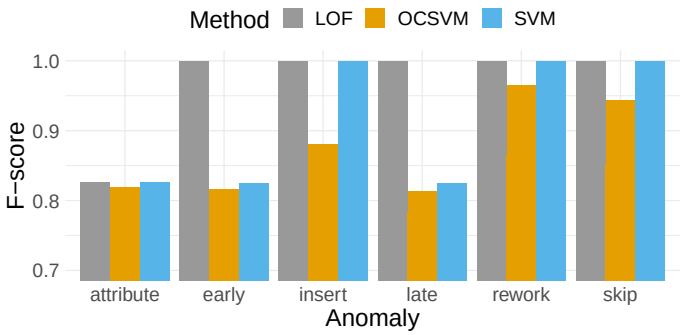


Fig. 7: Comparing the F-score of each algorithm using the event log `medium`

is sensible enough to detect that the trace context is different from normal behavior. For *early* and *late* anomalies, the effect on the context is more subtle because the activities are still in the trace even if in wrong positions. Therefore, this shows the importance of having an OCC classification on top of word2vec as it relies on normal samples, counterbalancing this issue. This is seen by the exceptional performance reached by LOF.

V. Discussion

Overall, SVM was slightly faster than OCC methods, however, SVM performance is significantly dependent on hyperparameters, e.g., good performance was achieved at the cost of a long tuning process. When comparing accuracy, LOF was superior, reaching the best predictive performance for most real-life and all the synthetic event logs. The SVM performance was competitive only in a few real-life datasets, though the pre-existence of non-reported anomalous cases can have swayed the learning process from OCC methods. OCSVM was the slowest and less predictive approach. It is important to mention that *early* and *late* anomalies, when compared in a synthetic log of medium size, related to a similar detection score. These anomalies are more difficult to be modeled since the events affected are still present in the trace with changed positions. For this reason, when aggregating the trace representation, the overall trace encoding can be similar to a normal trace. The phenomenon is observable because word2vec is more flexible to ordering than traditional conformance checking algorithms. That is, word2vec context composes the activities neighborhood more broadly, allowing for different sequences to have similar representations in some cases. However, when combining word2vec with LOF, this challenge was overcome, as Figure 7 shows. Another similar set was *insert*, *rework* and *skip*, that obtained higher detection scores. These anomalies are easily captured by word2vec because they insert an abnormal activity, repeat the execution of an activity, and skip an expected execution, respectively. Thus, the difference between normal behavior is more abrupt. The worst scores were observed with the *attribute* anomaly type. The *attribute* anomaly does not affect the control-flow perspective, it only affects the data attributes. This way, it is

more complex for encoders processing traces to capture this anomaly.

Deep learning methods [6], [16], [15] are considered out of the scope of this paper and we did not compare to the algorithms studied in this work. The reason is deep learning algorithms require ground truth labels and large availability of computational resources. They, in other words, match to application requirements completely different from those we assumed.

VI. Conclusion

The presented work proposes the use of word2vec, a traditional NLP technique, to encode business process behavior as the context of activities in an event log. Moreover, the method explores the use of OCC algorithms, showing their advantage in scenarios with a scarcity of labels. The performance obtained by the combination of trace encoding and OCC techniques demonstrates the feasibility of our method. Therefore, organizations can profit by the use of this methodology as detecting anomalies is a significant concern given their impact on process performance and resource consumption. Different word2vec configurations were quite similar in terms of predictive performance, demonstrating robustness to vector and window sizes. When comparing to supervised ML, OCC presented a better performance in most cases. As future work, we plan (i) to explore other aggregation techniques to represent a trace encoding, (ii) consider the data-flow perspective, and (iii) to apply OCC algorithms in online PM.

References

[1] W. M. P. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Heidelberg: Springer, 2016.

[2] M. Boltenhagen, T. Chatain, and J. Carmona, "Generalized alignment-based trace clustering of process behavior," in *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 2019, pp. 237–257.

[3] F. Bezerra, J. Wainer, and W. M. van der Aalst, "Anomaly detection using process mining," in *Enterprise, business-process and information systems modeling*. Springer, 2009, pp. 149–161.

[4] R. P. Jagadeesh Chandra Bose and W. van der Aalst, "Trace alignment in process mining: Opportunities for process diagnostics," in *Business Process Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 227–242.

[5] W. M. Van der Aalst, V. Rubin, H. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software & Systems Modeling*, vol. 9, no. 1, p. 87, 2010.

[6] T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser, "Analyzing business process anomalies using autoencoders," *Machine Learning*, vol. 107, no. 11, pp. 1875–1893, 2018.

[7] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[8] S. Barbon Junior, G. M. Tavares, V. G. T. da Costa, P. Ceravolo, and E. Damiani, "A framework for human-in-the-loop monitoring of concept-drift detection in event log stream," in *Companion Proceedings of the The Web Conference 2018*, ser. WWW '18. International World Wide Web Conferences Steering Committee, 2018, p. 319–326.

[9] A. Rogge-Solti and G. Kasneci, "Temporal anomaly detection in business processes," in *Business Process Management*. Cham: Springer International Publishing, 2014, pp. 234–249.

[10] M. Rovani, F. M. Maggi, M. De Leoni, and W. M. Van Der Aalst, "Declarative process mining in healthcare," *Expert Systems with Applications*, vol. 42, no. 23, pp. 9236–9251, 2015.

[11] F. Bezerra and J. Wainer, "Algorithms for anomaly detection of traces in logs of process aware information systems," *Information Systems*, vol. 38, no. 1, pp. 33 – 44, 2013.

[12] P. Ceravolo, G. Marques Tavares, S. B. Junior, and E. Damiani, "Evaluation goals for online process mining: a concept drift perspective," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[13] F. Bezerra, J. Wainer, and W. M. P. van der Aalst, "Anomaly detection using process mining," in *Enterprise, Business-Process and Information Systems Modeling*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 149–161.

[14] K. Böhmer and S. Rinderle-Ma, "Multi-perspective anomaly detection in business process execution events," in *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*. Cham: Springer International Publishing, 2016, pp. 80–98.

[15] T. Nolle, A. Seeliger, and M. Mühlhäuser, "Binet: Multivariate business process anomaly detection using deep learning," in *Business Process Management*. Cham: Springer International Publishing, 2018, pp. 271–287.

[16] T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser, "Binet: Multi-perspective business process anomaly classification," *Information Systems*, p. 101458, 2019.

[17] G. Tello, G. Gianini, R. Mizouni, and E. Damiani, "Machine learning-based framework for log-lifting in business process mining applications," in *Business Process Management*. Cham: Springer International Publishing, 2019, pp. 232–249.

[18] C. C. Aggarwal, "Proximity-based outlier detection," in *Outlier Analysis*. Springer, 2017, pp. 111–147.

[19] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *Irish conference on artificial intelligence and cognitive science*. Springer, 2009, pp. 188–197.

[20] I. Kang, M. K. Jeong, and D. Kong, "A differentiated one-class classification method with applications to intrusion detection," *Expert Systems with Applications*, vol. 39, no. 4, pp. 3899–3905, 2012.

[21] A. Burattin, "Plg2: Multiperspective processes randomization and simulation for online and offline settings," 2015.

[22] G. M. Tavares, P. Ceravolo, V. G. Turrisi Da Costa, E. Damiani, and S. Barbon Junior, "Overlapping analytic stages in online process mining," in *2019 IEEE International Conference on Services Computing (SCC)*, July 2019, pp. 167–175.

[23] P. Ceravolo, E. Damiani, M. Torabi, and S. Barbon, "Toward a new generation of log pre-processing methods for process mining," in *Business Process Management Forum*. Cham: Springer International Publishing, 2017, pp. 55–70.

[24] P. De Koninck, S. vanden Broucke, and J. De Weerdt, "act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes," in *Business Process Management*. Cham: Springer International Publishing, 2018, pp. 305–321.

[25] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, p. II–1188–II–1196.

[26] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[27] D. M. Tax and R. P. Duin, "Support vector data description," *Machine Learning*, vol. 54, no. 1, pp. 45–66, Jan 2004.

[28] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 93–104.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, 2018.

[31] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.