

Original software publication

VERONA: A python library for benchmarking deep learning in business process monitoring

Pedro Gamallo-Fernandez^{*}, Efrén Rama-Maneiro, Juan C. Vidal, Manuel Lama

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITUS), Universidade de Santiago de Compostela, 15782, Santiago de Compostela, Spain

ARTICLE INFO

Keywords:

Process mining
Predictive process monitoring
Benchmarking
Deep learning

ABSTRACT

Predictive process monitoring is a subfield of process mining that focuses on predicting the future behavior of real-world processes, anticipating constraint violations and bottlenecks, and enabling real-time decision making. Among other machine learning approaches, Deep Learning-based architectures have achieved high levels of prediction accuracy, becoming an increasingly prolific area of research in recent years. However, the variety of datasets, learning techniques, and metrics used makes the comparison of proposals complicated and biased. To address this problem this paper presents VERONA, a Python library designed for the development of the deep learning predictive process monitoring pipeline. Additionally, this library provides a framework for replicating the experimental setup of the state-of-the-art benchmark in the field, enabling streamlined comparison of new approaches and improving the reproducibility of experiments.

Code metadata

Current code version	v1.1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00064
Permanent link to Reproducible Capsule	none
Legal Code License	GNU Public License v3.0 (GPL3)
Code versioning system used	Git
Software code languages, tools, and services used	Python 3.11
Compilation requirements, operating environments & dependencies	PM4Py, Pandas, NumPy, Scikit-Learn, Plotly, Requests, CmdStanPy
If available Link to developer documentation/manual	https://verona-library.readthedocs.io
Support email for questions	pedro.gamallo.fernandez@usc.es ; efren.rama.maneiro@usc.es

1. Motivation and significance

Business processes play an essential role in the day-to-day operations of companies and institutions [1]. From a car factory assembly line to a hospital emergency room, these processes determine the sequence of events to be followed to achieve a business goal, i.e., providing a product or service. The correct analysis and understanding of the process flow is essential to optimize the use of resources, minimize waiting times, reduce costs, increase the quality of services and products, and achieve many other benefits for both organizations and users.

Process mining is the discipline that aims to discover, monitor and improve real business processes [2]. Process mining techniques examine the event logs of process-aware information systems (PAIS) and

extract knowledge about the underlying processes. These event logs, usually presented in tabular form as in Table 1, contain information about events from previously executed instances of the process, such as the recorded activity, the timestamp at which it was recorded, and the associated resource. The events of each process instance, known as a case or trace, are grouped under the same case identifier (*CaseID*).

One of the most interesting tasks in process mining is predicting how a running case will unfold, allowing organizations to anticipate constraint violations or bottlenecks and make early decisions. Predictive process monitoring (PPM) is a subfield of process mining that learns predictive models using the historical information from the event log for this application [3]. As Fig. 1 shows, PPM approaches receive as

^{*} Corresponding author.

E-mail address: pedro.gamallo.fernandez@usc.es (Pedro Gamallo-Fernandez).

Table 1
Extract from a real event log.

CaseID	Activity	Timestamp	Resource
Case3814	Assign seriousness	2012-05-04 15:23:08	Value 1
Case3814	Take in charge ticket	2012-05-07 07:01:12	Value 6
Case3814	Resolve ticket	2012-05-24 15:25:23	Value 12
Case3814	Closed	2012-06-08 15:25:48	Value 5
Case3815	Assign seriousness	2010-09-08 10:36:14	Value 2
Case3815	Wait	2010-09-08 15:32:25	Value 14
Case3815	Take in charge ticket	2010-09-13 09:01:56	Value 2

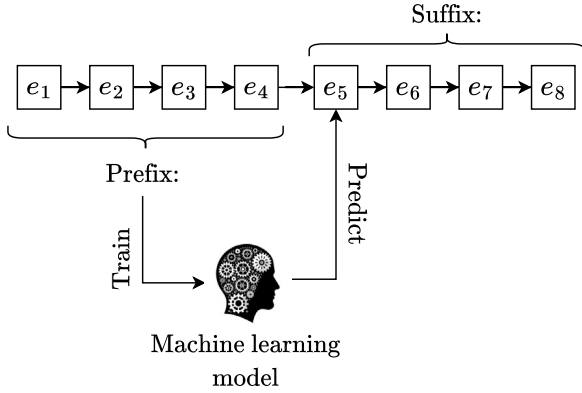


Fig. 1. Predictive monitoring workflow.

input a sequence of events from an ongoing case up to a certain point (*event prefix*). This information serves as historical knowledge that is then used to train the predictive model with the goal of predicting any variable of interest from the future events that occur after that point in time (*event suffix*). Specifically, the main predictions addressed are the prediction of the next activity [4], the suffix of activities until the end of the case [5], the timestamp of the next activity [6] or the remaining execution time of the case [7].

Many machine learning techniques have been applied in predictive monitoring, such as Decision Trees [8], Support Vector Machines [9], Clustering Analysis [10], Factorization Machines [11] and, more recently, Deep Learning-based architectures, with the latter being the most successful and popular. However, implementing these approaches can be a lengthy and tedious process due to the numerous datasets, partitioning methods, input encoding techniques, predictive tasks, and performance metrics available. This can result in valuable time being spent on preprocessing and postprocessing tasks that could be invested in the development of predictive models. Additionally, it can be challenging to compare fairly and impartially with other existing approaches. Therefore, two main problems arise:

1. While there are tools available to support the development of the PPM pipeline, such as *ProM* [12] or *Nirdizati* [13], they focus on the interactive execution of methods through a desktop or web application with a set of predefined models. Currently, there is no library available that implements the PPM preprocessing and postprocessing development stages while also offering flexibility and easy integration with the development pipeline of creating new models, as *pm4py* [14] does in a more generic way for process mining.
2. Although [15] presents a fair comparison of the most relevant deep learning-based predictive monitoring approaches, by standardizing their data partitioning strategy and performance metrics, this evaluation framework is difficult to replicate and requires a significant amount of work to integrate into the usual workflow of a data scientist or researcher who intends to develop a new predictive monitoring approach.

The VERONA library aims to tackle the aforementioned challenges by implementing a model agnostic framework to develop the PPM workflow and reproduce the benchmark [15] in a simple and user-friendly way. This library provides primitives for event log retrieval and pre-processing, data partitioning, evaluation with benchmark metrics, and visualization of results with plots, allowing the user to focus only on developing the predictive model. Furthermore, it also includes a port of the Bayesian statistical comparison functionalities proposed in [16] into the Python language, which is one of the core pillars of the benchmark [15]. By not relying on a specific prediction model, these functionalities can be used by any PPM researcher, regardless of the specific architecture being implemented. Finally, it allows importing the results of the original benchmark [4,6,7,17–23] as well as three additional approaches [24–26], so that any new developed approach can be easily compared with the ones already evaluated. Designed for easy integration, VERONA facilitates the encoding, execution, and evaluation of common predictive monitoring tasks with minimal code.

2. Software description

VERONA is a Python-based software library that simplifies the development of the pre-processing and results evaluation stages of the PPM workflow, enabling a fair comparison against established benchmarks [15]. This section describes its architecture and functionalities.

2.1. Benchmarking framework

The benchmarking framework implemented in VERONA, which also serves as the foundation for the comparisons in [15], is grounded in four main pillars:

- *Event logs*: a wide variety of event logs is available for researchers to compare their predictive models in various contexts. This allows for comparisons to be conducted based on the inherent characteristics of the event logs. In addition to the logs provided from the original benchmarking by [15], we have expanded the available collection to include many other publicly available logs.
- *Preprocessing*: the preprocessing strategy implemented in VERONA takes into account the unique nuances of predictive monitoring and also ensures that the results are reliable between different preprocessing executions.
- *Metrics*: a comprehensive set of metrics is provided so any new predictive model can be evaluated in the most common predictive tasks (next activity, time, and suffix). Also, we provide reliable implementations of such metrics in order to streamline the workflow of new researchers and diminish the probability of introducing software bugs in their software.
- *Statistical tests*: we provide Python implementations from the Bayesian statistical tests used in [15]. In particular, we ported every Bayesian statistical test from *scmamp* [16] (correlated T-Test [27], Hierarchical [27], Plackett–Luce ranking [28] and signed rank [27]) into Python providing an intuitive and easy-to-use interface that can be used by either predictive monitoring and machine learning researchers alike.

Fig. 2 shows the interactions between these components. The starting point is a given event log that is further preprocessed using a partitioning strategy, normally a 5-fold cross-validation. Then, each of these partitions is divided into tuples of prefixes and prediction targets, the latter depending on the predictive task at hand. Then, a predictive model is applied over this set of tuples, evaluated using the metrics established for the predictive task and, finally, compared with other approaches using the most appropriate statistical tests.

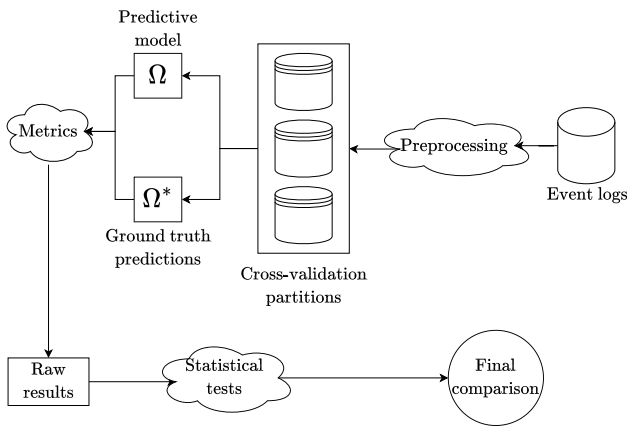


Fig. 2. Benchmarking framework implemented in VERONA.

2.2. Software architecture

The VERONA library organizes its functionalities into three primary packages based on their respective functionalities: **data**, **evaluation**, and **visualization**, which are represented in the package diagram of Fig. 3. The **evaluation** package, further, is divided into two sub-packages: **metrics** and **statstests**.

The **data** package contains the primitives to manage anything related to event logs, including obtain public event logs and experimental results used in [15], partitioning strategies, and generating prefix and target pairs. Furthermore, this package offers utilities to load existing benchmark results from [15], and generate event log statistics for reporting the experimental setup in potential publications.

The **evaluation** package includes functions for the evaluation stage of the model in the benchmarking framework. First, **evaluation.metrics** provides a variety of metrics to evaluate the predictive problems of next activity, suffix, and process times. Second, **evaluation.statstests** reimplements in Python the statistical tests used in [15] so that the developed predictive models can be compared in a fair and robust way.

Finally, the **visualizations** package offers graphic representations for metrics, statistical tests (specifically the ranking test), and event log statistics that adapt dynamically to the number of approaches and event logs utilized and enable users to customize the content of the visualizations. Thus, this package provides standardized visualizations, facilitating the development process of novel approaches.

2.3. Software functionalities

The functionalities implemented in VERONA satisfy the stages of the benchmarking framework described in Fig. 2. The complete list, shown in Table 2, is detailed below, following the order of the typical PPM pipeline:

- The first step when developing a new predictive monitoring approach is to obtain the event logs to be used. VERONA provides up to 27 publicly available event logs from real-life business processes that are commonly used in the process mining literature. The library allows the user to consult the entire list of available event logs and download each of them, both in CSV and XES [29] format, saving it to the path specified, and loading them into a *pandas dataframe* [30].
- To analyze the peculiarities of the event log and its traces, the tool provides a rich set of functions for extracting the main statistics from the dataset. The statistics include the number of traces, events, activities, and attributes, as well as the minimum, average, and maximum lengths and durations of cases, and the number of variants, which are sequences of different activities in the dataset.

- In the preprocessing stage, VERONA offers a suite of functions designed to enhance the manipulation of the event log. These functions facilitate multiple tasks, such as converting categorical attributes into numerical labels and extracting their one-hot vector representations, temporal sorting of events within a trace, and sorting of traces in the event log by their start date (timestamp of the first event).
- Predictive process monitoring, as a form of supervised learning, requires a training set for knowledge extraction and a test set for evaluation. Traditionally, these datasets are divided into three partitions, incorporating a validation set for model evaluation during training. These splits can be made by selecting the percentage of traces to be allocated to each partition (*hold-out*), or by ordering the traces temporally and setting dates from which all traces with a later start are placed in the next partition (*temporal split*). However, these methods may introduce bias due to data distribution differences [31]. To address this, [15] performs a *cross-validation* evaluation strategy, involving repeated application of *hold-out* partitioning with multiple test partitions. The library implements the three partitioning strategies, enabling the user to configure the parameters as desired or to fully replicate the experimental setup employed in the benchmark.
- One of the most time-consuming steps in encoding a predictive monitoring approach is the extraction of prefixes and their corresponding targets, the latter varying according to the predictive task at hand. VERONA automatically performs this operation by retrieving the tuples of prefixes-targets given a specific predictive task and partition, and allows many common encoding schemes, such as the ones proposed in [32]. The obtained prefixes can be used to select the last state (by setting the prefix size to 1), the last m states (where m is the size of the prefix), or an aggregate representation. The aggregate representation provides a single vector indicating the frequencies of the activities in the prefix, and statistical values such as maximum, minimum, mean, or sum can be obtained for numerical variables.
- An extensive list of metrics is provided to evaluate the predictive ability of the models depending on the task. Regarding the next activity prediction task, VERONA not only provides the usual metrics for classification tasks, such as accuracy, F1-Score, precision, and recall but also more nuanced metrics that allow gathering further insights about the performance of the approaches, such as F-beta, Matthews Correlation Coefficient (MCC) [33] and the Brier score loss [34].¹ For the suffix prediction task, the normalized Damerau-Levenshtein edit distance [35] is implemented, which takes into account the number of changes necessary to convert the predicted suffix into the ground truth suffix. Finally, for time-related predictive tasks, VERONA implements both the mean absolute error (MAE) and the mean squared error (MSE).
- The library also allows the user to import and manage the results of various approaches evaluated in [15] enabling the comparison of any other approach with them using the same Bayesian statistical tests.
- VERONA implements in Python the statistical tests available in *scmamp* [16], namely the Bayesian correlated t-test [27], the Bayesian version of the Wilcoxon signed-rank test [27], the hierarchical Bayesian correlated t-test [27], and the Bayesian Plackett-Luce ranking model [28].²
- Finally, the library includes tools for creating multiple graphical representations of metrics, statistical test results, and dataset statistics. Some of the available graphs for representing metrics include bar charts, line graphs, and box plots (see Fig. 4(a)),

¹ These metrics are thoroughly explained in the supplementary material.

² These statistical tests are described in detail in the supplementary material.

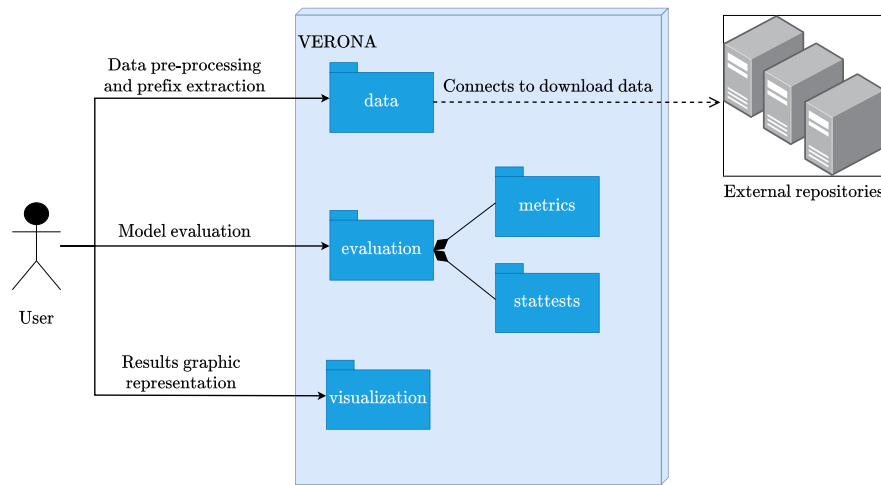


Fig. 3. VERONA software architecture.

Table 2
Summary of the functionalities implemented in VERONA.

Stage	Description	Supported methods
Event log retrieval	Retrieve any available event log from public sources.	– List available event log – Download event logs
Event log statistics	Obtain statistics about an event log.	– Activities and attributes – Variants – Case durations – Event durations
Event log transformation	Transform attributes, events and traces to prepare data to the predictive model input.	– Convert to categorical representation – Sort events and traces by their timestamps
Event log split	Divide the event log into training and test sets using a given partitioning strategy.	– Temporal split – Hold-out – Cross-validation
Prefix extraction	Extract prefixes and targets from a partition of the event log.	– Next activity/attribute prediction – Next timestamp prediction – Activity/attribute suffix prediction – Remaining time prediction
Model evaluation	Evaluate the performance of any predictive model.	– Accuracy, F-beta, Precision, Recall, MCC, Brier score loss – MAE, MSE – Metric by prefix size
Benchmark results	Import benchmark performance result data from the state-of-the-art approaches [15].	– Metrics results – Statistical tests results
Statistical tests	Compare the results of the predictive models using Bayesian statistical tests.	– Bayesian correlated t-test – Bayesian Wilcoxon signed-rank – Hierarchical Bayesian correlated t-test – Bayesian Plackett–Luce ranking model
Results visualization	Plot the results of a given predictive approach.	– Bar, line, box and error plots of the results – Custom plots by prefix size – Plot for Plackett–Luce rank test

as well as specialized graphs for metrics disaggregated by prefix length (see Fig. 4(b)). For statistical analyses, the Plackett–Luce rank test can be visualized through an error bar plot, as portrayed in Fig. 5.

3. Illustrative examples

In this section, we present examples of how to use the library and how to integrate its functionalities into the usual workflow of a PPM approach, minimizing the code required and the effort and time invested by the developer.

```

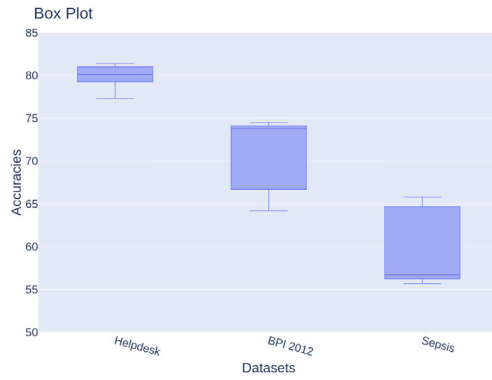
1 from verona.data import download, split, extractor
2 from verona.data import utils as data_utils
3
4 # Get BPI 2012 Complete event log
5 _, dataset = download.get_dataset('bpi2012comp',
    store_path='./data/csv', extension='csv')

```

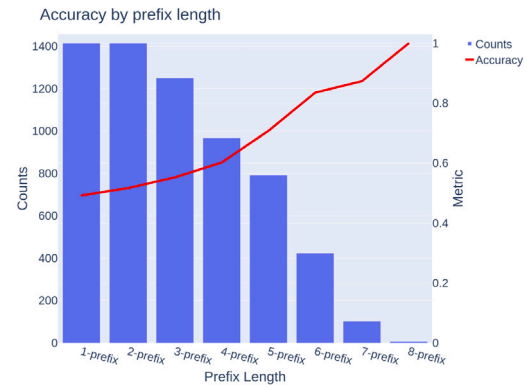
```

6
7 # Preprocess dataset
8 dataset = data_utils.unify_activity_and_lifecycle(
    dataset, activity_id='concept:name', lifecycle_id=
    'lifecycle:transition')
9 dataset['concept:name'], _, _ = data_utils.
    categorize_attribute(dataset['concept:name'])
10
11 # Split data using 5-folds cross-validation
12 CV_FOLDS = 5
13 train_dfs, val_dfs, test_dfs = split.
    make_crossvalidation(dataset, store_path='./data
    /csv/cv', cv_folds=CV_FOLDS, val_from_train=0.2)
14
15 for i in range(CV_FOLDS):
16     # Get prefixes and targets
17     train_prefixes, train_targets = extractor.
        get_prefixes_and_targets(train_dfs[i],
        prediction_task='activity_suffix')

```



(a) Box plot.



(b) Plot of metric disaggregated by prefix length.

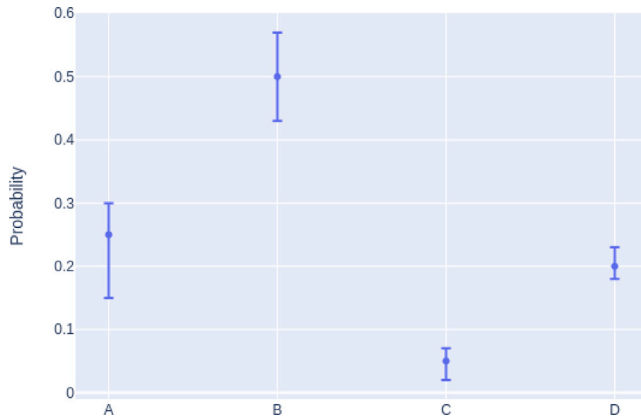
Fig. 4. Different plots available in *visualizations* package.

Fig. 5. Example of the Plackett-Luce error bar plot.

```

18 val_prefixes, val_targets = extractor.
   get_prefixes_and_targets(val_dfs[i],
   prediction_task='activity_suffix')
19 test_prefixes, test_targets = extractor.
   get_prefixes_and_targets(test_dfs[i],
   prediction_task='activity_suffix')

```

Listing 1: Example of the use of VERONA in the preprocessing stage.

The code fragment in listing 1 retrieves the dataset, preprocesses the attributes, partitions the data and extracts the prefixes in just a few lines of code. First, on line 5, the `get_dataset()` function of the `data.download` package downloads the selected event log from its repository in CSV format in the specified path, returning both the full path to the downloaded file and the dataset itself in Pandas DataFrame [30] format. Then, the initial pre-processing of the dataset utilizes the auxiliary functions available in the `data.utils` package to regularize the names of the process activities from their original names and their lifecycle status, following the benchmark [15] strategy (`unify_activity_and_lifecycle()`), and to convert the names of the activities in the dataset into numeric labels (`categorize_attribute()`). Later, at line 15, the `make_crossvalidation()` function from the `data.split` package partitions the dataset directly and transparently to the user, returning 3 lists with 5 partitions for training, validation, and testing respectively. Finally, for each folder, the `get_prefixes_and_targets()` function extracts the prefixes and their associated activity suffixes (`prediction_task='activity_suffix'`) without any programming effort by the user.

```

1 from verona.evaluation.metrics import suffix
2 from verona.evaluation.metrics import utils as
   metric_utils

```

```

3 from verona.visualization import metrics
4
5 # Get the global Damerau Levenshtein distance score
6 targets = np.array(list(targets.values()))
7 dl_score = suffix.get_damerau_levenshtein_score(
   predictions, targets, preds_format='onehot',
   gt_format='labels')
8
9 # Get the Damerau Levenshtein distance score per
   prefix length
10 prefix = list(prefixes.values())
11 dl_score_prefix_len = metric_utils.
   get_metric_by_prefix_len('damerau_levenshtein',
   predictions, targets, prefixes, preds_format='
   onehot', gt_format='labels')
12
13 # Plot the metric
14 fig = metric.plot_metric_by_prefixes_len(
   dl_score_prefix_len, metric_label='Damerau
   Levenshtein')
15 fig.show()

```

Listing 2: Example of the use of VERONA in the evaluation stage.

Listing 2 is an example of the use of VERONA functions for the evaluation of a model after it has been trained. Assuming that the predictions of the model on the test partition are stored in the variable `predictions`, the function `get_damerau_levenshtein_score()` of the package `evaluation.metrics.suffix` returns the value of the Damerau-Levenshtein score between these predictions and the targets extracted with the partitioning function used. Alternatively, the value of this metric disaggregated by prefix length is also calculated using the `get_metric_by_prefix_len()` function from the `evaluation.metrics.utils` package. The results are then plotted in line 14 using the `plot_metric_by_prefixes_len()` function from the `visualization.metric` package, which generates the specific graph shown in Fig. 4(b).

```

1 from verona.data.results import AvailableMetrics,
   MissingResultStrategy
2 from verona.evaluation.statstests.plackettluce import
   PlackettLuceRanking
3 from verona.visualization.statstests import
   plot_posteriors_plackett
4 from verona.evaluation.statstests.hierarchical import
   HierarchicalBayesianTest
5
6 mean_results, approaches = load_results_plackett_luce(
   AvailableMetrics.ActivitySuffix,
   DAMERAU_LEVENSHTEIN, MissingResultStrategy.
   DELETE_DATASET)
7 ranking = PlackettLuceRanking(mean_results,
   approaches)
8 results = ranking.run(suppress_output=True)
9
10 plot = plot_posteriors_plackett(results, save_path="
   ./posteriors.png", use_latex=True)
11

```

```

12 approach_1_df, approach_2_df, common_datasets =
    load_results_hierarchical("Tax", "Camargo_random"
        , metric=AvailableMetrics.ActivitySuffix.
        DAMERAU_LEVENSHTIN)
13
14 hierarchical_test = HierarchicalBayesianTest(
    approach_1_df, approach_2_df, approaches=["Tax",
        "Camargo_random"], datasets=common_datasets)
15 results = hierarchical_test.run(suppress_output=True)

```

Listing 3: Example of the usage of VERONA for statistical testing.

Regarding the usage of the statistical tests, the code fragment in listing 3 shows how the statistical tests are performed with the pre-computed results from the benchmark of [15] for the particular task of activity suffix prediction. First, in line 6, *load_results_plackett_luce()* function load the results for all the approaches, in order to perform the test with the results from the benchmark. Then in lines 7–8, we perform the ranking test using the default hyperparameters and seed from the benchmark, and we specify that the progress bars from the testing procedure should be hidden. Finally, line 12 shows how to create the error-bar plot from the results of the Plackett–Luce test, as shown in Fig. 5. On the other hand, lines 12 and 15 show how the hierarchical Bayesian test is performed using the *HierarchicalBayesianTest* class. Line 12 loads the individual cross-validation suffix prediction results for two specific approaches. Lines 14–15 show how this test is executed using the default hyperparameters used in the benchmark [15].

4. Impact and conclusions

The VERONA library poses a step forward in the field of predictive business process monitoring and deep learning. In particular, since VERONA provides accessibility utilities to reproduce the experimentation setup of a pre-existing benchmark [15], this greatly enhances the reuse of the previous results and the comparison of any other predictive monitoring approach. This increased availability of the results is essential for standardizing the evaluation method in the predictive monitoring field, bolstering the comparability and consistency on research outcomes.

On the other hand, by offering a tool that is easy to use, VERONA eases the research and development in the field, saving significant time and effort that would be otherwise dedicated to developing benchmarks from scratch or spent in the preprocessing stages of any new predictive monitoring approach. Thus, researchers can put their focus more on innovating and refining the models.

Finally, the practical implementation of VERONA will provide a valuable feedback loop that will allow the refinement of the benchmark [15]. Thus, the usage of the library will allow to spread of information about future updates on the benchmark, ensuring it remains relevant and effective in a constantly evolving field. Furthermore, VERONA will also serve as an educational tool for teaching and learning about predictive process monitoring, aiding in comprehending the most common concepts in this field in intersection with deep learning, and providing an easy hands-on learning experience.

Fundings

This work has received financial support from the Consellería de Educación, Universidade e Formación Profesional (accreditation 2019–2022 ED431G-2019/04), the European Regional Development Fund (ERDF), which acknowledges the CiTIUS - Centro Singular de Investigación en Tecnoloxías Intelixentes da Universidade de Santiago de Compostela as a Research Center of the Galician University System, and the Spanish Ministry of Science and Innovation (grants PDC2021-121072-C21 and PID2020-112623GB-I00).

CRediT authorship contribution statement

Pedro Gamallo-Fernandez: Writing – review & editing, Writing – original draft, Validation, Software, Methodology. **Efrén Rama-Maneiro:** Writing – review & editing, Validation, Software, Conceptualization. **Juan C. Vidal:** Writing – review & editing, Supervision. **Manuel Lama:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Pedro Gamallo-Fernandez reports financial support was provided by Spanish Ministry of Science and Innovation. If there are other authors, they 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

No data was used for the research described in the article.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at <https://doi.org/10.1016/j.softx.2024.101734>.

References

- [1] van der Aalst WMP. *Process mining - Data science in action*. 2nd ed. Springer; 2016.
- [2] Van Der Aalst W. *Process mining: Overview and opportunities*. *ACM Trans Manag Inf Syst (TMIS)* 2012;3(2):1–17.
- [3] Maggi FM, Di Francescomarino C, Dumas M, Ghidini C. *Predictive monitoring of business processes*. In: *Advanced information systems engineering - 26th international conference, CAISE 2014, Thessaloniki, Greece, June 16–20, 2014. proceedings. Lecture notes in computer science*, vol. 8484, Springer; 2014, p. 457–72.
- [4] Evermann J, Rehse J, Fettke P. *Predicting process behaviour using deep learning*. *CoRR abs/1612.04600*. 2016.
- [5] Rama-Maneiro E, Monteagudo-Lago P, Vidal JC, Lama M. *Encoder-decoder model for suffix prediction in predictive monitoring*. 2022, arXiv preprint [arXiv:2211.16106](https://arxiv.org/abs/2211.16106).
- [6] Tax N, Verenich I, Rosa ML, Dumas M. *Predictive business process monitoring with LSTM neural networks*. In: *Advanced information systems engineering - 29th international conference*. *Lecture notes in computer science*, vol. 10253, Springer; 2017, p. 477–92.
- [7] Navarin N, Vincenzi B, Polato M, Sperduti A. *LSTM networks for data-aware remaining time prediction of business process instances*. In: *2017 IEEE symposium series on computational intelligence*. IEEE; 2017, p. 1–7.
- [8] Di Francescomarino C, Dumas M, Maggi FM, Teinmaa I. *Clustering-based predictive process monitoring*. *IEEE Trans Serv Comput* 2019;12(6):896–909.
- [9] Cabanillas C, Di Ciccio C, Mendling J, Baumgras A. *Predictive task monitoring for business processes*. In: *Business process management - 12th international conference. Lecture notes in computer science*, vol. 8659, Springer; 2014, p. 424–32.
- [10] Verenich I, Dumas M, La Rosa M, Maggi FM, Di Francescomarino C. *Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring*. In: *Business process management workshops: BPM 2015, 13th international workshops*. Springer; 2016, p. 218–29.
- [11] Lee WLJ, Parra D, Munoz-Gama J, Sepulveda M. *Predicting process behavior meets factorization machines*. *Expert Syst Appl* 2018;112:87–98.
- [12] Federici M, Rizzi W, Di Francescomarino C, Dumas M, Ghidini C, Maggi FM, et al. *A prom operational support provider for predictive monitoring of business processes*. In: *Proceedings of the BPM demo session 2015; co-located with the 13th international conference on business process management*. CEUR-WS; 2015, p. 1–5.
- [13] Rizzi W, Simonetto L, Di Francescomarino C, Ghidini C, Kasekamp T, Maggi FM. *Nirdizati 2.0: New features and redesigned backend*. In: Depaire B, Smedt JD, Dumas M, Fahland D, Kumar A, Leopold H, Reichert M, Rinderle-Ma S, Schulte S, Seidel S, van der Aalst WMP, editors. *Proceedings of the dissertation award, doctoral consortium, and demonstration track at BPM 2019 co-located with 17th international conference on business process management*. CEUR workshop proceedings, vol. 2420, CEUR-WS.org; 2019, p. 154–8, URL <https://ceur-ws.org/Vol-2420/paperDT8.pdf>.

- [14] Berti A, van Zelst SJ, Schuster D. PM4Py: A process mining library for Python. *Softw Impacts* 2023;17:100556.
- [15] Rama-Maneiro E, Vidal JC, Lama M. Deep learning for predictive business process monitoring: Review and benchmark. *IEEE Trans Serv Comput* 2023;16(1):739–56.
- [16] Calvo B, Santafé G. Scmamp: Statistical comparison of multiple algorithms in multiple problems. *R J* 2016;8(1):248.
- [17] Camargo M, Dumas M, Rojas OG. Learning accurate LSTM models of business processes. In: *Business process management 17th international conference. Lecture notes in computer science*, vol. 11675, Springer; 2019, p. 286–302.
- [18] Hinkka M, Lehto T, Heljanko K. Exploiting event log event attributes in RNN based prediction. In: *Data-driven process discovery and analysis - 8th IFIP WG 2.6. Lecture notes in business information processing*, vol. 379, Springer; 2019, p. 67–85.
- [19] Khan MA, Le H, Do K, Tran T, Ghose A, Dam KH, Sindhgatta R. Memory-augmented neural networks for predictive process analytics, *CoRR abs/1802.00938*. 2018.
- [20] Di Mauro N, Appice A, Basile TMA. Activity prediction of business process instances with inception CNN models. In: *AI*IA 2019 - Advances in artificial intelligence - XVIIIth international conference of the Italian association for artificial intelligence. Lecture notes in computer science*, vol. 11946, Springer; 2019, p. 348–61.
- [21] Pasquabiscegile V, Appice A, Castellano G, Malerba D. Using convolutional neural networks for predictive process analytics. In: *International conference on process mining. IEEE*; 2019, p. 129–36.
- [22] Theis J, Darabi H. Decay replay mining to predict next process events. *IEEE Access* 2019;7:119787–803.
- [23] Di Francescomarino C, Ghidini C, Maggi FM, Petrucci G, Yeshchenko A. An eye into the future: Leveraging A-priori knowledge in predictive business process monitoring. In: *Business process management - 15th international conference. Lecture notes in computer science*, vol. 10445, Springer; 2017, p. 252–68.
- [24] Taymouri F, Rosa ML, Erfani SM. A deep adversarial model for suffix and remaining time prediction of event sequences. In: *Demeniconi C, Davidson I, editors. Proceedings of the 2021 SIAM international conference on data mining. SIAM*; 2021, p. 522–30.
- [25] Bukhsh ZA, Saeed A, Dijkman RM. ProcessTransformer: Predictive business process monitoring with transformer network, *CoRR abs/2104.00721*. 2021, arXiv:2104.00721.
- [26] Weinzierl S. Exploring gated graph sequence neural networks for predicting next process activities. In: *Marrella A, Weber B, editors. Business process management workshops. Lecture notes in business information processing*, vol. 436, Springer; 2021, p. 30–42.
- [27] Benavoli A, Corani G, Demšar J, Zaffalon M. Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. *J Mach Learn Res* 2017;18:77:1–36.
- [28] Calvo B, Ceberio J, Lozano JA. Bayesian inference for algorithm ranking analysis. In: *Proceedings of the genetic and evolutionary computation conference companion. ACM*; 2018, p. 324–5.
- [29] IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams. *IEEE Std 1849-2016* 2016;1–50.
- [30] McKinney W. Data structures for statistical computing in python. In: *Proceedings of the 9th Python in science conference*. 2010, p. 56–61.
- [31] Hastie T, Tibshirani R, Friedman JH. *The elements of statistical learning: Data mining, inference, and prediction*. 2nd ed. Springer; 2009.
- [32] Teinemaa I, Dumas M, Rosa ML, Maggi FM. Outcome-oriented predictive process monitoring: Review and benchmark. *ACM Trans Knowl Discov Data (TKDD)* 2019;13(2):1–57.
- [33] Matthews B. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochim Biophys Acta (BBA) - Protein Struct* 1975;405(2):442–51.
- [34] Brier GW. Verification of forecasts expressed in terms of probability. *Mon Weather Rev* 1950;78(1):1–3.
- [35] Boytsov L. Indexing methods for approximate dictionary searching. *ACM J Exp Algorithmics* 2011;16. <http://dx.doi.org/10.1145/1963190.1963191>.