The Recommender: A Decision Support Tool for Predictive Business Process Monitoring

Christoph Drodt[1], Sven Weinzierl[2], Martin Matzner[2] and Patrick Delfmann[1]

[1]Institute for IS Research, Universität Koblenz-Landau, Koblenz, Germany
[2]Institute of Information Systems, Friedrich-Alexander-Universität Erlangen-Nürnberg, Nuremberg, Germany

Abstract

Predictive business process monitoring (PBPM) provides a set of techniques to optimize the performance of operational business processes. Most recent PBPM techniques learn predictive models from historical event log data using machine learning algorithms (ML). However, there is no *silver bullet* approach for different event logs, and their performance depends on the characteristics of the underlying event logs. This paper demonstrates the decision support tool *Recommender*. The main idea of our tool is to recommend an appropriate pre-processing procedure, an ML algorithm, and the hyper-parameter configuration for a new event log based on its characteristics. While our tool can support researchers to better understand the relation between event log characteristics and ML-driven PBPM techniques, it supports practitioners in developing effective PBPM techniques.

Keywords

Predictive Business Process Monitoring, Machine Learning, Business Process Management, Process Mining, Decision Support

1. Introduction

Over the last years, business process management (BPM) researchers have developed a plethora of predictive business process monitoring (PBPM) techniques. PBPM techniques aim to predict aspects such as next activities, process outcomes, or next timestamps in running business processes. Based on these predictions, process stakeholders can proactively intervene in running business processes. In doing that, process stakeholders can improve the performance of operational business processes by mitigating risks or avoiding failures before these occur, or exploiting potentials in time [1].

Most recent PBPM techniques generate predictions through predictive models learned from historical event log data using machine learning (ML) algorithms [2]. Driven by the goal to achieve accurate predictions, many techniques with different data pre-processing procedures, ML algorithms, and hyper-parameter configurations have been proposed [3, 4].

However, PBPM research has shown that there is no *silver bullet* approach for event logs

Proceedings of the Demonstration & Resources Track, Best BPM Dissertation Award, and Doctoral Consortium at BPM 2021 co-located with the 19th International Conference on Business Process Management, BPM 2021, Rome, Italy, September 6-10, 2021

drodt@uni-koblenz.de (C. Drodt); sven.weinzierl@fau.de (S. Weinzierl)
http://fg-bks.uni-koblenz.de/ (C. Drodt)
0000-0002-4682-8036 (C. Drodt); 0000-0003-2268-7352 (S. Weinzierl); 0000-0001-5244-3928 (M. Matzner)

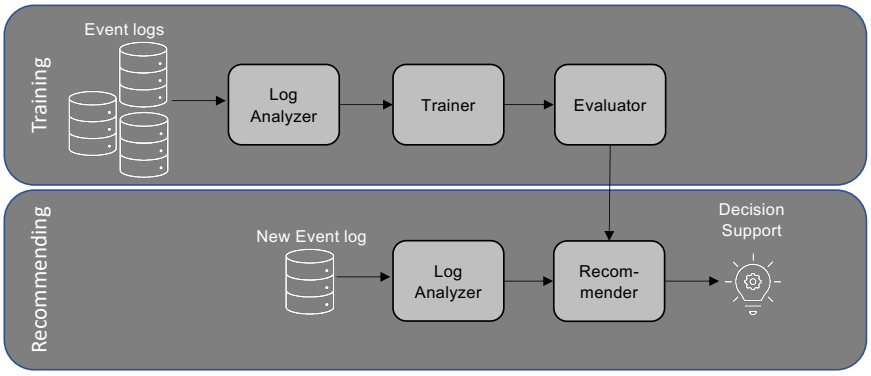

Figure 1: Four Components of the *Recomminder*

with different characteristics, even if predictive models are learned using deep learning (DL)[5]. Instead, most approaches are typically optimized based on a specific set or type of event logs but often show a poor generalization ability if applied to other event logs. In general, event logs differ regarding their characteristics, e.g., the number of activities, number of trace variants, or number of categorical data attributes. So far, little is known about the relation between event log characteristics and ML-driven approaches in BPM research. As a logical consequence of this missing understanding, it is challenging for practitioners to adopt an approach designed and implemented by research because their available event logs may comprise characteristics to that such an approach is generally not geared.

To overcome this challenge, we demonstrate the decision support tool *Recomminder* in this paper for the task of predicting next activities. The tool is mainly structured into the four components *LogAnalyzer*, *Trainer*, *Evaluator*, and *Recommender* (see Fig. 1). For a given set of event logs, the *LogAnalyzer* component first determines event log characteristics. Second, the *Trainer* component learns predictive models for different constellations of pre-processing procedures, ML algorithms, and hyper-parameter configurations. Third, the *Evaluator* tests the predictive models and calculates ML metrics to assess the models' predictive quality. Forth, the *Recomminder* component selects per event log the predictive model with the highest predictive quality and creates a meta predictive model in form of a decision tree (DT) that learns the mapping from the event log characteristics to the best performing (non-meta) predictive models. Finally, the meta predictive model can be applied to a new event log to determine an appropriate pre-processing procedure, ML algorithm, and hyper-parameter configuration based on its characteristics.

2. Backend

In this section, we explain the structure of the Recomminder and elaborate its technical background. The presented tool is written in Python 3.8.2. and is designed as a Python package, split into two main modules: *backend* (this section) and *frontend* (Section 3). The backend is organized into four components, namely *Log Analyzer*, *Trainer*, *Evaluator* and *Recommender*,

which will be described in more detail in this section. Advanced users can access these components and their modules by simply importing them and, thus, accessing their methods in their code. Additionally, we designed two phases that represent the fundamental workflows of Recomminder: the offline phase (Training/Feeder) and the online phase (Recommending) (see Fig. 1 and Section 1). On the coding perspective, those phases can be executed by calling *Recomminder.train()*, and *Recomminder.recommend()*. To make the features of the Recomminder also available for non-developers, we designed a web-based frontend that can be started via the shell or command line. For more details about the frontend, please see Section 3.

Another important aspect is to allow other developers to extend this artifact, especially the Trainer component, by implementing more classifiers. We developed a superclass to support developers in implementing further classifiers. More details on executing, extending, and installing the Recomminder can be found in the repository (see Section 4).

2.1. Log Analyzer

In terms of the feature extraction, we consider three different types of event log characteristics: *event-log-based* (e.g., concerning events, activities, and traces), *process-model-based* (e.g., concerning loops, noise, variants, and gateways), and *process-context based* features (e.g., concerning categorical attributes and numerical attributes). To extract some of these features, a Process Discovery algorithm has to produce a process model. In this tool, we used PM4PY[1]'s data structure for event logs and its Heuristic Miner [6] implementation to create a Petri net of a event log file. The Heuristic Miner is a common discovery algorithm in process mining (PM) that can, e.g., handle loops and noise in event log data [7, 8]. After feature extraction, the results are stored in an SQLite database for later use.

2.2. Trainer

To evaluate the best classifier for a given event log file, a predictive model must be trained for each classifier. In the Recomminder tool, we implemented two classifiers using existing ML frameworks: Random Forest[2][9] and LSTM[3][10]. The presented tool also provides some methods for pre-processing the event log. This includes event encoding methods (ordinal and one-hot encoding for categorical attributes and min-max normalization for continuous attributes), as well as sequence encoding techniques (window-based and index-based prefix generation). In addition, we implemented a test-train-split method to generate test and train data sets. Before executing the training, we evaluated the best fitting hyper-parameters using the Optuna[4] framework with 20 optimization rounds and a subset of the training set (90% training and 10% test split). Finally, the results are stored in the database. After each classifier has been trained, the model is transferred to the Evaluator.

[1] https://pm4py.fit.fraunhofer.de
[2] https://scikit-learn.org/
[3] https://www.tensorflow.org
[4] https://optuna.org

2.3. Evaluator

This component evaluates different common metrics (accuracy, precision, recall and f1 score) and stores them in the database. Once all Trainer processes have finished, the best classifier for an event log file is extracted and used as target values. In addition, the Evaluator gathers corresponding extracted features of the log which are used as input samples. Those data is the training set for the DT. The training of the DT is performed by the Evaluator and the resulting model represents the meta predictive model for the Recommender. Finally, we retrieve feature importance values from the trained meta model, that is the outcome of this component, and store a visualisation of the evaluation.

2.4. Recommender

As stated in the previous section, the Recommender components rely on a DT which is trained with the results from the Log Analyzer and Trainer. By calling the *Recomminder.recommend()* method, the Recommender uses the Log Analyzer to extract the feature of a given event log file and feeds them to the trained DT model. Following, the best matching classifier is returned.

3. Frontend

In addition to the developer access level, we also provide a rich, web-based frontend to allow non-developers to use the Recomminder. To implement the frontend, this tool uses TurboGears2[5] for content delivery, Bootstrap[6] to style the web pages, and jQuery[7] for asynchronous functions. The navigation is split into the two workflows described previously and is located at the top of the page. Stepping into one phase, the tool provides a sub-navigation that leads the user through the necessary steps. In both phases, users can upload event log files via an asynchronous upload script, so that multiple event logs can be provided easily. The frontend directly interacts with the backend and starts both functions (train end recommend) as a new thread. Using the threading extension of python, the frontend can proceed without waiting for the new process to finish. Further on, the frontend shows a live view on the Recomminder's log file, so users can always follow current process steps and retrieve the process. Finally, the frontend presents the results and in case of the training phase, it also includes three figures, including a representation of the DT, a plot of the metrics, and a bar chart stating the feature importance. Every progress is stored in the session, generated by TurboGears2, which allows users to return to the frontend after some time and pick the process up where they left it.

4. Conclusion

This paper demonstrated the decision support tool *Recominder*, consisting of four components that can support research and practice. With our tool, researchers can better understand the relationship between event log characteristics and ML-driven PBPM techniques. On the other

[5] https://turbogears.org
[6] https://getbootstrap.com
[7] https://jquery.com

hand, our tool can support practitioners in the development of effective PBPM techniques. In future research, we plan to extend the tool by:

- further prediction tasks such as the next timestamp prediction,
- further granularities of prediction tasks such as prediction per prefix size or prediction per decision point,
- further ML algorithms to learn predictive models,
- further sequence and event encoding techniques,
- further intrinsic explainable ML algorithms such as linear regression to learn the mapping between event log characteristics and the best-performing predictive models, and
- ensemble learning techniques such as stacking or boosting to improve the prediction accuracy by combining several predictive models.

A demonstration video of the *Recommender* can be found at https://youtu.be/37ikmt9g818. The source code is of the decision tool is available under the Lesser GNU Public License (LGPL) at https://gitlab.uni-koblenz.de/fg-bks/predictive-recommining.

References

[1] A. E. Marquez-Chamorro, M. Resinas, A. Ruiz-Cortes, Predictive Monitoring of Business Processes: A Survey, IEEE Transactions on Services Computing 11 (2018) 962–977.

[2] C. Di Francescomarino, C. Ghidini, F. M. Maggi, F. Milani, Predictive Process Monitoring Methods: Which One Suits Me Best?, in: Business Process Management, volume 11080, Springer International Publishing, 2018, pp. 462–479.

[3] K. Heinrich, P. Zschech, C. Janiesch, M. Bonin, Process Data Properties Matter: Introducing Gated Convolutional Neural Networks (GCNN) and Key-Value-Predict Attention Networks (KVP) for Next Event Prediction with Deep Learning, Decision Support Systems 143 (2021) 113494.

[4] S. Weinzierl, S. Zilker, J. Brunk, K. Revoredo, A. Nguyen, M. Matzner, J. Becker, B. Eskofier, An Empirical Comparison of Deep-Neural-Network Architectures for Next Activity Prediction Using Context-Enriched Process Event Logs, arXiv (2020).

[5] W. Kratsch, J. Manderscheid, M. Röglinger, J. Seyfried, Machine Learning in Business Process Monitoring: A Comparison of Deep Learning and Classical Approaches Used for Outcome Prediction, Business & Information Systems Engineering 63 (2021) 261–276.

[6] A. Weijters, W. M. P. van der Aalst, A. K. Alves de Medeiros, Process Mining with the HeuristicsMiner Algorithm, Citeseer, 2006.

[7] A. P. Kurniati, G. Kusuma, G. Wisudiawan, Implementing heuristic miner for different types of event logs, International Journal of Applied Engineering Research 11 (2016) 5523–5529.

[8] P. Weber, B. Bordbar, P. Tino, A principled approach to mining from noisy logs using Heuristics Miner, in: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2013, pp. 119–126.

[9] L. Breiman, Random Forests, Machine Learning 45 (2001) 5–32.

[10] S. Hochreiter, J. Schmidhuber, Long Short Term Memory. Neural Computation, Neural Computation 9 (1997) 1735–1780.