processpredictR: predictice process monitoring using bupaR

Ivan Esin[1], Dmitri Beloshitskiy[1] and Gert Janssenswillen[1,*]

[1]UHasselt - Faculty of Business Economics, Agoralaan, 3590 Diepenbeek, Belgium

Abstract
This demo paper introduces `processpredictR`, a new library for predictive process monitoring in the `bupaR`-ecosystem. The library provides functionalities with different levels of customization, from completely standard off-the-shelf models, to tools for advanced customization of preprocessing and model configuration.

Keywords
Predictive Process Analytics, Predictive Process Monitoring, Transformer, bupaR

1. Introduction

The field of predictive business process monitoring aims to improve the execution of processes by analyzing event logs and foreseeing future events and process outcome. Over the past decade, much research has been done in this field, comparing different model architectures and improving predictive accuracy. Implementing these state-of-the-art predictive models is often done in an ad-hoc way, using generic tools such as TensorFlow and Keras, creating a barrier to smooth adoption by practitioners. Notable exceptions on this front are Apromore [1] and Nirdizati [2]. In this demo paper, we extend this available off-the-shelf process prediction tool support with `processpredictR`.

`processpredictR` is an extension of the `bupaR` ecosystem for process analysis with R [3]. The models provided are based on the Transformer architecture proposed in [4]. The tool is designed to be used with varying levels of customisation, supporting entirely standardised models — which can be used to get familiar with predictive process monitoring without the need for far-fetched configurations — as well as models that can be highly customized, while still relying on `processpredictR` for typical steps such as data preparation and evaluation.

ICPM'23 Demo Track: International Conference on Process Mining, October 23–27, 2023, Rome, Italy

*Corresponding author.

ivan.esin@student.uhasselt.be (I. Esin); dmitri.beloshitskiy@student.uhasselt.be (D. Beloshitskiy); gert.janssenswillen@uhasselt.be (G. Janssenswillen)
0000-0002-7474-2088 (G. Janssenswillen)

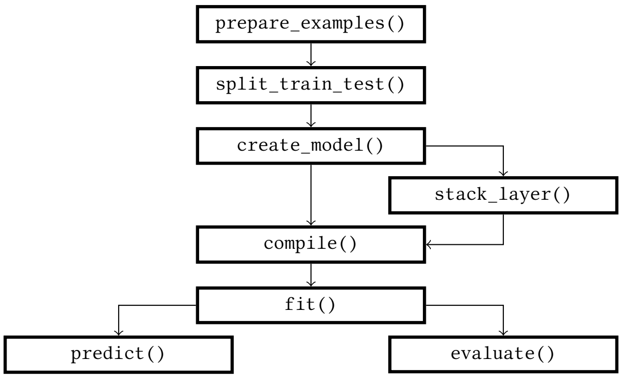

Figure 1: Main workflow.

2. Features

2.1. Main workflow

In the next paragraphs, we briefly describe the main workflow, as shown in Figure 1.

Prepare examples. The first step is preparing the dataset using the `prepare_examples` function. When preparing the examples for prediction, the user can select one of the five different prediction tasks: *next activity*, *next time*, *remaining trace*, *remaining time*, and *outcome*. The latter one, outcome, can be defined in various ways, e.g. based on the final activity, some attribute, or some other logical condition. At this point, it is also possible to specify any additional features that should be used in the prediction, next to the default ones, i.e. the activity prefix and time features. An example of the preparation step is shown in Listing 1, line 2-4.

Listing 1: Coding workflow.

```
1  library(processpredictR)
2  examples <- prepare_examples(log,
3                               task = ”outcome”,
4                               features = c(..., ...))
5
6  split <- split_train_test(examples, split = 0.7)
7  train <- split$train_df
8  test <- split$test_df
9
10 model <- create_model(
11         train, # training set
12         custom = FALSE, # default architecture
13         num_heads = 4, # number of attention heads
14         output_dim_emb = 36, # embeddings output dimensions
15         dim_ff = 64) # dimensions of feed-forward network
16
17 model <- compile(model)
18
19 hist <- fit(model, train_data=train, epochs=10)
20
21 predictions <- predict(model, test_data=test, output=’append’)
22
23 confusion_matrix(predictions)
24 plot(predictions)
```

```
                                                    25
evaluate(model, test_data=test)                     26
```

The output of this step is a datatable with examples. A case containing n activity instances is turned into $n - 1$ examples, each consisting of an activity prefix, the target variable (depending on the specified task), and any other features as specified in the call.

Split train test. The preprocessed dataset can be divided into train and test sets using the `split_train_test` function, simplifying dataset separation for model training and evaluation. This function takes the output of `prepare_examples()` and generates two data frames, as detailed in Listing 1, line 6-8.

Note that the event log is split chronologically, ensuring that all train examples precede test examples. Additionally, the split is based on case proportions rather than individual examples, preventing the division of the last case between the training and test sets, which would otherwise lead to observations scattered between both sets.

Create model. The next workflow step involves defining the model's structure. The `create_model()` function considers the task and dataset parameters (e.g. additional features, output possibilities, case length) and initializes a default transformer architecture (based on [4]). `create_model()` simplifies layer assembly and predefined dataset-based hyperparameters, providing an automated solution. Moreover, the model's complexity can be manually fine-tuned by adjusting hyperparameters like attention heads, embedding output dimensions, and feed-forward network dimensions (Listing 1, line 10-15).

Compile. To configure the training process the model must be compiled, specifying the optimization process, loss function and evaluation metrics. This is done automatically by the `compile` function (see Listing 1 l.17).

Fit. The training of the model is facilitated with the provided `fit` method (see Listing 1 line 19). It naturally allows flexibility in selecting hyperparameters (e.g. the number of training epochs, batch size, and validation split).

Predict. Having trained the model, the predictions can be made using the provided `predict` method. The function can return up to three different types of output: raw predicted values as returned by the generic keras `predict` function (without argmax applied), an array of predicted values, or a data frame combining the input data with the predicted values (as illustrated in Listing 1 line 21). Additionally, for classification tasks, `processpredictR` allows to quickly compute and visualize the confusion matrix (Listing 1 line 23-24).

Evaluate. The `evaluate` function allows to quickly evaluate the performance of the model on the test set, returning the value of the loss and accuracy metrics (see Listing 1 line 26).

2.2. Customization

Aside from the default models as in the main workflow, `processpredictR` provides customization options. Users can include extra categorical and/or numerical features alongside examples and adjust the model's complexity.

Additional features. Additional features to be used by the model, beyond examples (activity sequences) and default features, can be defined when using `prepare_examples()`. The features can be either numerical variables or categorical factors. Numerical variables will be automatically scaled using the min-max normalization. Factors will be automatically converted to hot-encoded variables.

Model complexity and custom layers. The default complexity settings for the models are determined by the following parameters which can be adjusted by the user: the number of attention heads ($num\_heads = 4$), the output dimensions of the embeddings ($output\_dim\_emb = 36$), and the dimensions of the feed-forward network ($dim\_ff = 64$) [5].

More advanced flexibility is offered by allowing the creation of a custom model. In such a case, `processpredictR` will provide a partial model, containing only the input layers and the encoder block of the model, which the user can then complete with generic `keras` layers using the provided `stack_layer` method.

Advanced customization. The methods described thus far abstract from many preprocessing and configurations steps. However, one may be interested in custom preprocessing, model architecture or simply getting a more detailed interpretation and understanding of the model parameters. Hence, additional auxiliary functions are made available, which together with generic `keras` methods offer even greater flexibility.

3. Maturity

While the `processpredictR` library was published only in January 2023, the back-end is based on the tried and tested functionalities of the keras library [6]. Furthermore, it is embedded in the `bupaR` eco-system [3], which has built up a strong user base over the last couple of years.

4. Further materials

The library is published on CRAN[1] and can be installed as a regular R-package. Usage does require a Python distribution to be installed also. More information on the installation can be found on docs.bupar.net.[2] A more detailed tutorial on the workflow as well as customization options can be found there as well.[3] A four minute screen-cast on `processpredictR` can be found here: https://tinyurl.com/processpredictR

5. Conclusions and Future Work

This demo paper introduced `processpredictR`, an easy-to-use tool for predictive process monitoring embedded in the `bupaR`-ecosystem. Next to default off-the-shelf models, the library provides multiple avenues to customize predictive models within a specific context. Future efforts will be focused on improving the default model architectures used, as well as more advanced visual tools to aid the interpretability of the obtained models and predictions.

[1]https://cran.r-project.org/package=processpredictR
[2]https://bupaverse.github.io/docs/install.html#Installing_bupaR
[3]https://bupaverse.github.io/docs/predict_workflow.html

References

[1] M. La Rosa, H. A. Reijers, W. M. Van Der Aalst, R. M. Dijkman, J. Mendling, M. Dumas, L. García-Bañuelos, Apromore: An advanced process model repository, Expert Systems with Applications 38 (2011) 7029–7040.

[2] W. Rizzi, L. Simonetto, C. Di Francescomarino, C. Ghidini, T. Kasekamp, F. M. Maggi, Nirdizati 2.0: New features and redesigned backend, Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019 2420 (2019) 154–158.

[3] G. Janssenswillen, B. Depaire, M. Swennen, M. Jans, K. Vanhoof, bupar: Enabling reproducible business process analysis, Knowledge-Based Systems 163 (2019) 927–930.

[4] Z. A. Bukhsh, A. Saeed, R. M. Dijkman, ProcessTransformer: Predictive Business Process Monitoring with Transformer Network, arXiv:2104.00721 [cs] (2021). ArXiv: 2104.00721.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention Is All You Need, arXiv:1706.03762 [cs] (2017). ArXiv: 1706.03762.

[6] F. Chollet, et al., Keras, 2015. URL: https://github.com/fchollet/keras.