The Journal of Open Source Software

tpcp: Tiny Pipelines for Complex Problems - A set of framework independent helpers for algorithms development and evaluation

Arne Küderle [1¶], **Robert Richer** [1], **Raul C. Sîmpetru** [2], **and Bjoern M. Eskofier** [1]

1 Machine Learning and Data Analytics Lab (MaD Lab), Department Artificial Intelligence in Biomedical Engineering (AIBE), Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) **2** Neuromuscular Physiology and Neural Interfacing (N-squared) Laboratory, Department Artificial Intelligence in Biomedical Engineering (AIBE), Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) ¶ Corresponding author

DOI: 10.21105/joss.04953

Software

- Review
- Repository
- Archive

Editor: Adi Singh

Reviewers:

- @xtruan
- @paxtonfitzpatrick

Submitted: 11 August 2022
Published: 13 February 2023

Summary

During algorithm development and analysis researchers regularly use software libraries developed for their specific domain. With such libraries, complex analysis tasks can often be reduced to a couple of lines of code. This not only reduces the amount of implementation required but also prevents errors.

The best developer experience is usually achieved when the entire analysis can be represented with the tools provided by a single library. For example, when an entire machine learning pipeline is represented by a `scikit-learn` pipeline (Pedregosa et al., 2018), it is extremely easy to switch out and train algorithms. Furthermore, train/test leaks and other methodological errors at various stages in the analysis are automatically prevented – even if the user might not be aware of these issue.

However, if the performed analysis gets too complex, too specific to an application domain, or requires the use of tooling and algorithms from multiple frameworks, developers lose a lot of the benefits provided by individual libraries. In turn, the required skill level and the chance of methodological errors rise.

With `tpcp` we attempt to overcome the issue by providing higher-level tooling and structure for algorithm development and evaluation that is independent of the frameworks required for the algorithm implementation.

Statement of Need

To better understand the need for `tpcp`, we want to provide two examples from application fields:

The first example is a comparison of different algorithms for sleep/wake detection based on wearable sensor data. These algorithms can either be heuristic/rule-based algorithms, "traditional" machine learning (ML) algorithms, or deep learning (DL) approaches (Palotti et al., 2019). When one attempts to compare multiple algorithms, it is not possible to use just a single high-level framework to implement and run all of them. Heuristic algorithms will most likely be implemented without specific frameworks, ML approaches are most likely based on `scikit-learn` (Pedregosa et al., 2018), and Deep Learning approaches based on `tensorflow` (Abadi et al., 2015) or `PyTorch` (Paszke et al., 2019). Further, the required data are usually multimodal time series (e.g., motion and cardiac data) (Zhai et al., 2020). Some algorithms might just require a subset of these modalities, which further complicates the overall data handling and potential cross-validations for algorithms evaluation. Additionally, a window-wise prediction of sleep and wake is desired.

Without tpcp, researchers would most likely develop their own set of helper functions to load and handle the data, to split data in train and test sets, and to perform cross-validation. Afterwards, they would need to create their own wrapper to define a unified interface for all algorithms so that they can compare all of them in a similar manner. All of this requires a profound understanding of machine learning to implement the train-test split and cross-validation, as well as extensive experience in the programming language of choice to design and implement an algorithm interface.

The second example is a comparison of stride detection algorithms based on IMU data recently published by Roth et al. (2021). The authors compared a custom Hidden Markov Model implemented using pomegranate (Schreiber, 2017) with an implementation of a template matching algorithm based on Barth et al. (2013). In their data, two recordings were available per participant – one in a controlled lab setting and one from an unsupervised recording at home. As part of their analysis, the authors wanted to show that it is sufficient to train algorithms based on the lab data without labeled data from the home environment required. The overall approach leads to a set of challenges: Neither algorithm fit in the realm of the typical ML frameworks, that would provide suitable helper for validation. Thus, custom helpers were required again to come up with uniform interfaces for training and running the algorithms. Further, the requirements for which data were used during training and testing is something that cannot be easily abstracted by any of the existing frameworks, even if all algorithms could be implemented in it.

While both examples could be (and have been) solved using additional custom tooling, the loss of a framework to support and guide the implementation raises the required software engineering skill and required understanding of the evaluation procedure. Further, developing custom algorithm interfaces and tooling for each analysis makes it difficult to reuse algorithms and pipelines across projects, as interfaces are likely to differ. With tpcp, we provide opinionated helpers to support data handling and evaluation via cross-validation, as well as interfaces that can guide the development of custom data analysis pipelines, independent of the underlying algorithms. This should ensure a more straightforward software development process and should simplify the reuse of tooling and algorithms across projects.

However, compared to a more specialized framework (e.g. scikit-learn), tpcp will always require more implementation from the developer side and can never provide an interface that is equally simple. This means, if an analysis could be done in the context of an already existing specialized library, this library should be used over tpcp. However, if an analysis spans multiple domains or requires flexibility that specialized frameworks cannot provide, tpcp provides an alternative that should be considered before switching to fully custom tooling.

Provided Functionality

The package tpcp provides three things:

1. Helper to create object-oriented **dataset** accessors
2. Helper to implement own **algorithms** and **pipelines** in an object-oriented fashion
3. Tools for **parameter optimization** and **algorithm evaluation** that work with the other structures

Beyond that, the documentation of tpcp attempts to provide fundamental information and recipes on how to approach algorithm development and algorithm evaluation.

The Journal of Open Source Software

Datasets

In cases where data points cannot be expressed by a simple feature vector, data loading and handling require non-negligible code complexity. Data is usually spread over multiple files and databases and requires data transformations during the loading process. Therefore, the resulting data structures are unlikely to be compatible with existing algorithms. Hence, researchers need to implement code abstractions of their datasets, often in the form of helper functions. With the `tpcp.Dataset` implementation, we suggest an alternative interface to diverse data structures by implementing data access using Python classes. Inspired by `pytorch` datasets, they provide a common interface and their structure can be iterated, filtered, and split. These datasets are compatible with other tooling provided in `tpcp` and allow to pass complex data structures through a cross-validations or gridsearch.

Algorithms

In `tpcp`, we do not provide any specific algorithm implementations, but only simple base classes to build algorithms with a `scikit-learn` inspired interface. Using this object-oriented interface to implement algorithms ensures comparable interfaces for similar algorithms. Using this part of `tpcp` is completely optional (i.e., all other features are completely independent of the algorithm implementation), but following our recommendations can simplify the integration with other parts of `tpcp`.

Pipelines

For any analysis, we need to bring the data together with the algorithms. In `tpcp`, we call this "gluing code" Pipelines. Many specialized frameworks are able to completely remove any of this gluing code as the data structures and the algorithm interfaces are strictly defined and, hence, algorithms can directly interface with the data. In `tpcp` we allow more flexibility to have different data and algorithm interfaces depending on the application and algorithm types. Therefore, we need Pipelines to connect the reusable Dataset and Algorithm interfaces for a specific analysis (Figure 1, Figure 2). Pipelines also provide a fixed and unified interface that utility methods in `tpcp` can use.

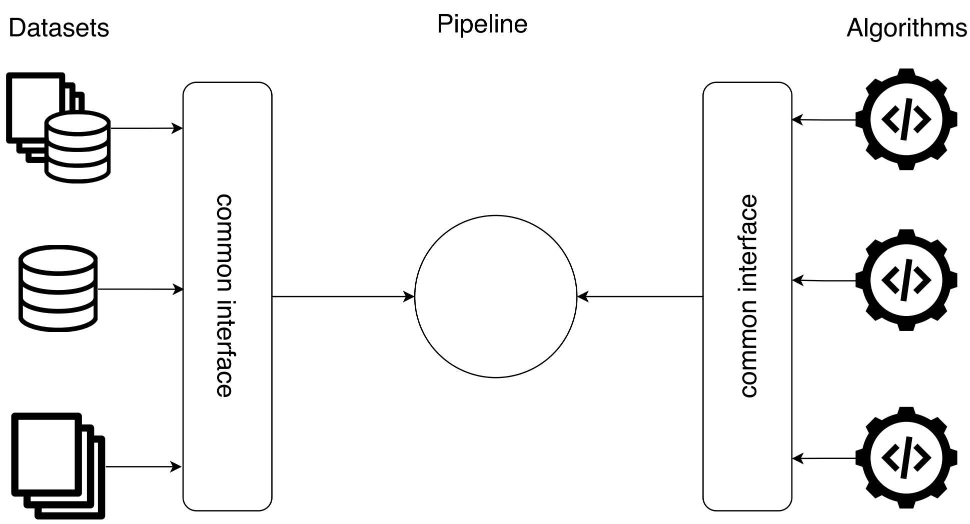

Figure 1: Simple case with a single Pipeline: The Pipeline can interface between all available Datasets and all Algorithms because they share a common interface.

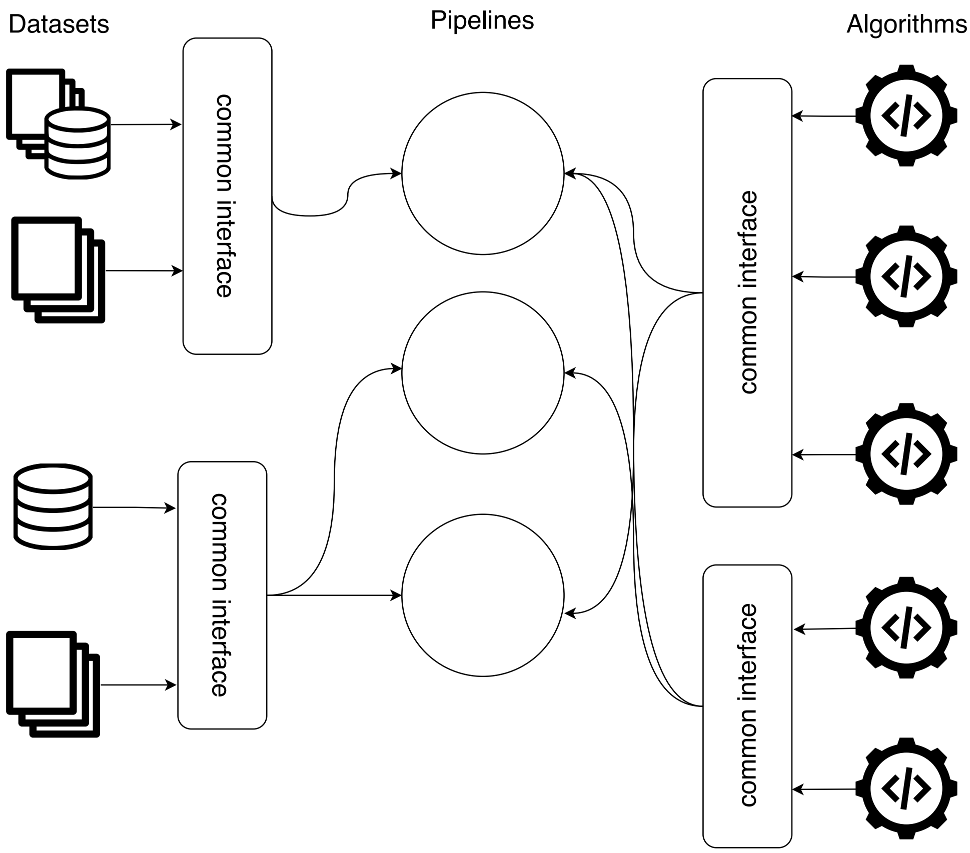

Figure 2: A more complex case: Pipelines act as gluing code for one Dataset interface with one or multiple Algorithm interfaces to perform one specific analysis.

Parameter Optimization and Evaluation Tools

To handle the often complex task of evaluation and Parameter Optimization, `tpcp` provides a re-implementation of the core evaluation (`cross_validate`) and parameter optimization (`GridSearch`, `GridSearchCV`) methods of `scikit-learn` that work with `Pipeline` and `Dataset` objects. Further we provide a generic wrapper for Optuna based optimization algorithms (Akiba et al., 2019) and documentation to implement custom parameter optimizers. This means that independent of the frameworks required for the algorithms, reliable tooling for these critical parts of most data-analysis pipelines can be used.

Availability

The software is available as a pip installable package (`pip install tpcp`) and via GitHub. Documentation can be found via Read the Docs.

Acknowledgments

Most of `tpcp` was created in reaction to problems and issues we ran into during our day-to-day work at the Machine Learning and Data Analytics Lab (MaD Lab) of Friedrich-Alexander-Universität Erlangen-Nürnberg and teaching signal analysis and machine learning to our students. Therefore, we would like to thank all students and MaD Lab members that engaged in our various discussions and brainstorming sessions about evaluation approaches for obscure algorithms and design for the algorithm interface.

Bjoern M. Eskofier gratefully acknowledges the support of the German Research Foundation (DFG) within the framework of the Heisenberg professorship programme (grant number ES 434/8-1). Further, this work was partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1483 – Project-ID 442419336, EmpkinS and by the Mobilise-D project that has received funding from the Innovative Medicines Initiative 2 Joint Undertaking (JU) under grant agreement No. 820820. This JU receives support from the European Union's Horizon 2020 research and innovation program and the European Federation of Pharmaceutical Industries and Associations (EFPIA). Content in this publication reflects the authors' view and neither IMI nor the European Union, EFPIA, or any Associated Partners are responsible for any use that may be made of the information contained herein.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems*. https://www.tensorflow.org/

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). *Optuna: A Next-generation Hyperparameter Optimization Framework* (No. arXiv:1907.10902). arXiv. http://arxiv.org/abs/1907.10902

Barth, J., Oberndorfer, C., Kugler, P., Schuldhaus, D., Winkler, J., Klucken, J., & Eskofier, B. (2013). Subsequence dynamic time warping as a method for robust step segmentation using gyroscope signals of daily life activities. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, 6744–6747. https://doi.org/10.1109/EMBC.2013.6611104

Palotti, J., Mall, R., Aupetit, M., Rueschman, M., Singh, M., Sathyanarayana, A., Taheri, S., & Fernandez-Luque, L. (2019). Benchmark on a large cohort for sleep-wake classification with machine learning techniques. *Npj Digital Medicine*, *2*(1), 50. https://doi.org/10.1038/s41746-019-0126-9

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2018). *Scikit-learn: Machine Learning in Python* (No. arXiv:1201.0490). arXiv. https://doi.org/10.48550/arXiv.1201.0490

Roth, N., Küderle, A., Ullrich, M., Gladow, T., Marxreiter, F., Klucken, J., Eskofier, B. M., & Kluge, F. (2021). Hidden Markov Model based stride segmentation on unsupervised free-living gait data in Parkinson's disease patients. *Journal of NeuroEngineering and Rehabilitation*, *18*(1), 93. https://doi.org/10.1186/s12984-021-00883-7

Schreiber, J. (2017). Pomegranate: Fast and flexible probabilistic modeling in python. *Journal of Machine Learning Research*, *18*(1), 5992–5997.

Zhai, B., Perez-Pozuelo, I., Clifton, E. A. D., Palotti, J., & Guan, Y. (2020). Making Sense of Sleep: Multimodal Sleep Stage Classification in a Large, Diverse Population Using Movement and Cardiac Sensing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *4*(2), 1–33. https://doi.org/10.1145/3397325