The Journal of Open Source Software

mlrl-testbed: A command line utility for tabular machine learning experiments

Michael Rapp [1]

1 Independent Researcher, Germany

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 14 December 2025
Published: unpublished

Summary

The Python package mlrl-testbed provides a command line utility designed to support researchers in conducting reproducible machine learning experiments. It offers a *straightforward*, *easily configurable*, and *extensible* workflow that supports the full experimental lifecycle:

- Loading a dataset.
- Splitting it into training and test sets.
- Training one or more models.
- Evaluating the models' predictive performance.
- Saving experimental results to output files.

By default, mlrl-testbed executes a single experiment using a given dataset and parameter setting. However, it can also be operated in the following modes:

- **Batch mode:** Allows running multiple independent experiments with varying datasets and parameter settings. Installing the optional package mlrl-testbed-slurm enables to run experiments via the *Slurm Workload Manager*[1].
- **Read mode:** Allows inspecting the results of previous experiments and saving them to new output files. When view results obtained in batch mode, results are automatically aggregated across different experiments.
- **Run mode:** Allows re-running previously conducted experiments with the option to partly override their configuration. Experiments for which results are already available can be skipped.

Originally developed to support work on the BOOMER algorithm (Rapp et al., 2020; Rapp, 2021), mlrl-testbed has since evolved into a standalone utility for empirical machine learning studies.

Statement of Need

The rapid growth of machine learning research has led to a variety of tools for evaluating machine learning methods and tracking the results of empirical experiments. Most prominently, this includes commercial platforms like *Google AutoML*[2], *H2O Driverless AI*[3], *neptune.ai*[4], or *Comet.ML*[5]. They typically offer a web-based interface with a rich feature set, including visualization tools, AutoML features and more. While convenient, these tools are proprietary, focus increasingly on large language models rather than tabular machine learning, and may restrict functionality for non-paying users. Some commercial products are available under open

[1] https://slurm.schedmd.com/
[2] https://cloud.google.com/automl
[3] https://h2o.ai/platform/ai-cloud/make/h2o-driverless-ai/
[4] https://neptune.ai/
[5] https://www.comet.com/

source licenses, such as *MLflow* (Zaharia et al., 2018), *Weights and Biases*[6], or *KNIME*[7]. Open source alternatives tend to focus on specific problems of the machine learning toolchain. Desktop applications like *WEKA* (Markov & Russell, 2006) and *Orange* (Demšar et al., 2013) focus on interactive pipeline construction with algorithms included in the respective software. *DataVersionControl* (Barrak et al., 2021) implements a version control system for models and data. *TensorBoard*[8] specializes in visualization. And *PyExperimenter* (Tornede et al., 2023), *Sacred* (Greff et al., 2017), and *Sumatra* (Davison et al., 2018) help with job distribution and keeping track of experimental results.

As a lightweight and cross-platform command line utility, mlrl-testbed aims to fill a niche: It allows to flexibly configure and run experiments in a reproducible manner via a straight-forward, but feature-rich, command line interface. It can be used interactively or in scripts as part of larger workflows. Because it is distributed as a Python package, it can easily be installed on most systems, including headless servers and high-performance computing environments. Rather than implementing any algorithms itself, mlrl-testbed focuses on integrating well-tested algorithms offered by other open source projects into a unified workflow. Out-of-the-box support is provided for algorithms from the *scikit-learn* (Pedregosa et al., 2011) ecosystem. The modular design discussed below allows third parties to add support for additional algorithms or even different machine learning domains.

Command Line Interface

All commands for executing mlrl-testbed follow the following scheme:

```
mlrl-testbed <runnable> [mode] <[control arguments]> [hyperparameters]
```

In contrast to optional arguments (enclosed by [and]), mandatory arguments (surrounded by < and >) must always be specified. These include arguments for specifying a *runnable*. This is a Python source file or module implementing a simple API to integrate an algorithm with mlrl-testbed and possibly extend it with additional functionality. This abstraction allows users to integrate custom methods with little effort, as described in our documentation[9]. For tabular machine learning tasks, no custom code is required: The package mlrl-testbed-sklearn provides a ready-to-use integration with the scikit-learn framework. It can easily be installed via a Python package manager such as *pip*:

```
python -m pip install mlrl-testbed-sklearn
```

We further distinguish between *control arguments* and *hyperparameters*. Arguments belonging to the former category can be mandatory and are used for controlling the behavior of experiments. The arguments for setting an algorithm's hyperparameters depend on the runnable and are always optional, using the algorithm's default if omitted.

Technical Overview

Depending on the runnable and the mode of operation (some steps may be unnecessary in certain modes), mlrl-testbed follows the experimental procedure outlined in Figure 1.

[6] https://github.com/wandb
[7] https://www.knime.com/knime-analytics-platform
[8] https://www.tensorflow.org/tensorboard
[9] https://mlrl-boomer.readthedocs.io/en/stable/user_guide/testbed/

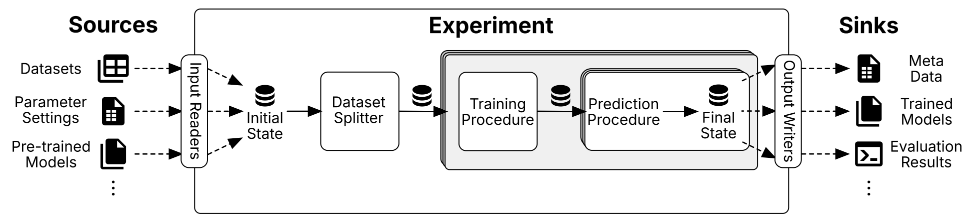

Figure 1: Illustration of the workflow implemented by mlrl-testbed.

Each experiment starts by loading input data from different *sources*. For example, datasets may be read from *LIBSVM* or *ARFF* files, hyperparameter settings may be read from *CSV* files, or previously trained models may be loaded to avoid re-training. After it has finished, an experiment might write output data to so-called *sinks*, e.g., the console log or output files. This may include trained models, the hyperparameters used for training, performance statistics according to common measures, the predictions provided by models, statistics about the dataset, and more. The sources and sinks can be configured in a fine-grained manner via control arguments. Runnables can add new sources and sinks in addition to those supported out-of-the-box.

The workflow followed by an experiment can be viewed as a tree, where each node is associated with a state. The inputs read from different sources make up the initial state at the root node. This state is passed down the tree and may be extended at each node by newly gathered data. For example, before training any machine learning models, the dataset is split into distinct training and test sets following a configurable procedure, e.g., a cross validation, to be able to obtain unbiased performance estimates later on. For each fold, the training and test sets to be used are put into a copy of the state and passed down to a corresponding child node, where the training procedure is invoked. In the same manner, after models have been trained on a training set, they are passed to child nodes, where they can be used to obtain predictions for one or several test sets. These predictions are included in the final state, associated with a leaf of the workflow tree, from which experimental results are extracted. For assessing the quality of different types of predictions commonly used in tabular classification and regression problems, mlrl-testbed automatically picks a suitable selection of the many evaluation measures offered by scikit-learn.

References

Barrak, A., Eghan, E. E., & Adams, B. (2021). On the co-evolution of ML pipelines and source code-empirical study of DVC projects. *IEEE International Conference on Software Analysis, Evolution and Reengineering*, 422–433. https://doi.org/10.1109/saner50967.2021.00046

Davison, A. P., Mattioni, M., Samarkanov, D., & Teleńczuk, B. (2018). Sumatra: A toolkit for reproducible research. In *Implementing reproducible research* (pp. 57–78). Chapman; Hall/CRC. https://doi.org/10.1201/9781315373461-3

Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevar, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., & others. (2013). Orange: Data mining toolbox in Python. *The Journal of Machine Learning Research*, *14*(1), 2349–2353.

Greff, K., Klein, A., Chovanec, M., Hutter, F., & Schmidhuber, J. (2017). The sacred infrastructure for computational research. *SciPy*, *17*, 49–56. https://doi.org/10.25080/shinma-7f4c6e7-008

Markov, Z., & Russell, I. (2006). An introduction to the WEKA data mining system. *ACM SIGCSE Bulletin*, *38*(3), 367–368. https://doi.org/10.1145/1140123.1140127

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rapp, M. (2021). BOOMER – an algorithm for learning gradient boosted multi-label classification rules. *Software Impacts*, *10*, 100137. https://doi.org/10.1016/j.simpa.2021.100137

Rapp, M., Loza Mencía, E., Fürnkranz, J., Nguyen, V.-L., & Hüllermeier, E. (2020). Learning gradient boosted multi-label classification rules. *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, 124–140. https://doi.org/10.1007/978-3-030-67664-3_8

Tornede, T., Tornede, A., Fehring, L., Gehring, L., Graf, H., Hanselle, J., Mohr, F., & Wever, M. (2023). PyExperimenter: Easily distribute experiments and track results. *Journal of Open Source Software*, *8*(84), 5149. https://doi.org/10.21105/joss.05149

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., & others. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Engineering Bulletin*, *41*(4), 39–45.