mldr3pipelines – Flexible Machine Learning Pipelines in R

Martin Binder¹
Florian Pfisterer¹
Michel Lang¹
Lennart Schneider¹
Lars Kotthoff²
Bernd Bischl¹

¹ Department of Statistics, LMU Munich, Germany
² Department of Computer Science, University of Wyoming, USA

Editor: Alexandre Gramfort

Abstract

Recent years have seen a proliferation of ML frameworks. Such systems make ML accessible to non-experts, especially when combined with powerful parameter tuning and AutoML techniques. Modern, applied ML extends beyond direct learning on clean data, however, and needs an expressive language for the construction of complex ML workflows beyond simple pre- and post-processing. We present mldr3pipelines, an R framework which can be used to define linear and complex non-linear ML workflows as directed acyclic graphs. The framework is part of the mldr3 ecosystem, leveraging convenient resampling, benchmarking, and tuning components.

Keywords: machine learning pipelines, preprocessing, automated machine learning

1. Introduction

As one of the most popular and widely-used software systems for statistics and ML, R (R Core Team, 2020) has several packages that provide a standardized interface for predictive modeling, such as caret (Kuhn, 2008), tidymodels (Kuhn and Wickham, 2020b), mldr (Bischl et al., 2016), and its successor mldr3 (Lang et al., 2019). But real-world applications often require complex combinations of ML (pre-) processing steps, which can be expressed as a directed acyclic graph (DAG); we will call such graphs ML pipelines or ML workflows. Specifying such a workflow in an ML system without direct support requires error-prone glue code to combine the individual pieces. One particular difficulty is that (in ML) each pipeline operation is not a stateless function application, but consists of a train and predict stage, where the former not only transforms its inputs into an output, but also learns an internal parameter state, which the latter relies on.¹ mldr3pipelines provides a domain-specific language which allows building ML pipelines from individual processing operations (PipeOps, also see Figure 1). It ships with a large collection of such operations and allows their custom extension through user-defined operations.

¹ This implies that the pipeline idiom in mldr3pipelines is quite different compared to magrittr, dplyr, and tidymodels.

©2021 Martin Binder, Florian Pfisterer, Michel Lang, Lennart Schneider, Lars Kotthoff, and Bernd Bischl.
License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/ Attribution requirements are provided at http://jmlr.org/papers/v22/21-0281.html
2. Related Work

By far, the most widely-used implementation of ML pipelines is Python scikit-learn’s (Pedregosa et al., 2011) `pipeline` module (Buitinck et al., 2013). Unlike our software, scikit-learn supports only linear pipelines directly, although complex pipelines can be expressed through a wrapper mechanism. Several extensions such as baikal (Tineo, 2019) and neuraxle (Chevalier et al., 2019) extend scikit-learn’s pipelining via a graph-based API similar to ours. tidymodels provides the recipes R package (Kuhn and Wickham, 2020a) for building linear preprocessing pipelines with limited flexibility and the workflows package (Vaughan, 2020) for combining these with models into pipelines. The mlr extension mlrCPO (Binder, 2021) also focuses on linear pipelines and has limited support for more complex structures. Industry is increasingly providing systems that support ML pipelines, e.g., Microsoft’s ml.net (Ahmed et al., 2019) for C# and H2O (H2O.ai, 2021) with bindings for Python and R. The d3m software (Milutinovic et al., 2017) was developed as part of DARPA’s Data Driven Discovery of Models program (Shen, 2018) and includes a pipeline system to combine ML primitives, again without the full flexibility of mlr3pipelines. The DAGs in mlr3pipelines go beyond simple combinations of preprocessing and ML models. They support ensemble models and conditional branching that can be represented explicitly as part of the graph structure. Existing pipelining frameworks are often limited to passing training or prediction data objects through the pipeline, while mlr3pipelines allows for passing arbitrary objects. Some operators, for example, pass on functions, which are then used to influence the behavior of operators later in the graph.

3. Design, Functionality, and Examples

mlr3pipelines represents ML workflows as Graph objects: DAGs, whose vertices are PipeOps, which represent arbitrary ML processing operations. The pipeline can either be called to train or predict. Inputs and intermediate objects, most commonly data, move along the DAG’s edges. When they pass through a vertex, they are processed by the cor-
factor_xgboost = po("encode") %>>% lrn("classif.xgboost")

pipe = ppl("branch", list(
  xgboost = factor_xgboost,
  ranger = lrn("classif.ranger")
))

Listing 1 (left) and Figure 2 (right): Example of a branching pipeline. LHS: The %>>% operator builds a linear partial graph. The “ppl branch” template constructs two alternative paths xgboost and ranger, where the latter does not require factor encoding. A new hyperparameter controls the path through which the data will flow. RHS: The pipeline can be plotted with pipe$plot(html = TRUE).

responding PipeOp, and, depending on the call, are either transformed by its train() or predict() method, where the former also creates the operator’s internal state.

This ensures that no information leakage from test data occurs, which is required for the evaluation of predictive systems (Bischl et al., 2012). mlr3pipelines and the mlr3 ecosystem are integrated with each other, so that mlr3’s Learners can be used as PipeOps and Graphs adhere to the same interface as mlr3 learners and can be, for example, resampled and tuned just like any other Learner. This also enables effortless parallelization of these operations for pipelines. mlr3pipelines provides the %>>% operator, which concatenates Graphs (or PipeOps) into larger Graphs. Templates for more complex but frequently used graph patterns are provided through the ppl() lookup function. Outputs from different nodes can be combined in non-trivial ways, for example, joining features created by different preprocessing steps, to create non-linear structures. Other examples include alternative path branching (one of several flows is executed, depending on a hyperparameter), ensembling (predictions from different PipeOps are averaged), and stacking (predictions from different PipeOps are combined in another PipeOp, usually a Learner, to produce a final prediction). Listing 1 shows an example of branching for model and preprocessing selection. Many more examples can be found at https://mlr3gallery.mlr-org.com/#category:mlr3pipelines.

Figure 3 shows examples of complex pipeline components. Some of these are already used in other packages in the mlr3 ecosystem, e.g., mlr3proba (Sonabend et al., 2021) uses the pipeline in Figure 3(i). Users can easily implement their own PipeOps and define their exposed hyperparameters, by inheriting from the PipeOp class to, for example, implement custom feature extraction and processing.

4. Hyperparameter Tuning and AutoML

Each Graph exposes the hyperparameters of its constituent PipeOps for joint tuning via any of the automated tuning methods in mlr3. Simple tuners such as grid and random search, as well as advanced black-box optimizers like Bayesian Optimization (Snoek et al., 2012) and Hyperband (Li et al., 2018) are available through mlr3tuning. Building upon the branching principle of Listing 1, this allows to build entire AutoML systems by combining
multiple learners and preprocessing options. Jointly tuning the selection of these steps with their (subordinate) hyperparameters yields a single pipeline, tailored for a specific task.

5. Availability, Documentation, Code Quality Control

All packages of the mlr3 ecosystem are released under LGPL-3 on GitHub (https://github.com/mlr-org) and on CRAN. Package documentation is available at https://mlr3pipelines.mlr-org.com and in the (work-in-progress) mlr3 book (https://mlr3book.mlr-org.com), with examples in the mlr3 gallery (https://mlr3gallery.mlr-org.com). An extensive suite of unit tests is run on each change via a continuous integration system.

6. Outlook

mlr3pipelines is complete and ready for production use. Our focus for future improvements is better integration of automated ML and deep learning (through mlr3keras and mlr3torch), and leveraging parallel processing specifically for pipelines.

Acknowledgments

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. LK is supported by NSF grant #1813537.
References


Max Kuhn and Hadley Wickham. recipes: Preprocessing tools to create design matrices, 2020a. URL https://CRAN.R-project.org/package=recipes. R package version 0.1.16.


