Journal of Machine Learning Research 17 (2016) 1-5 Submitted 2/15; Revised 8/16; Published 9/16

mlr: Machine Learning in R

Bernd Bischl BERND.BISCHL@STAT.UNI-MUENCHEN.DE
Michel Lang LANG@STATISTIK.TU-DORTMUND.DE
Lars Kotthoff LARSKO@CS.UBC.CA
Julia Schiffner SCHIFFNER@MATH.UNI-DUESSELDORF.DE
Jakob Richter JAKOB.RICHTER@TU-DORTMUND.DE
Erich Studerus ERICH.STUDERUS@UPKBS.CH
Giuseppe Casalicchio GIUSEPPE.CASALICCHIO@STAT.UNI-MUENCHEN.DE
Zachary M. Jones ZMJ@ZMJONES.COM
Department of Statistics
Ludwig-Maximilians-University Munich
Ludwigstrasse 33, 80539 Munich, Germany

Editor: Antti Honkela

Abstract

The MLR package provides a generic, object-oriented, and extensible framework for classification, regression, survival analysis and clustering for the **R** language. It provides a unified interface to more than 160 basic learners and includes meta-algorithms and model selection techniques to improve and extend the functionality of basic learners with, e.g., hyperparameter tuning, feature selection, and ensemble construction. Parallel high-performance computing is natively supported. The package targets practitioners who want to quickly apply machine learning algorithms, as well as researchers who want to implement, benchmark, and compare their new methods in a structured environment.

Keywords: machine learning, hyperparameter tuning, model selection, feature selection, benchmarking, R, visualization, data mining

1. Introduction

R is one of the most popular and widely-used software systems for statistics, data mining, and machine learning. However, it does not define a standardized interface to, e.g., supervised predictive modelling. For any non-trivial experiment one needs to write lengthy, tedious, and error-prone code to unify calling methods and handling output. The MLR package offers a clean, easy-to-use, and flexible domain-specific language for machine learning experiments in **R**. It supports classification, regression, clustering, and survival analysis with more than 160 modelling techniques. Defining learning tasks, training models, making predictions, and evaluating their performance abstracts from the implementation of the underlying learner through an object-oriented interface. Replacing one learning algorithm with another becomes as easy as changing a string. MLR goes far beyond simply providing a unified interface. It implements a generic architecture that allows the assessment of generalization performance, comparison of different algorithms in a scientifically rigorous way, feature selection, and hyperparameter tuning for any method, as well as extending

the functionality of learners through a wrapper mechanism. Queryable properties provide a reflection mechanism for machine learning objects. Finally, mlr provides sophisticated visualization methods that allow to show effects of partial dependence of models. mlr's long term goal is to provide a high-level domain-specific language to express as many aspects of machine learning experiments as possible.

2. Implemented Functionality

mlr uses **R**'s S3 object system and follows a clear structure. Everything is an object and the classes are as reusable and extensible as possible. This permits to extend the package; e.g., connect a new model from a third-party package or write a custom performance measure.

Tasks and Learners. Tasks encapsulate the data and further relevant information like the name of the target variable for supervised learning problems. They are organized hierarchically, with an abstract `Task` at the top and specific subclasses. mlr supports regular, multilabel and cost-sensitive classification, regression, survival analysis, and clustering. The integrated learners specialize to these task types. Currently 82 classification learners, 61 regression learners, 13 survival learners, and 9 cluster learners are integrated. Cost-sensitive classification with observation-dependent costs is supported through a cost-sensitive one-versus-one approach, which delegates to ordinary weighted binary classification.

Evaluation and Resampling. mlr provides 46 different performance measures and implements the resampling methods subsampling (including simple holdout), bootstrapping (OOB, B632, B632+), and cross-validation (normal, leave-one-out, repeated). All resampling strategies may be stratified on both target classes and categorical input features. Observations may be partitioned into inseparable blocks (e.g., when observations come from the same image, sound file, or clinic). Moreover, nested resampling is supported and the resampling strategies used in the outer and inner loops can be combined arbitrarily.

Tuning. In practice, successful modelling often depends on a number of choices like the applied learner, its hyperparameter settings, or the data preprocessing. mlr implements joint optimization of hyperparameters of any learning algorithm and any pre- and postprocessing methods for any task, any resampling strategy, and any performance measure, including categorical and conditional hyperparameters. Random search, grid search, evolutionary algorithms, iterated F-racing, and sequential model-based optimization are available.

Feature Selection. Feature selection can improve the interpretability and performance of a learned predictive model. mlr supports *filter* and *wrapper* approaches, while *embedded* techniques like L_1-penalization are included directly in the learners. Supported selection techniques include information gain, MRMR, and RELIEF, with forward and backward search. Filter scores and sequential wrapper search results can be visualized.

Wrapper Extensions. mlr's wrapper mechanism allows to extend learners through pre-train, post-train, pre-predict, and post-predict hooks. We provide wrappers for missing value imputation, user-defined preprocessing, class imbalance correction, feature selection, tuning, bagging, and stacking. Wrappers can be nested to combine functionalities. Wrapped learners behave like base learners, with added functionality and expanded hyperparameter set. During resampling, all added steps are carried out in each iteration. During tuning,

the joint parameter space can be optimized. For example thresholds for feature filtering can be tuned jointly with other hyperparameters (Lang et al., 2015).

Benchmarking and Parallelization. The `benchmark` function evaluates the performance of multiple learners on multiple tasks. As benchmark studies can quickly become very resource-demanding, mlr natively supports parallelization through the parallelMap package (Bischl and Lang, 2015) that can use local multicore, socket, and MPI computation modes. BatchJobs (Bischl et al., 2015) provides distribution on compute clusters. Operations to be parallelized can be selected explicitly.

Properties and Parameters. Many of the mlr objects have properties that allow them to be used programmatically, e.g., check whether a task has missing values, whether a learner can handle categorical variables, or list all learners suitable for a given task. Every learner includes a description object that defines all hyperparameters, including type, default value, and feasible range. This information is usually not readily available from the implementation of an integrated learning method and may only be listed in its documentation.

3. Example

The following example demonstrates the use of mlr. After loading required packages and the "Sonar" data set (Line 1), we create a classification task and a support vector machine learner (Lines 2–3). The resample description tells mlr to use a 5-fold cross-validation (Line 4). Hyperparameters and box-constraints for tuning are specified in Lines 5–11. We optimize over the choice of a polynomial versus a Gaussian kernel by making their individual parameters dependent on the kernel via the `requires` setting (Lines 9 and 11). We use random search with at most 50 evaluations (Line 12). The values for `C` and `sigma` are sampled on a log-scale through the transformation functions given as the `trafo` argument (Lines 7–8). Line 13 binds everything together and optimizes for mean misclassification error (mmce). `res` holds the best configuration and information on the evaluated parameters.

```
1  library(mlr); library(mlbench); data(Sonar)
2  task = makeClassifTask(data=Sonar, target="Class")
3  lrn = makeLearner("classif.ksvm")
4  rdesc = makeResampleDesc(method="CV", iters=5)
5  ps = makeParamSet(
6    makeDiscreteParam("kernel", values=c("polydot", "rbfdot")),
7    makeNumericParam("C", lower=-15, upper=15, trafo=function(x) 2^x),
8    makeNumericParam("sigma", lower=-15, upper=15, trafo=function(x) 2^x,
9     requires = quote(kernel == "rbfdot")),
10   makeIntegerParam("degree", lower = 1, upper = 5,
11    requires = quote(kernel == "polydot")))
12 ctrl = makeTuneControlRandom(maxit=50)
13 res = tuneParams(lrn, task, rdesc, par.set=ps, control=ctrl, measures=mmce)
```

4. Availability, Documentation, Maintenance, and Code Quality Control

The mlr source code is available under the BSD 2-clause license and hosted on GitHub (`https://github.com/mlr-org/mlr`). Stable releases are frequently published on the Contributed R Archive Network (CRAN), which lists mlr in Task View 'Machine Learning & Statistical Learning'. We provide extensive API documentation through **R**'s internal help

system and a very detailed tutorial (Schiffner et al., 2016) that guides the user from very basic tasks to complex applications with worked examples and is continuously extended. An issue tracker, the test framework testthat (with more than 10,000 lines of tests and more than 1,200 assertions), and the CI systems Travis and Jenkins support the correctness of the code base. In addition, we provide documentation and coding guidelines for developers and contributors.

5. Comparison to Similar Toolkits/Frameworks

Several other **R** packages provide frameworks for handling prediction models, including caret (Kuhn, 2008), DMwR (Torgo, 2010), CORElearn (Robnik-Sikonja and with contributions from John Adeyanju Alao, 2016), rattle (Williams, 2011), rminer (Cortez, 2010), CMA (Slawski et al., 2008), and ipred (Peters and Hothorn, 2015). The first 5 only support classification and regression, CMA only classification. mlr's generic wrapper mechanism is not provided by any other package in this form. Although caret and CMA can fuse a learner with a preprocessing or variable selection method, only mlr can seamlessly tune these methods simultaneously (Koch et al., 2012). Only mlr, rminer, and CMA support nested cross-validation. A similar degree of flexibility can be achieved in caret, but requires custom implementations. Only mlr supports ensemble learning through stacking natively, mlr and caret support bagging natively. Bagging is also available in ipred and caretEnsemble provides stacking for caret. Only mlr and caret have native support for parallel computations. Similar toolkits exist for other languages, e.g., Weka for Java (Hall et al., 2009) and scikit-learn for Python (Pedregosa et al., 2011).

6. Conclusions and Outlook

We presented the mlr package, which provides a unified interface to machine learning in **R**. It implements a generic architecture for a range of common machine learning tasks. mlr is alive and under active development. It has a growing user community and is used for teaching and research.

Major directions for future extensions include better support for large-scale data, a closer connection to the OpenML project (Vanschoren et al., 2013) for open machine learning experiments,[1] and better integration of sequential model-based optimization.[2]

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft [SCHW 1508/3-1 to J.S.] and Collaborative Research Center SFB 876, project A3.

1. An OpenML-R connector package is available at `https://github.com/openml/r`.
2. mlr supports an experimental integration via mlrMBO (`https://github.com/mlr-org/mlrMBO`).

References

B. Bischl and M. Lang. *parallelMap: Unified interface to some popular parallelization backends for interactive usage and package development*, 2015. URL `https://github.com/berndbischl/parallelMap`. R package version 1.3.

B. Bischl, M. Lang, O. Mersmann, J. Rahnenführer, and C. Weihs. BatchJobs and BatchExperiments: Abstraction mechanisms for using R in batch environments. *Journal of Statistical Software*, 64(11), 2015.

P. Cortez. Data Mining with Neural Networks and Support Vector Machines using the R/rminer Tool. In P. Perner, editor, *Advances in Data Mining. Applications and Theoretical Aspects*, volume 6171 of *LNCS*, pages 572–583, Berlin, Germany, 2010. Springer.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

P. Koch, B. Bischl, O. Flasch, T. Bartz-Beielstein, C. Weihs, and W. Konen. Tuning and evolution of support vector kernels. *Evolutionary Intelligence*, 5(3):153–170, 2012.

M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008.

M. Lang, H. Kotthaus, P. Marwedel, C. Weihs, J. Rahnenführer, and B. Bischl. Automatic model selection for high-dimensional survival analysis. *Journal of Statistical Computation and Simulation*, 85(1):62–76, 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

A. Peters and T. Hothorn. *ipred: Improved Predictors*, 2015. URL `http://CRAN.R-project.org/package=ipred`. R package version 0.9-5.

M. Robnik-Sikonja and P. S. with contributions from John Adeyanju Alao. *CORElearn: Classification, Regression and Feature Evaluation*, 2016. URL `https://CRAN.R-project.org/package=CORElearn`. R package version 1.48.0.

J. Schiffner, B. Bischl, M. Lang, J. Richter, Z. M. Jones, P. Probst, F. Pfisterer, M. Gallo, D. Kirchhoff, T. Kühn, J. Thomas, and L. Kotthoff. mlr tutorial, 2016.

M. Slawski, M. Daumer, and A.-L. Boulesteix. CMA – a comprehensive Bioconductor package for supervised classification with high dimensional data. *BMC Bioinformatics*, 9 (1):439, 2008.

L. Torgo. *Data Mining with R: Learning with Case Studies*. Data Mining and Knowledge Discovery Series. Chapman and Hall/CRC, Boca Raton, FL, 2010.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

G. J. Williams. *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*. Use R! Springer, New York, NY, 2011.