Flexible and Robust Machine Learning Using mlr3 in R

Table of contents

List of Figures

List of Tables

Getting Started

Editors

Michel Lang, Raphael Sonabend, Lars Kotthoff, Bernd Bischl

Contributing authors

- Marc Becker
- Przemysław Biecek
- Martin Binder
- Bernd Bischl
- Lukas Burk
- Giuseppe Casalicchio
- Sebastian Fischer
- Natalie Foss
- Lars Kotthoff
- Michel Lang
- Florian Pfisterer
- Damir Pulatov
- Lennart Schneider
- Patrick Schratz
- Raphael Sonabend
- Marvin Wright

Welcome to the Machine Learning in R universe. This is the electronic version of the upcoming book *Flexible and Robust Machine Learning Using mlr3 in R*. This book will teach you about the mlr3 universe of packages, from some machine learning methodology to implementations of complex algorithmic pipelines. We will cover how to use the mlr3 family of packages for data processing, fitting and training of machine learning models, tuning and hyperparameter optimization, feature selection, pipelines, data preprocessing, and model interpretability. In addition we will look at how our interface works beyond classification and regression settings to other fields including survival analysis, clustering, and more. Finally we will demonstrate how you can contribute to our universe by creating packages, learners, measures, pipelines, and other features.

We hope you enjoy reading our book and always welcome comments and feedback. If you notice any mistakes in the book we would appreciate if you could open an issue in the mlr3book issue tracker. All content in this book is licenced under CC BY-NC 4.0.

Preface

Welcome to the Machine Learning in R universe (mlr3verse)! Before we begin, make sure you have installed `mlr3` if you want to follow along. We recommend installing the complete `mlr3verse`, which will install all of the important packages.

```r
install.packages("mlr3verse")
```

Or you can install just the base package:

```r
install.packages("mlr3")
```

In our first example, we will show you some of the most basic functionality – training a model and making predictions.

```r
library(mlr3)
task = tsk("penguins")
split = partition(task)
learner = lrn("classif.rpart")

learner$train(task, row_ids = split$train)
learner$model
```

```
n= 231

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 231 129 Adelie (0.441558442 0.199134199 0.359307359)
  2) flipper_length< 207.5 145  44 Adelie (0.696551724 0.296551724 0.006896552)
    4) bill_length< 44.65 100   2 Adelie (0.980000000 0.020000000 0.000000000) *
    5) bill_length>=44.65 45   4 Chinstrap (0.066666667 0.911111111 0.022222222) *
  3) flipper_length>=207.5 86   4 Gentoo (0.011627907 0.034883721 0.953488372) *
```

```r
predictions = learner$predict(task, row_ids = split$test)
predictions
```

```
<PredictionClassif> for 113 observations:
    row_ids     truth  response
          3    Adelie    Adelie
          4    Adelie    Adelie
          5    Adelie    Adelie
---
        341 Chinstrap    Adelie
        343 Chinstrap    Gentoo
        344 Chinstrap Chinstrap
```

```r
predictions$score(msr("classif.acc"))
```

```
classif.acc
  0.9380531
```

In this example, we trained a decision tree on a subset of the `penguins` dataset, made predictions on the rest of the data and then evaluated these with the accuracy measure. In Chapter **??** we will break this down in more detail.

`mlr3` makes training and predicting easy, but it also allows us to perform very complex operations in just a few lines of code:

```r
library(mlr3verse)
library(mlr3pipelines)
library(mlr3benchmark)

tasks = tsks(c("breast_cancer", "sonar"))
tuned_rf = auto_tuner(
    tnr("grid_search", resolution = 5),
    lrn("classif.ranger", num.trees = to_tune(200, 500)),
    rsmp("holdout")
)
tuned_rf = pipeline_robustify(NULL, tuned_rf, TRUE) %>>%
    po("learner", tuned_rf)
stack_lrn = ppl(
    "stacking",
    base_learners = lrns(c("classif.rpart", "classif.lda")),
    lrn("classif.log_reg"))
stack_lrn = pipeline_robustify(NULL, stack_lrn, TRUE) %>>%
    po("learner", stack_lrn)

learners = c(tuned_rf, stack_lrn)
bm = benchmark(benchmark_grid(tasks, learners, rsmp("holdout")))
```

```r
bma = bm$aggregate(msr("classif.acc"))[, c("task_id", "learner_id",
  "classif.acc")]
bma$learner_id = rep(c("RF", "Stack"), 2)
bma
```

```
         task_id learner_id classif.acc
1: breast_cancer         RF   0.9605263
2: breast_cancer      Stack   0.9254386
3:         sonar         RF   0.7681159
4:         sonar      Stack   0.6376812
```

```r
as.BenchmarkAggr(bm)$friedman_test()
```

```

	Friedman rank sum test

data:  ce and learner_id and task_id
Friedman chi-squared = 2, df = 1, p-value = 0.1573
```

In this (much more complex!) example we chose two tasks and two machine learning (ML) algorithms ("learners" in `mlr3` terms). We used automated tuning to optimize the number of trees in the random forest learner (Chapter **??**) and a ML pipeline that imputes missing data, collapses factor levels, and creates stacked models (Chapter **??**). We also showed basic features like loading learners (Chapter **??**) and choosing resampling strategies for benchmarking (Chapter **??**). Finally, we compared the performance of the models using the mean accuracy on the test set, and applied a statistical test to see if the learners performed significantly different (they did not!).

You will learn how to do all this and more in this book. We will walk through the functionality offered by `mlr3` and the packages in the `mlr3verse` step by step. There are a few different ways you can use this book, which we will discuss next.

How to use this book

The mlr3 ecosystem is the result of many years of methodological and applied research and improving the design and implementation of the packages over the years. This book describes the resulting features of the `mlr3verse` and discusses best practices for ML, technical implementation details, extension guidelines, and in-depth considerations for optimizing ML. It is suitable for a wide range of readers and levels of ML expertise.

Chapter **??**, Chapter **??**, and Chapter **??** cover the basics of mlr3. These chapters are essential to understanding the core infrastrucure of ML in mlr3. We recommend that all readers study these chapters to become familiar with basic mlr3 terminology, syntax, and style. Chapter **??**, Chapter **??**, and Chapter **??** contain more advanced implementation details and some ML theory. Chapter **??** delves into detail on domain-specific methods that are implemented in our extension packages. Readers may choose to selectively read sections in this chapter depending on your use cases (i.e., if you have domain-specific problems to tackle), or to use these as introductions to new domains to explore. Chapter **??** contains technical implementation details that are essential reading for advanced users who require parallelisation, custom error handling, and fine control over hyperparameters and large databases. Chapter **??** discusses packages that can be integrated with mlr3 to provide model-agnostic interpretability methods. Finally, anyone who would like to contribute to our ecosystem should read Chapter **??**.

Of course, you can also read the book cover to cover from start to finish. We have marked any section that contains complex technical information with an exclamation mark (!). You may wish to skip these sections if you are only interested in basic functionality. Similarly, we have marked sections that are optional, such as parts that are more methodological focused and do not discuss the software implementation, with an asterisk (*). Readers that are interested in the more technical detail will likely want to pay attention to the tables at the end of each chapter that show the relationship between our S3 'sugar' functions and the underlying R6 classes; this is explained in more detail in Chapter **??**.

This book tries to follow the Diátaxis framework[1] for documentation and so we include tutorials, how-to guides, API references, and explanations. This means that the conclusion of each chapter includes a short reference to the core functions learnt in the chapter, links to relevant posts in the mlr3gallery[2], and a few exercises that will cover content introduced in the chapter. You can find the solutions to these exercises in **?@sec-solutions**.

Finally, if you want to reproduce any of the results in this book, note that the random seed is set as the chapter number and the `sessionInfo()` printed in Appendix **??**.

Installation guidelines

All packages in the mlr3 ecosystem can be installed from GitHub and R-universe; the majority (but not all) packages can also be installed from CRAN. We recommend adding the mlr-org R-universe[3] to your R options so that you can install all packages with `install.packages()` without having to worry which package repository it comes from. To do this, install the `usethis` package and run the following:

```r
usethis::edit_r_profile()
```

In the file that opens add or change the `repos` argument in `options` so it looks something like this (you might need to add the full code block below or just edit the existing `options` function).

```r
options(repos = c(
  mlrorg = "https://mlr-org.r-universe.dev",
  CRAN = "https://cloud.r-project.org/"
))
```

Save the file, restart your R session, and you are ready to go!

```r
install.packages("mlr3verse")
```

If you want latest development versions of any of our packages, run

```r
remotes::install_github("mlr-org/{pkg}")
```

[1]https://diataxis.fr/
[2]https://mlr-org.com/gallery.html
[3]R-universe is an alternative package repository to CRAN. The bit of code below tells R to look at both R-universe and CRAN when trying to install packages. R will always install the latest version of a package.

with `{pkg}` replaced with the name of the package you want to install. You can see an up-to-date list of all our extension packages at https://github.com/mlr-org/mlr3/wiki/Extension-Packages.

Community links

The mlr community is open to all and we welcome everybody, from those completely new to ML and R to advanced coders and professional data scientists. You can reach us on our Mattermost[4].

For case studies and how-to guides, check out the mlr3gallery[5] for extended practical blog posts. For updates on mlr you might find our blog[6] a useful point of reference.

We appreciate all contributions, whether they are bug reports, feature requests, or pull requests that fix bugs or extend functionality. Each of our GitHub repositories includes issues and pull request templates to ensure we can help you as much as possible to get started. Please make sure you read our code of conduct[7] and contribution guidelines[8]. With so many packages in our universe it may be hard to keep track of where to open issues. As a general rule:

1. If you have a question about using any part of the mlr3 ecosystem, ask on StackOverflow and use the tag #mlr3 – one of our team will answer you there. Be sure to include a reproducible example (reprex) and if we think you found a bug then we will refer you to the relevant GitHub repository.
2. Bug reports or pull requests about core functionality (train, predict, etc.) should be opened in the mlr3 GitHub repository.
3. Bug reports or pull requests about learners should be opened in the mlr3extralearners GitHub repository.
4. Bug reports or pull requests about measures should be opened in the mlr3measures GitHub repository.
5. Bug reports or pull requests about domain specific functionality should be opened in the GitHub repository of the respective package (see Chapter **??**).

Do not worry about opening an issue in the wrong place, we will transfer it to the right one!

Citation info

Every package in the mlr3verse has its own citation details that can be found on the respective GitHub repository.

To reference this book please use:

```
Becker M, Binder M, Bischl B, Foss N, Kotthoff L, Lang M, Pfisterer F,
Reich N G, Richter J, Schratz P, Sonabend R, Pulatov D.
2023. "Flexible and Robust Machine Learning Using mlr3 in R". https://mlr3book.mlr-org.com.
```

[4] https://lmmisld-lmu-stats-slds.srv.mwn.de/signup_email?id=6n7n67tdh7d4bnfxydqomjqspo
[5] https://mlr-org.com/gallery.html
[6] https://mlr-org.com/blog.html
[7] https://github.com/mlr-org/mlr3/blob/main/.github/CODE_OF_CONDUCT.md
[8] https://github.com/mlr-org/mlr3/blob/main/CONTRIBUTING.md

```
@misc{
    title = Flexible and Robust Machine Learning Using mlr3 in R
    author = {Marc Becker, Martin Binder, Bernd Bischl, Natalie Foss,
    Lars Kotthoff, Michel Lang, Florian Pfisterer, Nicholas G. Reich,
    Jakob Richter, Patrick Schratz, Raphael Sonabend, Damir Pulatov},
    url = {https://mlr3book.mlr-org.com},
    year = {2023}
}
```

To reference the `mlr3` package, please cite our JOSS paper:

```
Lang M, Binder M, Richter J, Schratz P, Pfisterer F, Coors S, Au Q,
Casalicchio G, Kotthoff L, Bischl B (2019). "mlr3: A modern object-oriented
machine learning framework in R." Journal of Open Source Software.
doi: 10.21105/joss.01903.

@Article{mlr3,
  title = {{mlr3}: A modern object-oriented machine learning framework in {R}},
  author = {Michel Lang and Martin Binder and Jakob Richter and Patrick Schratz and
  Florian Pfisterer and Stefan Coors and Quay Au and Giuseppe Casalicchio and
  Lars Kotthoff and Bernd Bischl},
  journal = {Journal of Open Source Software},
  year = {2019},
  month = {dec},
  doi = {10.21105/joss.01903},
  url = {https://joss.theoj.org/papers/10.21105/joss.01903},
}
```

mlr3book style guide

Throughout this book we will use our own style guide that can be found in the mlr3 wiki[9]. Below are the most important style choices relevant to the book.

1. We always use `=` instead of `<-` for assignment.
2. Class names are in `UpperCamelCase`
3. Function and method names are in `lower_snake_case`
4. When referencing functions, we will only include the package prefix (e.g., `pkg::function`) for functions outside the mlr3 universe or when there may be ambiguity about in which package the function lives. Note you can use `environment(function)` to see which namespace a function is loaded from.
5. We denote packages, fields, methods, and functions as follows:
    - `package` - With link (if online) to package CRAN, R-Universe, or GitHub page

[9]https://github.com/mlr-org/mlr3/wiki/Style-Guide

- `package::function()` (for functions *outside* the mlr-org ecosystem)
- `function()` (for functions *inside* the mlr-org ecosystem) - With link to function documentation page
- `$field` for fields (data encapsulated in a R6 class)
- `$method()` for methods (functions encapsulated in a R6 class)

1 Introduction and Overview

The (**M**achine **L**earning in **R**) `mlr3` (Lang et al. 2019) package and ecosystem provide a generic, object-oriented, and extensible framework for regression (Appendix **??**), classification (Section **??**), and other machine learning tasks (Chapter **??**) for the R language (R Core Team 2019). This unified interface provides functionality to extend and combine existing machine learning algorithms (learners (Section **??**)), intelligently select and tune the most appropriate technique for a given machine learning task (Appendix **??**), and perform large-scale comparisons that enable meta-learning. Examples of this advanced functionality include hyperparameter tuning (Chapter **??**) and feature selection (Chapter **??**). Parallelization of many operations is natively supported (Section **??**).

`mlr3` has similar overall aims to `caret`, `tidymodels`, scikit-learn[1] for Python, and MLJ[2] for Julia. In general `mlr3` is designed to provide more flexibility than other machine learning frameworks while still offering easy ways to use advanced functionality. While in particular tidymodels makes it very easy to perform simple machine learning tasks, `mlr3` is more geared towards advanced machine learning. To get a quick overview of how to do things in the `mlr3verse`, see the mlr3 cheatsheets[3].

Note

`mlr3` provides a unified interface to existing learners in R. With few exceptions, we do not implement any learners ourselves, although we often augment the functionality provided by the underlying learners. This includes, in particular, the definition of hyperparameter spaces for tuning.

1.1 Target audience

We assume that users of `mlr3` have taken an introductory machine learning course or have the equivalent expertise and some basic experience with R. A background in computer science or statistics is beneficial for understanding the advanced functionality described in the later chapters of this book, but not required. (James et al. 2014) provides a comprehensive introduction for those new to machine learning.

`mlr3` provides a domain-specific language for machine learning in R that allows to do everything from simple exercises to complex projects. We target both **practitioners** who want to quickly apply machine learning algorithms and **researchers** who want to implement, benchmark, and compare their new methods in a structured environment.

[1] https://scikit-learn.org/
[2] https://alan-turing-institute.github.io/MLJ.jl/dev/
[3] https://cheatsheets.mlr-org.com/

1.2 From mlr to mlr3

The `mlr` package (Bischl et al. 2016) was first released to CRAN[4] in 2013, with the core design and architecture dating back much further. Over time, the addition of many features has led to a considerably more complex design that made it harder to build, maintain, and extend than we had hoped for. In hindsight, we saw that some design and architecture choices in `mlr` made it difficult to support new features, in particular with respect to machine learning pipelines. Furthermore, the R ecosystem and helpful packages such as `data.table` have undergone major changes after the initial design of `mlr`.

It would have been impossible to integrate all of these changes into the original design of `mlr`. Instead, we decided to start working on a reimplementation in 2018, which resulted in the first release of `mlr3` on CRAN in July 2019.

The new design and the integration of further and newly-developed R packages (especially `R6`, `future`, and `data.table`) makes `mlr3` much easier to use, maintain, and in many regards more efficient than its predecessor `mlr`. The packages in the ecosystem are less tightly coupled, making them easier to maintain and easier to develop, especially very specialized packages.

1.3 Design principles

Some readers may want to skip this section of the book.

We follow these general design principles in the `mlr3` package and `mlr3verse` ecosystem.

- **Separation of computation and presentation**. Most packages of the `mlr3` ecosystem focus on processing and transforming data, applying machine learning algorithms, and computing results. Our core packages do not provide graphical user interfaces (GUIs) because their dependencies would make installation unnecessarily complex, especially on headless servers. For the same reason, visualizations of data and results are provided in the extra package `mlr3viz`, which avoids dependencies on `ggplot2`.
- **Object-oriented programming (OOP)**. We embrace `R6` for a clean, object-oriented design, object state-changes, and reference semantics. This means that the state of common objects (e.g. tasks (Appendix **??**) and learners (Section **??**)) is encapsulated within the object, for example to keep track of whether a model has been trained without the user having to worry about this. We also use inheritance to specialize objects, e.g. all learners are derived from a common base class that provides basic functionality.
- **Tabular data**. Embrace `data.table` for its top-notch computation performance as well as tabular data as a data structure which can be easily processed further.
- **Unify input and output data formats.** This considerably simplifies the API and allows easy selection and "split-apply-combine" (aggregation) operations. We combine `data.table` and `R6` to place references to non-atomic and compound objects in tables and make heavy use of list columns.

[4]https://cran.r-project.org

- **Defensive programming and type safety**. All user input is checked with `checkmate` (Lang 2017). We document return types, and avoid mechanisms popular in base R which "simplify" the result unpredictably (e.g., `sapply()` or the `drop` argument for indexing data.frames). And we have extensive unit tests!
- **Light on dependencies**. One of the main maintenance burdens for `mlr` was to keep up with changing learner interfaces and behavior of the many packages it depended on. We require far fewer packages in `mlr3` to make installation and maintenance easier. We still provide the same functionality, but it is split into more packages that have fewer dependencies individually. As mentioned above, this is particularly the case for all visualization functionality, which is contained in a separate package to avoid unnecessary dependencies in all other packages.

1.4 Package ecosystem

`mlr3` depends on the following popular and well-established packages that are not developed by core members of the `mlr3` team:

- `R6`: The class system predominantly used in mlr3.
- `data.table`: High-performance extension of R's `data.frame`.
- `digest`: Cryptographic hash functions.
- `uuid`: Generation of universally unique identifiers.
- `lgr`: Highly configurable logging library.
- `mlbench` and `palmerpenguins`: More machine learning data sets.
- `evaluate`: For capturing output, warnings, and exceptions (Section **??**).
- `future` / `future.apply`: For parallelization (Section **??**).

The `mlr3` package itself provides the base functionality that the rest of ecosystem (`mlr3verse`) relies on and the fundamental building blocks for machine learning. **?@fig-mlr3verse** shows the packages in the `mlr3verse` that extend `mlr3` with capabilities for preprocessing, pipelining, visualizations, additional learners, additional task types, and more.

Tip

A complete list with links to the repositories for the respective packages can be found on our package overview page[5].

We build on `R6` for object orientation and `data.table` to store and operate on tabular data. Both are core to `mlr3`; we briefly introduce both packages for beginners. While in-depth expertise with these packages is not necessary, a basic understanding is required to work effectively with `mlr3`.

1.5 Quick R6 introduction for beginners

`R6` is one of R's more recent paradigm for object-oriented programming (OOP). It addresses shortcomings of earlier OO implementations in R, such as S3, which we used in `mlr`. If you have done

[5]https://mlr-org.com/ecosystem.html

any object-oriented programming before, R6 should feel familiar. We focus on the parts of R6 that you need to know to use `mlr3`.

Objects are created by calling the constructor of an `R6::R6Class()` object, specifically the initialization method `$new()`. For example, `foo = Foo$new(bar = 1)` creates a new object of class `Foo`, setting the `bar` argument of the constructor to the value `1`.

Objects have mutable state that is encapsulated in their fields, which can be accessed through the dollar operator. We can access the `bar` value in the `foo` variable from above through `foo$bar` and set its value by assigning the field, e.g. `foo$bar = 2`.

In addition to fields, objects expose methods that allow to inspect the object's state, retrieve information, or perform an action that changes the internal state of the object. For example, the `$train()` method of a learner changes the internal state of the learner by building and storing a model, which can then be used to make predictions.

Objects can have public and private fields and methods. The public fields and methods define the API to interact with the object. Private methods are only relevant for you if you want to extend `mlr3`, e.g. with new learners.

Technically, R6 objects are environments, and as such have reference semantics. For example, `foo2 = foo` does not create a copy of `foo` in `foo2`, but another reference to the same actual object. Setting `foo$bar = 3` will also change `foo2$bar` to `3` and vice versa.

To copy an object, use the `$clone()` method and the `deep = TRUE` argument for nested objects, for example, `foo2 = foo$clone(deep = TRUE)`.

Tip

For more details on R6, have a look at the excellent R6 vignettes[6], especially the introduction[7]. For comprehensive R6 information, we refer to the R6 chapter from Advanced R[8].

1.6 Quick `data.table` introduction for beginners

The package `data.table` implements a popular alternative to R's `data.frame()`, i.e. an object to store tabular data. We decided to use `data.table` because it is blazingly fast and scales well to bigger data.

Note

Many `mlr3` functions return `data.tables` which can conveniently be subsetted or combined with other outputs. If you do not like the syntax or are feeling more comfortable with other tools, base `data.frames` or `tibble`/`dplyrs` are just a single `as.data.frame()` or `as_tibble()` away.

[8]https://r6.r-lib.org/
[8]https://r6.r-lib.org/articles/Introduction.html
[8]https://adv-r.hadley.nz/r6.html

Data tables are constructed with the `data.table()` function (whose interface is similar to `data.frame()`) or by converting an object with `as.data.table()`.

```r
library("data.table")
dt = data.table(x = 1:6, y = rep(letters[1:3], each = 2))
dt
```

```
   x y
1: 1 a
2: 2 a
3: 3 b
4: 4 b
5: 5 c
6: 6 c
```

`data.table`s can be used much like `data.frame`s, but they do provide additional functionality that makes complex operations easier. For example, data can be summarized by groups with the `[` operator:

```r
dt[, mean(x), by = "y"]
```

```
   y  V1
1: a 1.5
2: b 3.5
3: c 5.5
```

There is also extensive support for many kinds of database join operations (see e.g. this RPubs post by Ronald Stalder[9]) that make it easy to combine multiple `data.table`s in different ways.

Tip

For an in-depth introduction, we refer the reader to the excellent data.table introduction vignette[10].

1.7 Essential mlr3 utilities

Sugar functions

Most objects in mlr3 can be created through convenience functions called *sugar functions*. They provide shortcuts for common code idioms, reducing the amount of code a user has to write. We heavily use sugar functions throughout this book and give the equivalent "full form" for full detail. In most cases, the sugar functions will achieve what you want to do, and you only have

[9]https://rstudio-pubs-static.s3.amazonaws.com/52230_5ae0d25125b544caab32f75f0360e775.html
[10]https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html

to consider using the full R6 code if you program custom objects or extensions. For example `lrn("regr.rpart")` is the sugar version of `LearnerRegrRpart$new()`.

Dictionaries

`mlr3` uses dictionaries to store objects like learners or tasks. These are key-value stores that allow to associate a key with a value that can be an R6 object, much like paper dictionaries associate words with their definitions. Often, values in dictionaries are accessed through sugar functions that automatically use the applicable dictionary without the user having to specify it; only the key to be retrieved needs to be specified. Dictionaries are used to group relevant objects so that they can be listed and retrieved easily. For example, a learner can be retrieved directly from the `mlr_learners` dictionary using the key `"classif.featureless"` (`mlr_learners$get("classif.featureless")`) and you can get an overview over all stored learners with `as.data.table(mlr_learners)`.

mlr3viz

`mlr3viz` is the package for all plotting functionality in the `mlr3` ecosystem. The package uses a common theme (`ggplot2::theme_minimal()`) so that all generated plots have a similar aesthetic. Under the hood, `mlr3viz` uses `ggplot2`. `mlr3viz` extends `fortify` and `autoplot`for use with common `mlr3` outputs including Prediction, Learner, and Benchmark objects (these objects will be introduced and covered in the next chapter). The most common use of `mlr3viz` is the `autoplot()` function, where the type of the object passed determines the type of the plot. Plot types are documented in the respective manual page that can be accessed through `?autoplot.X`. For example, the documentation of plots for regression tasks can be found by running `?autoplot.TaskRegr`.

2 Fundamentals

We describe and explain the basic building blocks of mlr3 and how to train and evaluate simple machine learning models. The chapter introduces the different types of tasks that mlr3 supports and how to work with them, learners and how to train models, how to make predictions using those trained models, and how to evaluate the quality of the predictions in a principled fashion. Only the basic concepts are introduced, but we give pointers on where to learn more in the rest of the book, and overviews of other concepts. After reading this chapter, you will be able to use mlr3 for most machine learning workflows.

In this chapter, we will introduce the `mlr3` objects and corresponding `R6` classes that implement the essential building blocks of machine learning. These building blocks include the data (and the methods of creating training and test sets), the machine learning algorithm (and its training and prediction process), and evaluation measures to assess the quality of predictions.

In essence, machine learning means learning relationships from data. In supervised learning, datasets consist of observations (rows in tabular data) that are labeled, which means that each data point includes features (columns in tabular data) and a quantity that we are trying to predict, also called 'target'. For example, we might want to predict the miles per gallon a car gets based on features such as its horsepower and the number of gears. Data and information on what they represent, along with what quantities to predict are called "tasks" in `mlr3` (Appendix **??**) – they can be thought of as machine learning tasks we are trying to solve. There can be more than one task per dataset, for example ones that include different sets of features, observations, or predict different target quantities.

Supervised learning can be further divided into regression (predicting numeric target values) and classification (predicting categorical target values/labels). In either case, the goal is to build a model that captures the relationship between features and target. We can build such models using machine learning algorithms, for example decision trees, support vector machines, neural networks, and many more. A machine learning algorithm, given training data, induces such a model. Machine learning algorithms are called "learners" in `mlr3` (Section **??**) – given data, they learn models. Each learner has a parameterized space that potential models are drawn from and during the training process, these parameters are fitted to best match the data. For example, the parameters could be the weights given to individual features when predicting a quantity in linear regression. For other learners, the parameters are not as explicit, for example for decision tree learners where a fitted model corresponds to a particular decision tree. All learners optimize a so-called loss function during training, i.e. training a learner means finding the model that optimizes the loss. In general, a loss function quantifies the mismatch between ground truth target values in the training data and the predictions of the model.

Given a model, we can make predictions (Section **??**) on new data. A model is only useful though if it generalizes beyond the training data. Otherwise, we could build a perfect model by simply memorizing the training data. Therefore, separate test data is used to evaluate models in an unbiased way and to assess to what extent they have learned the true relationships that underlie the data (Chapter **??**). We can evaluate models in `mlr3` in many ways (Section **??**). In general, we use the same kind of loss function that the learner used to build the model, but now with data that was not used during training. The performance of a model, quantified by the value of the loss function when evaluated on new data, is called the estimate of the generalization error – how well do we expect this model to do in general? `mlr3` calls loss functions "measures". We can use different measures for training and testing, although it makes most sense for the measures to be the same.

Much more information on (supervised) machine learning can be found in Hastie, Friedman, and Tibshirani (2001), James et al. (2014), or Bishop (2006).

The basic idea is illustrated in the following figure:

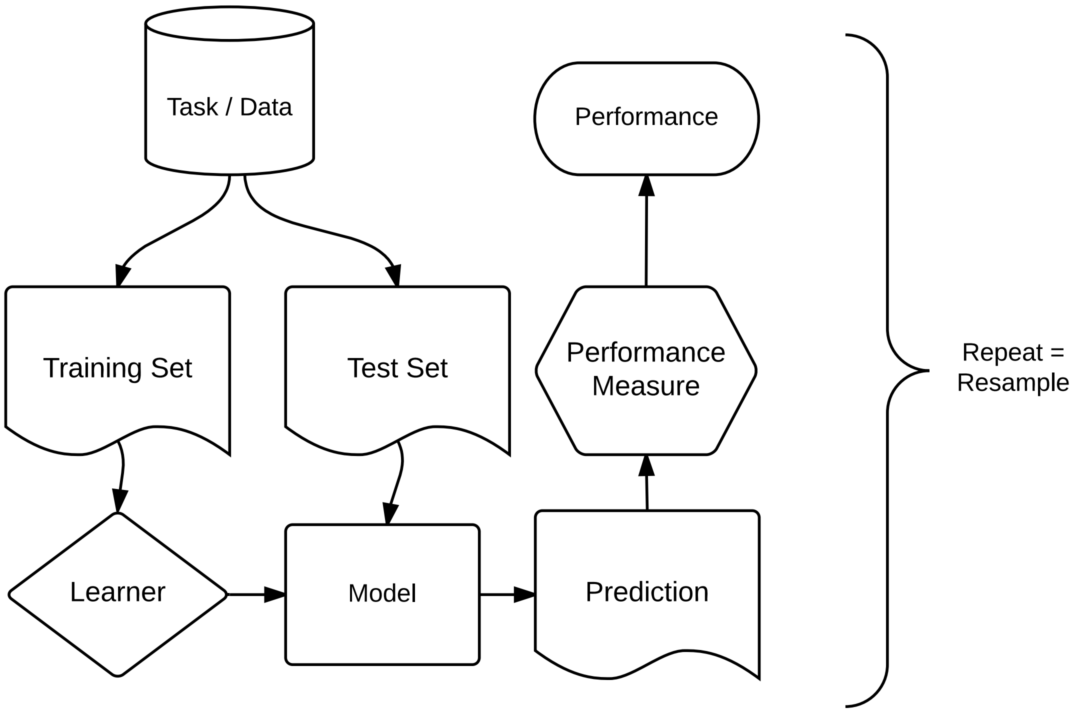

Figure 2.1: General overview of the machine learning process.

2.1 Tasks

Tasks are objects that contain the (usually tabular) data and additional meta-data that define a machine learning problem. The meta-data contain, for example, the name of the target feature for supervised machine learning problems. This information is used automatically by operations that can be performed on a task so that for example the user does not have to specify the prediction target every time a model is trained.

2.1.1 Built-in Tasks

mlr3 includes a few predefined machine learning tasks in an R6 Dictionary named mlr_tasks.

```
mlr_tasks
```

```
<DictionaryTask> with 19 stored values
Keys: bike_sharing, boston_housing, breast_cancer, german_credit, ilpd,
  iris, kc_housing, moneyball, mtcars, optdigits, penguins,
  penguins_simple, pima, sonar, spam, titanic, usarrests, wine, zoo
```

To get a task from the dictionary, use the tsk() function and assign the return value to a new variable. Here, we retrieve the mtcars regression task, which is provided by the package datasets:

```
task_mtcars = tsk("mtcars")
task_mtcars
```

```
<TaskRegr:mtcars> (32 x 11): Motor Trends
* Target: mpg
* Properties: -
* Features (10):
  - dbl (10): am, carb, cyl, disp, drat, gear, hp, qsec, vs, wt
```

To get more information about a particular task, it is easiest to use the help() method that all mlr3-objects come with:

```
task_mtcars$help()
```

Tip

If you are familiar with R's help system (i.e. the help() and ? functions), this may seem confusing. task_mtcars is the variable that holds the mtcars task, not a function, and hence we cannot use help() or ?.

Alternatively, the corresponding man page can be found under mlr_tasks_<id>, e.g.

```
help("mlr_tasks_mtcars")
```

We can also load the data separately and convert it to a task, without using the tsk() function that mlr3 provides. If the data we want to use does not come with mlr3, it has to be done this way.

For example, the data for `mtcars` is also available separately, as a `data.frame()` and not a task. `mtcars` contains characteristics for different types of cars, along with their fuel consumption. We want to predict the numeric target feature stored in column `"mpg"` (miles per gallon).

```r
data("mtcars", package = "datasets")
str(mtcars)
```

```
'data.frame':	32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

We create the regression task, i.e. we construct a new instance of the R6 class `TaskRegr`. An easy way to do this is to use the function `as_task_regr()` to convert our `data.frame()` to a regression task, specifying the target feature in an additional argument. Before we give the data to `as_task_regr()`, we can process it using the usual R functions, for example to select a subset of data.

```r
library("mlr3")
mtcars_subset = subset(mtcars, select = c("mpg", "cyl", "disp"))

task_mtcars = as_task_regr(mtcars_subset, target = "mpg", id = "cars")
task_mtcars
```

```
<TaskRegr:cars> (32 x 3)
* Target: mpg
* Properties: -
* Features (2):
  - dbl (2): cyl, disp
```

Tip

The task constructors `as_task_regr()` and `as_task_classif()` will check for non-ASCII characters in the column names of your data. As many ML models do not work properly with arbitrary UTF8 names, `mlr3` defaults to throw an error if any of the column names contains either a non-ASCII character or does not comply with R's variable naming scheme. We generally recommend converting names with `make.names()` first, but you can also set the