Applied Machine Learning

Machine Learning in R:
MLR3 Basics & Data Handling

Learning goals

- Introduction to MLR3 ecosystem
- Understanding Tasks and Data structures
- Working with Dictionaries

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- . . . but without a unified interface
- things like performance evaluation are cumbersome

Example:

```
# Specify what we want to model in a formula: target ~ features
svm_model = e1071::svm(Species ~ ., data = iris)
```

vs.

```
# Pass the features as a matrix and the target as a vector
xgb_model = xgboost::xgboost(data = as.matrix(iris[1:4]),
  label = iris$Species, nrounds = 10)
```

SO YOU WANT TO DO ML IN R

```r
library("mlr3")
```

Ingredients:

- Data / Task
- Learning Algorithms
- Performance Evaluation
- Performance Comparison

R6 – ALL YOU NEED TO KNOW

`mlr3` uses the *R6* class system which facilitates OOP by allowing the creation of custom objects with methods and properties (it may look unusual if you see it the first time).

- *Objects* are created using `<Class>$new()`.

```r
task = TaskClassif$new(id = "iris", backend = iris, target = "Species")
```

- Alternatively, the function `as_task_classif` can be used (or `as_task_regr` to construct a `TaskRegr` object for regression tasks). By default, the name of the object passed to `x` is used as `id`:

```r
task = as_task_classif(x = iris, target = "Species")
```

R6 – ALL YOU NEED TO KNOW

mlr3 uses the *R6* class system which facilitates OOP by allowing the creation of custom objects with methods and properties (it may look unusual if you see it the first time).

- *Objects* are created using <Class>$new().

```
task = TaskClassif$new(id = "iris", backend = iris, target = "Species")
```

- Alternatively, the function as_task_classif can be used (or as_task_regr to construct a TaskRegr object for regression tasks). By default, the name of the object passed to x is used as id:

```
task = as_task_classif(x = iris, target = "Species")
```

- Objects have *fields* that contain information about the object.

```
task$nrow
#> [1] 150
```

R6 – ALL YOU NEED TO KNOW

mlr3 uses the *R6* class system which facilitates OOP by allowing the creation of custom objects with methods and properties (it may look unusual if you see it the first time).

- *Objects* are created using <Class>$new().

```
task = TaskClassif$new(id = "iris", backend = iris, target = "Species")
```

- Alternatively, the function as_task_classif can be used (or as_task_regr to construct a TaskRegr object for regression tasks). By default, the name of the object passed to x is used as id:

```
task = as_task_classif(x = iris, target = "Species")
```

- Objects have *fields* that contain information about the object.

```
task$nrow

#> [1] 150
```

- Objects have *methods* that are called like functions:

```
task$filter(rows = 1:10)
```

R6 – ALL YOU NEED TO KNOW

`mlr3` uses the *R6* class system which facilitates OOP by allowing the creation of custom objects with methods and properties (it may look unusual if you see it the first time).

- *Objects* are created using `<Class>$new()`.

```r
task = TaskClassif$new(id = "iris", backend = iris, target = "Species")
```

- Alternatively, the function `as_task_classif` can be used (or `as_task_regr` to construct a `TaskRegr` object for regression tasks). By default, the name of the object passed to `x` is used as `id`:

```r
task = as_task_classif(x = iris, target = "Species")
```

- Objects have *fields* that contain information about the object.

```r
task$nrow

#> [1] 150
```

- Objects have *methods* that are called like functions:

```r
task$filter(rows = 1:10)
```

- Methods may change (“mutate”) the object (reference semantics)!

```r
task$nrow

#> [1] 10
```

R6 AND ACTIVE BINDINGS

Some fields of R6-objects may be *“Active Bindings”*. Internally they are realized as functions that are called whenever the value is set or retrieved.

- Active bindings for read-only fields

```
task$nrow = 11
#> Error in assert_ro_binding(rhs):  Field/Binding is
read-only
```

R6 AND ACTIVE BINDINGS

Some fields of R6-objects may be *“Active Bindings”*. Internally they are realized as functions that are called whenever the value is set or retrieved.

- Active bindings for read-only fields

```
task$nrow = 11
#> Error in assert_ro_binding(rhs):  Field/Binding is
read-only
```

- Active bindings for argument checking

```
task$properties = NULL
#> Error in assert_set(rhs, .var.name = "properties"):
Assertion on ’properties’ failed:  Must be of type
’character’, not ’NULL’.
task$properties = c("property1", "property2")  # works
```

MLR3 PHILOSOPHY

- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics

MLR3 PHILOSOPHY

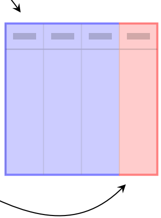
- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics
- Embrace **data.table**, both for arguments and internally
  - Fast operations for tabular data
  - List columns to arrange complex objects in tabular structure

MLR3 PHILOSOPHY

- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics
- Embrace **data.table**, both for arguments and internally
  - Fast operations for tabular data
  - List columns to arrange complex objects in tabular structure
- Be **light on dependencies**:
  - `R6`, `data.table`, `lgr`, `uuid`, `mlbench`, `digest`
  - Plus some of our own packages (`backports`, `checkmate`, ...)

DATA

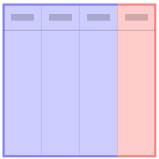
- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression
  - ⇒ target determines the machine learning “Task”



DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  ⇒ target determines the machine learning “Task”



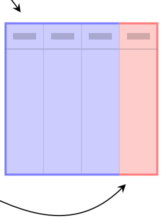
```
print(iris)  # included in R

#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1          5.1         3.5          1.4         0.2  setosa
#> 2          4.9         3.0          1.4         0.2  setosa
#> ...
```

DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  ⇒ target determines the machine learning “Task”



```
print(iris)  # included in R

#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1          5.1         3.5          1.4         0.2  setosa
#> 2          4.9         3.0          1.4         0.2  setosa
#> ...
```

Task ID ↘ data ↓ target name ↙

```
task = TaskClassif$new("iris", iris, "Species")
task = as_task_classif(x = iris, target = "Species", id = "iris")
```

DATA

```
task = as_task_classif(x = iris, target = "Species")
```

```
print(task)

# <TaskClassif:iris> (150 x 5)
# * Target: Species
# * Properties: multiclass
# * Features (4):
#   - dbl (4): Petal.Length, Petal.Width, Sepal.Length,
#     Sepal.Width
```

```
task$ncol
task$nrow
task$feature_names
task$target_names
```

```
task$head(n = )
task$truth(row_ids = )
task$data(rows = ,
          cols = )
```

```
task$select(cols = )
task$filter(rows = )
task$cbind(data = )
task$rbind(data = )
```

DICTIONARIES

- `mlr3` uses `R6` classes to create dictionaries that store key-value pairs, i.e., associate keys (unique identifiers) with values (R6 objects).
- Dictionaries are easily extendable and allow adding and removing key-value pairs, e.g., add-on packages such as `mlr3learners` populate dictionaries with additional key-value pairs.

DICTIONARIES

- `mlr3` uses `R6` classes to create dictionaries that store key-value pairs, i.e., associate keys (unique identifiers) with values (R6 objects).
- Dictionaries are easily extendable and allow adding and removing key-value pairs, e.g., add-on packages such as `mlr3learners` populate dictionaries with additional key-value pairs.
- `mlr3` offers *Short Form* functions to get objects from a `Dictionary`:

Object	Dictionary	Short Form
`Task`	`mlr_tasks`	`tsk()`
`Learner`	`mlr_learners`	`lrn()`
`Measure`	`mlr_measures`	`msr()`
`Resampling`	`mlr_resamplings`	`rsmp()`

DICTIONARIES

```
# list items
tsk()

#> <DictionaryTask> with 11 stored values
#> Keys: boston_housing, breast_cancer, german_credit, iris,
#>   mtcars, penguins, pima, sonar, spam, wine, zoo

# retrieve object
tsk("iris")

#> <TaskClassif:iris> (150 x 5): Iris Flowers
#> * Target: Species
#> * Properties: multiclass
#> * Features (4):
#>   - dbl (4): Petal.Length, Petal.Width, Sepal.Length,
#>     Sepal.Width
```

SHORT FORMS AND DICTIONARIES

`as.data.table(<DICTIONARY>)` creates a `data.table` with metadata about objects in dictionaries:

```
as.data.table(mlr_learners)[1:5, c("key", "packages", "predict_types")]

# Key: <key>
#                     key   packages predict_types
#                  <char>     <list>        <list>
# 1:        classif.debug       mlr3 response,prob
# 2: classif.featureless       mlr3 response,prob
# 3:        classif.rpart mlr3,rpart response,prob
# 4:           regr.debug       mlr3   response,se
# 5:     regr.featureless mlr3,stats   response,se

library(mlr3learners) # mlr_learners dictionary gets populated
as.data.table(mlr_learners)[1:5, c("key", "packages", "predict_types")]

# Key: <key>
#                     key                 packages predict_types
#                  <char>                   <list>        <list>
# 1:   classif.cv_glmnet mlr3,mlr3learners,glmnet response,prob
# 2:       classif.debug                     mlr3 response,prob
# 3: classif.featureless                     mlr3 response,prob
# 4:      classif.glmnet mlr3,mlr3learners,glmnet response,prob
# 5:        classif.kknn   mlr3,mlr3learners,kknn response,prob
```