Modern Machine Learning in R

mlr3

Department of Statistics – LMU Munich

Intro

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:

```
svm_model = e1071::svm(Species ~ ., data = iris)
```

vs.

```
xgb_model = xgboost::xgboost(as.matrix(iris[1:4]), iris$Species,
  nrounds = 10)
```

SO YOU WANT TO DO ML IN R

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

Ingredients:

- Data
- Learning Algorithms
- Performance Evaluation
- Performance Comparison

R6

R6 – ALL YOU NEED TO KNOW

`mlr3` uses the *R6* class system. Some things may seem unusual if you see them for the first time.

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

```
task = TaskClassif$new("iris", iris, "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)
```

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

```
task$nrow

#> [1] 10
```

R6 AND REFERENCE SEMANTICS

R6 objects have *"Reference Semantics"*: copies have to created explicitly with `$clone()` if they should not be changed.

- We conduct an experiment: `task_two` is not a copy of `task` but refers to the *same* object:

```
task = TaskClassif$new("iris", iris, "Species")
task_two = task
task_clone = task$clone(deep = TRUE)
```

- We mutate `task`:

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

- `task_two` has changed, `task_clone` has not.

```
task$nrow

#> [1] 10

task_two$nrow

#> [1] 10

task_clone$nrow

#> [1] 150
```

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 (function () :  unused argument (base::quote(11))
```

- 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
```

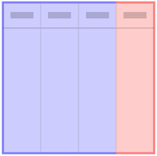
MLR3 PHILOSOPHY

- 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
- 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`, `Metrics`, `lgr`, `uuid`, `mlbench`, `digest`
  - Plus some of our own packages (`backports`, `checkmate`, ...)

Data

DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression
  - ⇒ data 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")
```

DATA

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

```r
print(task)

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

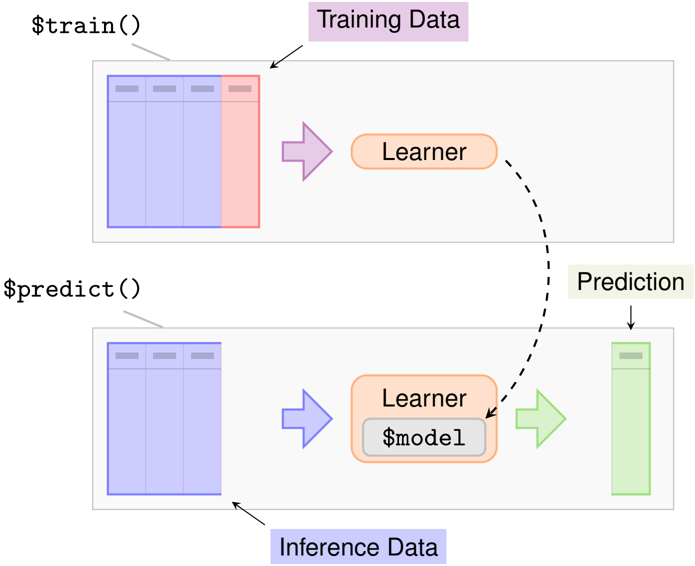
```r
task$ncol
task$nrow
task$feature_names
task$target_names
```

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

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

Learning Algorithms

LEARNING ALGORITHMS



LEARNING ALGORITHMS

- Get a `Learner` provided by `mlr`

```
learner = lrn("classif.rpart")
```

- Train the `Learner`

```
learner$train(task)
```

- The `$model` is the `rpart` model: a decision tree

```
print(learner$model)

#> n= 150
#>
#> node), split, n, loss, yval, (yprob)
#>       * denotes terminal node
#>
#> 1) root 150 100 setosa (0.333 0.333 0.333)
#>   2) Petal.Length< 2.5 50   0 setosa (1.000 0.000 0.000) *
#>   3) Petal.Length>=2.5 100  50 versicolor (0.000 0.500 0.500)
#>     6) Petal.Width< 1.8 54   5 versicolor (0.000 0.907 0.093) *
#>     7) Petal.Width>=1.8 46   1 virginica (0.000 0.022 0.978) *
```

HYPERPARAMETERS

- Learners have *hyperparameters*

```
learner$param_set

#> ParamSet:
#>                id    class lower upper levels default value
#> 1:       minsplit ParamInt     1   Inf             20
#> 2:             cp ParamDbl     0     1           0.01
#> 3:     maxcompete ParamInt     0   Inf              4
#> 4:   maxsurrogate ParamInt     0   Inf              5
#> 5:       maxdepth ParamInt     1    30             30
#> 6:           xval ParamInt     0   Inf             10     0
```

- Changing them changes the `Learner` behavior

```
learner$param_set$values = list(maxdepth = 1, xval = 0)

learner$train(task)
```

HYPERPARAMETERS

- This gives a smaller decision tree

```
print(learner$model)

#> n= 150
#>
#> node), split, n, loss, yval, (yprob)
#>       * denotes terminal node
#>
#> 1) root 150 100 setosa (0.33 0.33 0.33)
#>   2) Petal.Length< 2.5 50   0 setosa (1.00 0.00 0.00) *
#>   3) Petal.Length>=2.5 100  50 versicolor (0.00 0.50 0.50) *
```

- Instead of assigning `$values` a `list()`, we can change individual parameters

```
learner$param_set$values$maxdepth = 10
```

PREDICTION

- Let's make a prediction

```
new_data

#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1            4           3            2           1
# 2            2           2            3           2
```

- Call `$predict_newdata()` with the data

```
prediction = learner$predict_newdata(new_data)
```

- We get a `Prediction` object:

```
prediction

#> <PredictionClassif> for 2 observations:
#>   row_id truth   response
#>        1  <NA>     setosa
#>        2  <NA> versicolor
```

PREDICTION

- We can make the `Learner` predict *probabilities* when we set `predict_type`:

```
learner$predict_type = "prob"
learner$predict_newdata(new_data)
```

```
# <PredictionClassif> for 2 observations:
#  row_id truth  response prob.setosa prob.versicolor
#       1  <NA>    setosa           1             0.0
#       2  <NA> virginica           0             0.5
#  prob.virginica
#             0.0
#             0.5
```

PREDICTION

What exactly is a `Prediction` object?

- Contains predictions and offers useful access fields / methods

⇒ Raw data in `$data`

```
prediction$data

#> $tab
#>    row_id truth   response
#> 1:      1  <NA>     setosa
#> 2:      2  <NA> versicolor
```

⇒ Active bindings and functions that give further information: `$response`, `$truth`, ...

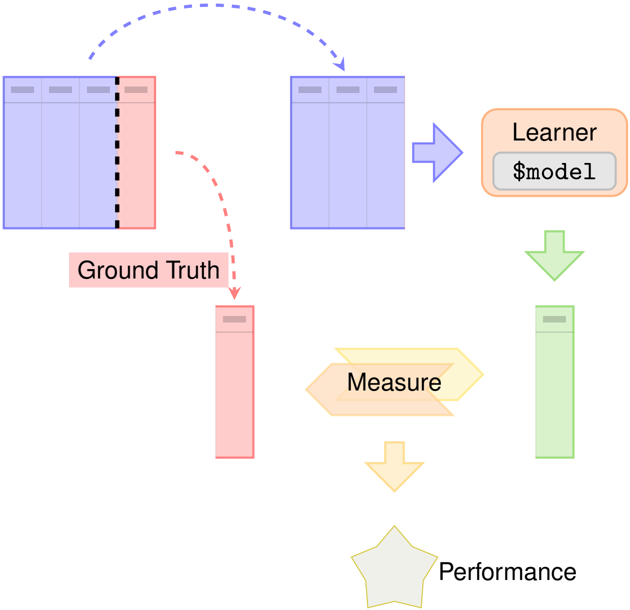
```
prediction$response

#> [1] setosa     versicolor
#> Levels: setosa versicolor virginica
```

⇒ Use `as.data.table()` to get a `data.table` for analysis

```
as.data.table(prediction)

#>    row_id truth   response
#> 1:      1  <NA>     setosa
#> 2:      2  <NA> versicolor
```

Performance

PERFORMANCE EVALUATION



PERFORMANCE EVALUATION

- Prediction ‘Task‘ with known data

```
known_truth_task$data()

#     Species Petal.Length Petal.Width Sepal.Length Sepal.Width
# 1:   setosa            2           1            4           3
# 2:   setosa            3           2            2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)
pred

#> <PredictionClassif> for 2 observations:
#>  row_id  truth  response
#>       1 setosa    setosa
#>       2 setosa virginica
```

- Score the prediction

```
pred$score(msr("classif.ce"))

#> classif.ce
#>        0.5
```

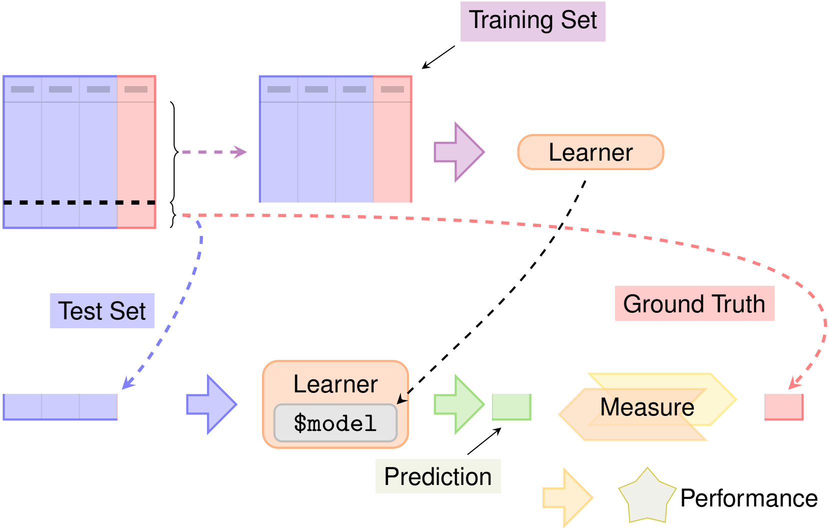
PERFORMANCE EVALUATION

- Confusion Matrix

```
pred

#> <PredictionClassif> for 2 observations:
#>  row_id  truth  response
#>       1 setosa    setosa
#>       2 setosa virginica

pred$confusion

#>             truth
#> response     setosa versicolor virginica
#>   setosa          1          0         0
#>   versicolor      0          0         0
#>   virginica       1          0         0
```

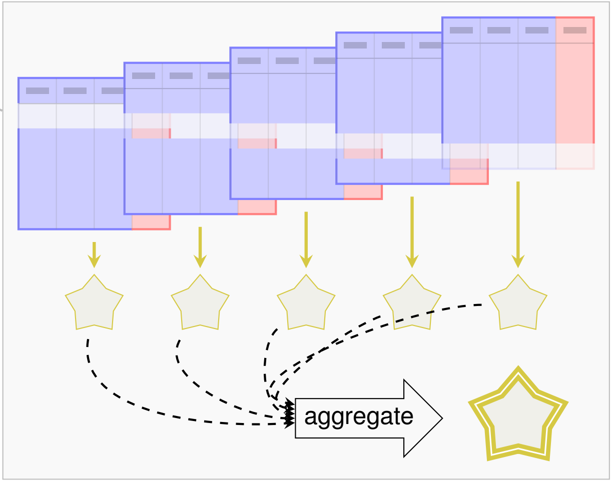
Resampling

RESAMPLING


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Measure
Prediction
Performance

RESAMPLING

`resample()`



RESAMPLING

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

- Use the `resample()` function for resampling:

```
rr = resample(task, learner, cv5)
```

- We get a `ResamplingResult` object:

```
print(rr)

#> <ResampleResult> of 5 iterations
#> * Task: iris
#> * Learner: classif.rpart
#> * Warnings: 0 in 0 iterations
#> * Errors: 0 in 0 iterations
```

RESAMPLING RESULTS

What exactly is a `ResamplingResult` object?
Remember `Prediction`:

- Raw data in `$data` field
- Get a table representation using `as.data.table()`

```
rr_table = as.data.table(rr)

print(rr_table)

#                task                 learner     resampling iter...
# 1: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>      ...
# 2: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>      ...
# 3: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>      ...
# 4: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>      ...
# 5: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>      ...
```

- Active bindings and functions that make information easily accessible

RESAMPLING RESULTS

- Get performance:

```r
rr$aggregate(msr("classif.ce"))
```

```
#> classif.ce
#>       0.06
```

- Get predictions

```r
rr$prediction()
```

```
#> <PredictionClassif> for 150 observations:
#>     row_id     truth   response
#>          8    setosa     setosa
#>         10    setosa     setosa
#>         32    setosa     setosa
#> ---
#>        118 virginica  virginica
#>        119 virginica  virginica
#>        130 virginica versicolor
```

RESAMPLING

- Predictions of individual folds

```
predictions = rr$predictions()
predictions[[1]]
```

```
#> <PredictionClassif> for 30 observations:
#>     row_id     truth  response
#>          8    setosa    setosa
#>         10    setosa    setosa
#>         32    setosa    setosa
#> ---
#>        144 virginica virginica
#>        148 virginica virginica
#>        149 virginica virginica
```

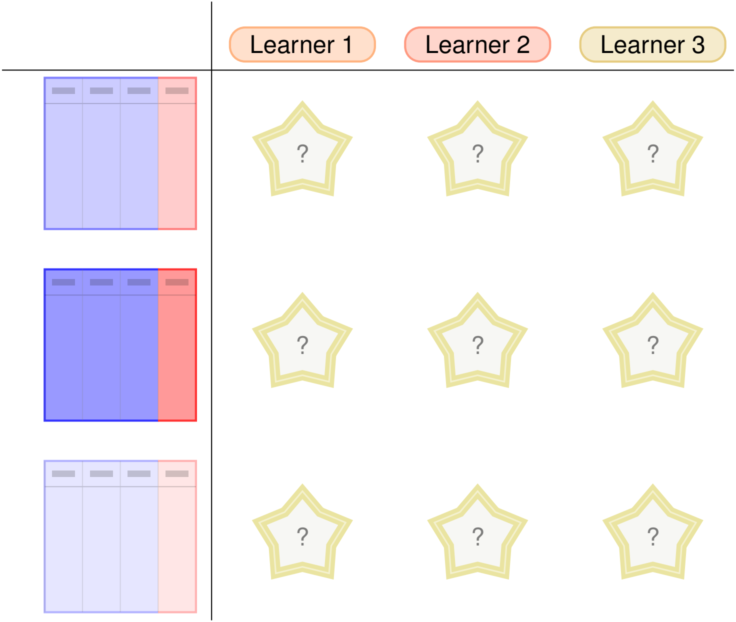
- Score of individual folds

```
scores = rr$score()
scores[1:3, c("iteration", "classif.ce")]
```

```
#>    iteration classif.ce
#> 1:         1      0.100
#> 2:         2      0.033
#> 3:         3      0.033
```

Benchmark

PERFORMANCE COMPARISON



PERFORMANCE COMPARISON

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

- Set up the *design* and execute benchmark:

```
design = benchmark_grid(tasks, learners, cv5)
bmr = benchmark(design)
```

- We get a `BenchmarkResult` object which shows that `kknn` outperforms `rpart`:

```
bmr_ag = bmr$aggregate()
bmr_ag[, c("task_id", "learner_id", "classif.ce")]

#>    task_id    learner_id classif.ce
#> 1:    iris classif.rpart      0.053
#> 2:    iris  classif.kknn      0.053
#> 3:   sonar classif.rpart      0.221
#> 4:   sonar  classif.kknn      0.149
#> 5:    wine classif.rpart      0.106
#> 6:    wine  classif.kknn      0.034
```

BENCHMARK RESULT

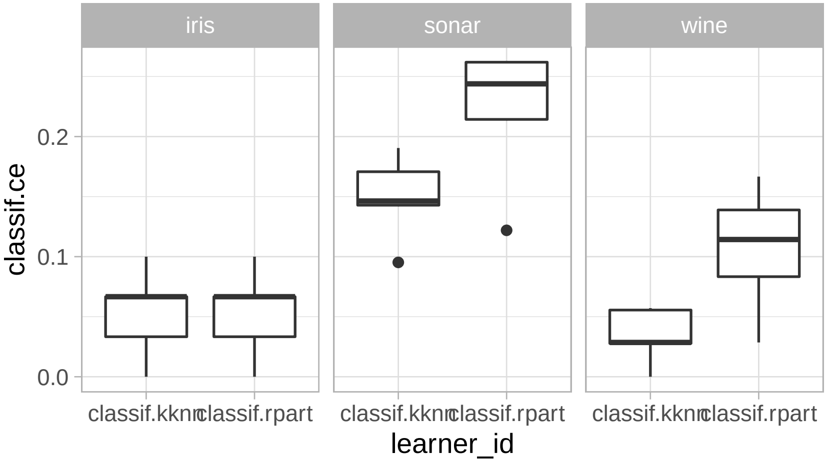
What exactly is a `BenchmarkResult` object?
Just like `Prediction` and `ResamplingResult`!

- Raw data in `$data` field
- Table representation using `as.data.table()`
- Active bindings and functions that make information easily accessible

BENCHMARK RESULT

The `mlr3viz` package contains `autoplot()` functions for some mlr3 objects

```r
library(mlr3viz)
autoplot(bmr)
```



Short Forms and Dictionaries

SHORT FORMS AND DICTIONARIES

- Ordinary constructors: `LearnerClassifRpart$new()`

⇒ `mlr3` offers *Short Form Constructors* that are less verbose

- They access `Dictionary` of objects:

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

- Use `Dictionary$keys()` method to list available items

```
mlr_resamplings$keys()

#> [1] "bootstrap"   "custom"      "cv"             "holdout"
#> [5] "repeated_cv" "subsampling"
```

- Dictionaries can get populated by add-on packages (e.g. `mlr3learners`)

SHORT FORMS AND DICTIONARIES

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

```
mlr_learners_table = as.data.table(mlr_learners)

mlr_learners_table[1:10, c("key", "packages", "predict_types")]

#                     key    packages predict_types
#  1:         classif.debug              response,prob
#  2: classif.featureless              response,prob
#  3:       classif.glmnet      glmnet response,prob
#  4:         classif.kknn withr,kknn response,prob
#  5:          classif.lda        MASS response,prob
#  6:      classif.log_reg       stats response,prob
#  7: classif.naive_bayes       e1071 response,prob
#  8:          classif.qda        MASS response,prob
#  9:       classif.ranger      ranger response,prob
# 10:        classif.rpart       rpart response,prob
```

How to get Help

HOW TO GET HELP

- Where to start?
  - Check these slides
  - **Check the mlr3book https://mlr3book.mlr-org.com**
- Get help for R6 objects?
  1. Find out what kind of R6 object you have:

     ```
     class(bmr)
     #> [1] "BenchmarkResult" "R6"
     ```

  2. Go to the corresponding help page:

     ```
     ?BenchmarkResult
     ```

- Why does this not work?
  - Ask at stackoverflow https://stackoverflow.com/questions/tagged/mlr3
  - Write a GitHub issue (in the according project)

Advanced Topics

CONTROL OF EXECUTION

Parallelization

```
future::plan("multicore")
```

- runs each resampling iteration as a job
- also allows nested resampling (although not needed here)

Encapsulation

```
learner$encapsulate = c(train = "callr", predict = "callr")
```

- Spawns a separate R process to train the learner
- Learner may segfault without tearing down the session
- Logs are captured
- Possibilty to have a fallback to create predictions

OUT-OF-MEMORY DATA

- Task stores data in a `DataBackend`:
  - `DataBackendDataTable`: Default backend for dense data (in-memory)
  - `DataBackendMatrix`: Backend for sparse numerical data (in-memory)
  - `DataBackendDplyr`: Backend for many DBMS (out-of-memory)
  - `DataBackendCbind`: Combine backends in a `cbind()` fashion (virtual)
  - `DataBackendRbind`: Combine backends in a `rbind()` fashion (virtual)
- Backends are immutable
  - Filtering rows or selecting columns just modifies the "view" on the data
  - Multiple tasks can share the same backend
- Example: Interface a read-only MariaDB with `DataBackendDplyr`, add generated features via `DataBackendDataTable`

Outro

OVERVIEW

Main things remember about `mlr3`:

- Short forms & Data / Control Objects

  `tsk()` $\mapsto$ `Task`

  `lrn()` $\mapsto$ `Learner`

  `rsmp()` $\mapsto$ `Resampling`

  `msr()` $\mapsto$ `Measure`

- Result Objects
  - `ResampleResult`, `BenchmarkResult`
  - Have `$data` slot and provide `as.data.table()`
- Functions
  - `resample()`:

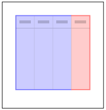
    (`Task`, `Learner`, `Resampling`) $\mapsto$ `ResampleResult`
  - `benchmark_grid(), benchmark()`:

    (`Task`, `Learner`, `Resampling`) $\mapsto$ `BenchmarkResult`

SO YOU WANT TO DO ML IN R

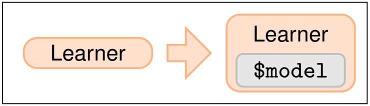
Ingredients:

Data



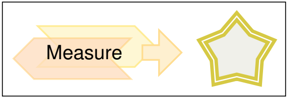
`TaskClassif`,
`TaskRegr`,
`tsk()`

Learning Algorithms



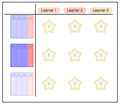
`lrn()` ⇒ `Learner`,
`$train()`,
`$predict()` ⇒ `Prediction`

Performance Evaluation



`rsmp()` ⇒ `Resampling`,
`msr()` ⇒ `Measure`,
`resample()` ⇒ `ResamplingResult`,
`$aggregate()`

Performance Comparison



`benchmark_grid()`,
`benchmark()` ⇒ `BenchmarkResult`