mlr3

Modern machine learning in R

Michel Lang, Bernd Bischl, Jakob Richter, Patrick Schratz, Martin Binder

bit.ly/2LMwE7W

mlr-v2

Meta framework for everything machine learning (evaluation, visualization, tuning, wrapping, bagging, ...)

Monolithic package

- Interfaces > 150 learners
  → Dependencies (direct / recursive): 119 / 1436
  → Unit tests take > 2h
  → Continuous integration split into multiple stages, rather unstable
- Most unit tests disabled for CRAN to comply to their policy
  → No tests in reverse dependency checks on CRAN
  → Package developers changed their API and (unknowingly) broke mlr
- High barrier for new contributors

mlr-v2

Missing OO

- S3 reaches its limitations in larger software projects
- Many different container types for results with awkward accessors: `getBMRAggrPerformances()`
- `NAMESPACE` has > 1200 lines, > 440 exported functions and objects
- Wrappers (pipelines) hard to customize and to work with

Further Design Issues

- Only works on in-memory data
- No nested parallelization

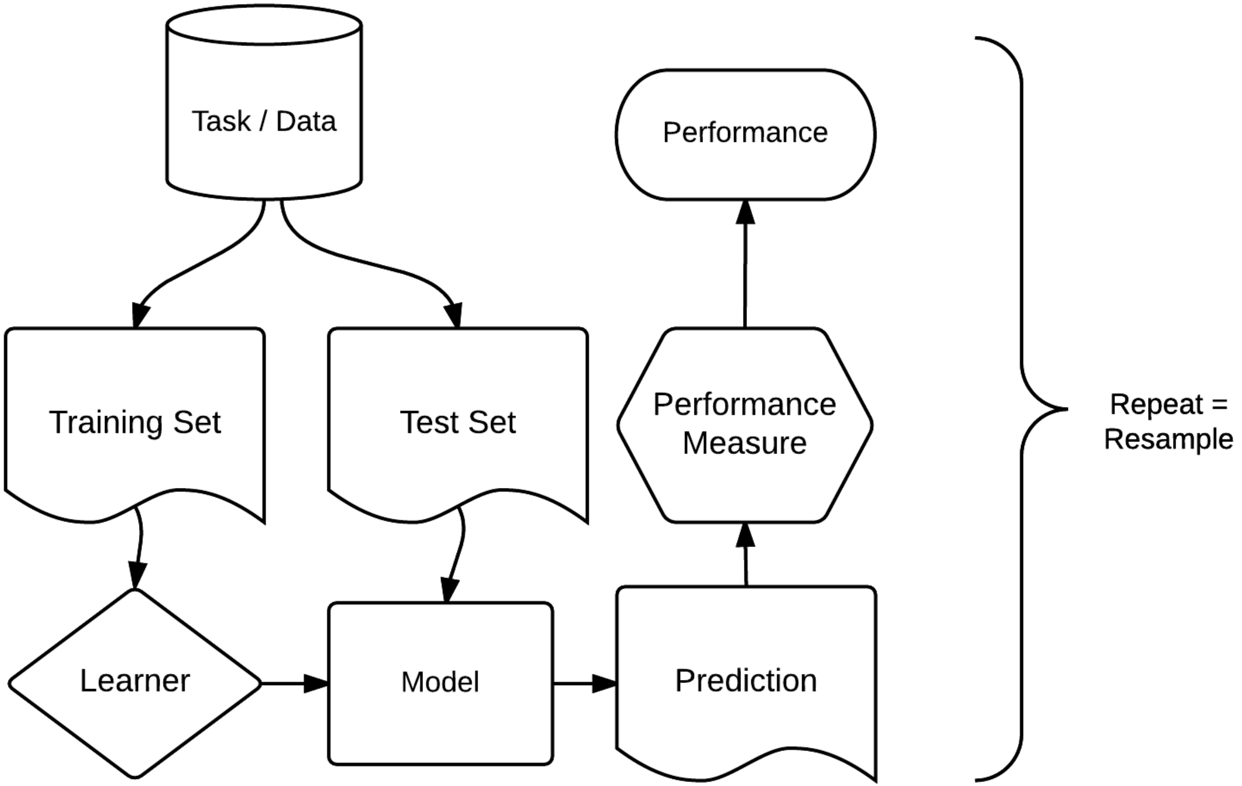
mlr3

mlr3

- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented (OO): data and methods together
  - Inheritance
  - Reference semantics
- Embrace **data.table**, both for arguments and for internal data structures
  - Fast operations for tabular data
  - Better support for list columns to arrange complex objects in a tabular structure
  - Reference semantics
- Be **light on dependencies**. Direct and recursive dependencies:
  - `R6`, `data.table`, `Metrics`, `lgr`
  - Some self-maintained packages (`backports`, `checkmate`, ...)

Building Blocks

Building Blocks



Tasks

→ Create your own task

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

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

→ Retrieve a predefined task from the task dictionary

```
mlr_tasks
```

```
## <DictionaryTask> with 9 stored values
## Keys: boston_housing, german_credit, iris, mtcars, pima, sonar,
##   spam, wine, zoo
```

```
task = mlr_tasks$get("iris")
```

Learner

→ Retrieve a predefined learner from the learner dictionary

```
mlr_learners
```

```
## <DictionaryLearner> with 21 stored values
## Keys: classif.debug, classif.featureless, classif.glmnet,
##   classif.kknn, classif.lda, classif.log_reg, classif.naive_bayes,
##   classif.qda, classif.ranger, classif.rpart, classif.svm,
##   classif.xgboost, regr.featureless, regr.glmnet, regr.kknn,
##   regr.km, regr.lm, regr.ranger, regr.rpart, regr.svm,
##   regr.xgboost
```

Learner

→ Retrieve a predefined learner from the learner dictionary

```
learner = mlr_learners$get("classif.rpart")
print(learner)
```

```
## <LearnerClassifRpart:classif.rpart>
## Model: -
## Parameters: xval=0
## Packages: rpart
## Predict Type: response
## Feature types: logical, integer, numeric, character, factor,
##   ordered
## Properties: importance, missings, multiclass, selected_features,
##   twoclass, weights
```

Learner

→ Querying and setting hyperparameters

```
# query
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
```

```
# set
learner$param_set$values = list(xval = 0, cp = 0.1)
```

Learner

→ Training

```
task = mlr_tasks$get("iris")
learner$train(task, row_ids = 1:120)
```

NB: This changes the learner in-place, model is now stored inside the learner.

Learner

→ Accessing the learner model

```
learner$model
```

```
## n= 120
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 120 70 setosa (0.41666667 0.41666667 0.16666667)
##   2) Petal.Length< 2.45 50  0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 70 20 versicolor (0.00000000 0.71428571 0.28571429)
##     6) Petal.Length< 4.95 49  1 versicolor (0.00000000 0.97959184 0.02040816) *
##     7) Petal.Length>=4.95 21  2 virginica (0.00000000 0.09523810 0.90476190) *
```

→ Variable importance

```
learner$importance()
```

```
## Petal.Length  Petal.Width Sepal.Length  Sepal.Width
##     69.42177     65.04211     41.85520     29.11840
```

Predictions

→ Generate predictions

```r
p = learner$predict(task, row_ids = 121:150)
head(as.data.table(p), 3)
```

```
##    row_id     truth   response
## 1:    121 virginica  virginica
## 2:    122 virginica versicolor
## 3:    123 virginica  virginica
```

→ Confusion matrix

```r
p$confusion
```

```
##             truth
## response     setosa versicolor virginica
##   setosa          0          0         0
##   versicolor      0          0         5
##   virginica       0          0        25
```

Performance Assessment

→ Retrieve a predefined measure from the measure dictionary

```
measure = mlr_measures$get("classif.acc")
measure
```

```
## <MeasureClassifACC:classif.acc>
## Packages: Metrics
## Range: [0, 1]
## Minimize: FALSE
## Properties: -
## Predict type: response
```

→ Calculate performance

```
p$score(c("classif.acc", "time_train"))
```

```
## classif.acc  time_train
##   0.8333333   0.0000000
```

Rinse and Repeat

Resample

→ Resampling Object

```
cv3 = mlr_resamplings$get("cv", param_vals = list(folds = 3))
```

Splits into train/test are efficiently stored and can be accessed with `$train_set(i)` and `$test_set(i)`.

→ Resample a regression tree on the Boston housing data using a 3-fold CV

```
# string -> object conversion via dictionary
rr = resample("boston_housing", "regr.rpart", cv3)
```

→ Aggregated performance

```
rr$aggregate("regr.mse")
```

```
## regr.mse
## 2.973355
```

Benchmarking

→ Exhaustive grid design

```
grid = expand_grid(
    tasks = "iris",
    learners = c("classif.featureless", "classif.rpart"),
    resamplings = "cv3"
)
bmr = benchmark(grid, ctrl = list(store_models = TRUE))
aggr = bmr$aggregate("classif.acc")
aggr[, 2:6]
```

```
##     resample_result task_id          learner_id resampling_id classif.acc
## 1: <ResampleResult>    iris classif.featureless           cv3   0.2866667
## 2: <ResampleResult>    iris       classif.rpart           cv3   0.9466667
```

Benchmarking

→ Retrieving objects

```
aggr$resample_result[[2]]$prediction$confusion
```

```
##             truth
## response     setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         45         3
##   virginica       0          5        47
```

Tuning

- Algorithms: *Grid Search, Random Search, Simulated Annealing*
- In process: *Bayesian Optimization, iterated F-racing, EAs*
- Budget via class `Terminator`: iterations, performance, runtime, real time
- Nested resampling via class `AutoTuner`

```
ps = ParamSet$new(list(
  ParamInt$new("num.trees", lower = 50, upper = 500),
  ParamInt$new("mtry", lower = 1, upper = 5)
))

at = AutoTuner$new(
  learner = "classif.ranger",
  resampling = "cv3", # inner resampling
  measures = "classif.acc",
  param_set = ps,
  terminator = TerminatorEvaluations$new(10),
  tuner = TunerRandomSearch
)

resample(
  task = "spam",
  learner = at,
  resampling = "holdout" # outer resampling
)
```

Behind the Curtain

Internal Data Structure

All result objects (`resample()`, `benchmark()`, tuning, ...) share the same structure:

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

```
##                learner       prediction      task     resampling iteration
## 1: <LearnerRegrRpart> <PredictionRegr> <TaskRegr> <ResamplingCV>         1
## 2: <LearnerRegrRpart> <PredictionRegr> <TaskRegr> <ResamplingCV>         2
## 3: <LearnerRegrRpart> <PredictionRegr> <TaskRegr> <ResamplingCV>         3
```

Combining R6 and data.table

- Not the objects are stored, but pointers to them
- Inexpensive to work on:
  - `rbind()`: copying R6 objects ≙ copying pointers
  - `cbind()`: `data.table()` over-allocates columns, no copies
  - `[i, ]`: lookup row (possibly hashed), create a list of pointers
  - `[, j]`: direct access to list element

Control of Execution

→ Parallelization

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

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

→ Encapsulation

```
ctrl = mlr_control(encapsulate_train = "callr")
benchmark(grid, ctrl = ctrl)
```

- Spawns a separate R process to train the learner
- Learner may segfault without tearing down the master session
- Logs are captured
- Possibility to have a fallback learner 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`

Current state

- Preview release uploaded to CRAN
- Started extension packages:
  - `mlr3db` for additional backends
  - `mlr3pipelines` to create workflows
  - `mlr3learners` for recommended learners
  - `mlr3tuning` for tuning
  - `mlr3survival` for survival analysis
  - `mlr3viz` for visualizations
- Planned extensions:
  - forecasting
  - spatio-temporal analysis
  - deep learning with keras
  - connector to Apache Spark

Want to contribute?

mlr3.mlr-org.com