Configuring mlr

Table of Contents

Source: `vignettes/tutorial/configureMlr.Rmd` (https://github.com/mlr-org/mlr/blob/master/vignettes/tutorial/configureMlr.Rmd)

`mlr` is designed to make usage errors due to typos or invalid parameter values as unlikely as possible. Occasionally, you might want to break those barriers and get full access, for example to reduce the amount of output on the console or to turn off checks. For all available options simply refer to the documentation of `configureMlr()` (`../../reference/configureMlr.html`). In the following we show some common use cases.

Generally, function `configureMlr()` (`../../reference/configureMlr.html`) permits to set options globally for your current **R** session.

It is also possible to set options locally.

- All options referring to the behavior of learners (these are all options except `show.info`) can be set for an individual learner via the `config` argument of `makeLearner()` (`../../reference/makeLearner.html`). The local precedes the global configuration.
- Some functions like `resample()` (`../../reference/resample.html`), `benchmark()` (`../../reference/benchmark.html`), `selectFeatures()` (`../../reference/selectFeatures.html`), `tuneParams()` (`../../reference/tuneParams.html`), and `tuneParamsMultiCrit()` (`../../reference/tuneParamsMultiCrit.html`) have a `show.info` flag that controls if progress messages are shown. The default value of `show.info` can be set by `configureMlr()` (`../../reference/configureMlr.html`).

Example: Reducing the output on the console

You are bothered by all the output on the console like in this example?

```
rdesc = makeResampleDesc (../../reference/makeResampleDesc.html)("Holdout")
r = resample (../../reference/resample.html)("classif.multinom", iris.task, rdesc
## Resampling: holdout
## Measures:             mmce
## # weights:  18 (10 variable)
## initial  value 109.861229
## iter  10 value 11.464290
## iter  20 value 0.635657
## iter  30 value 0.115392
## iter  40 value 0.069894
## iter  50 value 0.048895
## iter  60 value 0.046374
## iter  70 value 0.044837
## iter  80 value 0.034222
## iter  90 value 0.028482
## iter 100 value 0.027726
## final  value 0.027726
## stopped after 100 iterations
## [Resample] iter 1:    0.1000000
##
## Aggregated Result: mmce.test.mean=0.1000000
##
```

You can suppress the output for this Learner `makeLearner()` (`../../reference/makeLearner.html`) and this `resample()` (`../../reference/resample.html`) call as follows:

```
lrn = makeLearner (../../reference/makeLearner.html)("classif.multinom", config =
r = resample (../../reference/resample.html)(lrn, iris.task, rdesc, show.info = F
```

(Note that `nnet::multinom()` (`http://www.rdocumentation.org/packages/nnet/topics/multinom`) has a `trace` switch that can alternatively be used to turn off the progress messages.)

To globally suppress the output for all subsequent learners and calls to `resample()` (`../../reference/resample.html`), `benchmark()` (`../../reference/benchmark.html`) etc. do the following:

```
configureMlr (../../reference/configureMlr.html)(show.learner.output = FALSE, sho
r = resample (../../reference/resample.html)("classif.multinom", iris.task, rdesc
```

Accessing and resetting the configuration

Function `getMlrOptions()` (`../../reference/getMlrOptions.html`) returns a

`base::list()` (http://www.rdocumentation.org/packages/base/topics/list) with the current configuration.

```
getMlrOptions ( ../../reference/getMlrOptions.html)()
## $show.info
## [1] FALSE
##
## $on.learner.error
## [1] "stop"
##
## $on.learner.warning
## [1] "warn"
##
## $on.par.without.desc
## [1] "stop"
##
## $on.par.out.of.bounds
## [1] "stop"
##
## $on.measure.not.applicable
## [1] "stop"
##
## $show.learner.output
## [1] FALSE
##
## $on.error.dump
## [1] FALSE
```

To restore the default configuration call `configureMlr()` (`../../reference/configureMlr.html`) with an empty argument list.

```
configureMlr ( ../../reference/configureMlr.html)()
```

```
getMlrOptions (../../reference/getMlrOptions.html)()
## $show.info
## [1] TRUE
##
## $on.learner.error
## [1] "stop"
##
## $on.learner.warning
## [1] "warn"
##
## $on.par.without.desc
## [1] "stop"
##
## $on.par.out.of.bounds
## [1] "stop"
##
## $on.measure.not.applicable
## [1] "stop"
##
## $show.learner.output
## [1] TRUE
##
## $on.error.dump
## [1] FALSE
```

Example: Turning off parameter checking

It might happen that you want to set a parameter of a Learner (`makeLearner()` (`../../reference/makeLearner.html`), but the parameter is not registered in the learner's parameter set (`ParamHelpers::makeParamSet()` (`http://www.rdocumentation.org/packages/ParamHelpers/topics/makeParamSet`)) yet. In this case you might want to contact us (https://github.com/mlr-org/mlr#get-in-touch) or open an issue (https://github.com/mlr-org/mlr/issues/new) as well! But until the problem is fixed you can turn off `mlr`'s parameter checking. The parameter setting will then be passed to the underlying function without further ado.

```
# Support Vector Machine with linear kernel and new parameter 'newParam'
lrn = makeLearner(../../reference/makeLearner.html)("classif.ksvm", kernel = "va
## Error in setHyperPars2.Learner(learner, insert(par.vals, args)): classif.ksvm
## Did you mean one of these hyperparameters instead: degree scaled kernel
## You can switch off this check by using configureMlr!

# Turn off parameter checking completely
configureMlr(../../reference/configureMlr.html)(on.par.without.desc = "quiet")
lrn = makeLearner(../../reference/makeLearner.html)("classif.ksvm", kernel = "va
train(../../reference/train.html)(lrn, iris.task)
##  Setting default kernel parameters
## Model for learner.id=classif.ksvm; learner.class=classif.ksvm
## Trained on: task.id = iris-example; obs = 150; features = 4
## Hyperparameters: fit=FALSE,kernel=vanilladot,newParam=3

# Option "quiet" also masks typos
lrn = makeLearner(../../reference/makeLearner.html)("classif.ksvm", kernl = "van
train(../../reference/train.html)(lrn, iris.task)
## Model for learner.id=classif.ksvm; learner.class=classif.ksvm
## Trained on: task.id = iris-example; obs = 150; features = 4
## Hyperparameters: fit=FALSE,kernl=vanilladot

# Alternatively turn off parameter checking, but still see warnings
configureMlr(../../reference/configureMlr.html)(on.par.without.desc = "warn")
lrn = makeLearner(../../reference/makeLearner.html)("classif.ksvm", kernl = "van
## Warning in setHyperPars2.Learner(learner, insert(par.vals, args)): classif.ksv
## Did you mean one of these hyperparameters instead: kernel nu degree
## You can switch off this check by using configureMlr!
## Warning in setHyperPars2.Learner(learner, insert(par.vals, args)): classif.ksv
## Did you mean one of these hyperparameters instead: degree scaled kernel
## You can switch off this check by using configureMlr!

train(../../reference/train.html)(lrn, iris.task)
## Model for learner.id=classif.ksvm; learner.class=classif.ksvm
## Trained on: task.id = iris-example; obs = 150; features = 4
## Hyperparameters: fit=FALSE,kernl=vanilladot,newParam=3
```

Example: Handling errors in a learning method

If a learning method throws an error the default behavior of `mlr` is to generate an exception as well. However, in some situations, for example if you conduct a larger bechmark experiment (benchmark_experiments.html) with multiple data sets and learners, you usually don't want the whole experiment stopped due to one error. You can prevent this using the `on.learner.error` option of `configureMlr()` (../../reference/configureMlr.html).

```
# This call gives an error caused by the low number of observations in class "vi
train (../../reference/train.html)("classif.qda", task = iris.task, subset = 1:1(
## Error in qda.default(x, grouping, ...): some group is too small for 'qda'

# Get a warning instead of an error
configureMlr (../../reference/configureMlr.html)(on.learner.error = "warn")
mod = train (../../reference/train.html)("classif.qda", task = iris.task, subset
## Warning in train("classif.qda", task = iris.task, subset = 1:104): Could not t
##   some group is too small for 'qda'

mod
## Model for learner.id=classif.qda; learner.class=classif.qda
## Trained on: task.id = iris-example; obs = 104; features = 4
## Hyperparameters:
## Training failed: Error in qda.default(x, grouping, ...) :
##   some group is too small for 'qda'
##
## Training failed: Error in qda.default(x, grouping, ...) :
##   some group is too small for 'qda'

# mod is an object of class FailureModel
isFailureModel (../../reference/isFailureModel.html)(mod)
## [1] TRUE

# Retrieve the error message
getFailureModelMsg (../../reference/getFailureModelMsg.html)(mod)
## [1] "Error in qda.default(x, grouping, ...) : \n  some group is too small for

# predict and performance return NA's
pred = predict(mod, iris.task)
pred
## Prediction: 150 observations
## predict.type: response
## threshold:
## time: NA
##   id  truth response
## 1  1 setosa     <NA>
## 2  2 setosa     <NA>
## 3  3 setosa     <NA>
## 4  4 setosa     <NA>
## 5  5 setosa     <NA>
## 6  6 setosa     <NA>
##  ... (#rows: 150, #cols: 3)

performance (../../reference/performance.html)(pred)
## mmce
##   NA
```

If `on.learner.error = "warn"` a warning is issued instead of an exception and an object of

class `FailureModel()` (`../../reference/FailureModel.html`) is created. You can extract the error message using function `getFailureModelMsg()` (`../../reference/getFailureModelMsg.html`). All further steps like prediction and performance calculation work and return `NA's`.