Package 'automlr'

March 17, 2016

Title automatic machine learning in R

Description Package for interfacing mlr with algorithm configuration optimization libraries, giving functionality for optimizing across different ML algorithms.

URL https://github.com/mlr-org/automlr

BugReports https://github.com/mlr-org/automlr/issues

License AGPL-3

Encoding UTF-8

Depends R (>= 3.0.2), mlr (>= 2.8), ParamHelpers (>= 1.7), utils

Imports BBmisc (>= 1.9), checkmate (>= 1.7.1), parallelMap (>= 1.3)

Suggests mlrMBO (>= 1.0), smoof (>= 1.3), irace (>= 1.05), fastICA, testthat, FSelector, randomForestSRC, glmnet, stats, lqa, stepPlr, MASS, DiscriMiner, sparseLDA, elasticnet, rrlda, klaR, sda, caret, mda, sparsediscrim, kknn, class, rknn, FNN, RWeka, party, nodeHarvest, rpart, randomForest, extraTrees, ranger, rFerns, rotationForest, ada, mboost, adabag, gbm, xgboost, SwarmSVM, LiblineaR, kernlab, e1071, deepnet, nnet, neuralnet, kohonen

Roxygen list(wrap = FALSE)

LazyData yes

ByteCompile yes

Version 0.1.0

RoxygenNote 5.0.1

Collate 'amexowrapper.R' 'aminterface.R' 'autolearner.R' 'defaults.R' 'lsambackends.R' 'mlrLearners.R' 'automlr.R' 'buildLearners.R' 'convertParamSetToIrace.R' 'optDummy.R' 'optGeneric.R' 'optIRace.R' 'optMBO.R' 'optRandom.R' 'preprocess.R' 'utils.R' 'wrappers.R' 'zzz.R'

NeedsCompilation no

Author mb706 [aut, cre]

Maintainer mb706 <mb706@uni>

R topics documented:

`amaddprior` *update the environment's prior by adding the information contained in 'prior'.*

Description

This is entirely backend-dependent.

Usage

```
amaddprior(env, prior)
```

Arguments

`env`	`[environment]` The private data of this backend
`prior`	[any] A 'prior' object.

amaddprior.amdummy *Reference implementation that exemplifies the backend interface.*

Description

Counts the number of priors used.

Usage

```
## S3 method for class 'amdummy'
amaddprior(env, prior)
```

Arguments

`env`	`[environment]` The private data of this backend
`prior`	[any] A 'prior' object.

amfinish *Converte the `AMState` object as returned by `automlr` to an `AMResult` object.*

Description

The result object contains information about the solution that is relatively backend-independent.

Usage

```
amfinish(amstate)
```

Arguments

`amstate`	`[AMState]` The AMState object which is to be converted.

Value

`AMResult` Object representing the optimum found by the `automlr` run.

Object members:

learner [`Learner` **]** The (constructed) learner that achieved the optimum.

opt.point [`list` **]** Optimal hyperparameters for learner.

opt.val [`numeric` **]** Optimum reached for the `AMState`'s `measure`.

opt.path [`OptPath` **]** Information about all the evaluations performed.

result [any] Furthe backend-dependent information about the optimum.

... (further elements) Elements of the `AMState` object.

`amgetprior` *return whatever kind of object this backend uses as 'prior'.*

Description

This is called usually after `amaddprior` to retrieve an updated prior object.

Usage

```
amgetprior(env)
```

Arguments

`env`	`[environment]` The private data of this backend.

Value

any An object representing the prior.

`amgetprior.amdummy` *Reference implementation that exemplifies the backend interface.*

Description

Just give the prior object, which will be a number in this case.

Usage

```
## S3 method for class 'amdummy'
amgetprior(env)
```

Arguments

`env`	`[environment]` The private data of this backend.

Value

any An object representing the prior.

`amoptimize` *Perform optimization, respecting the given budget.*

Description

return a vector detailing the spent budget.

optimization progress should be saved to 'env'.

Usage

```
amoptimize(env, stepbudget)
```

Arguments

`env`	`[environment]` The private data of this backend.
`stepbudget`	`[numeric]` The budget for this optimization step with one or several of the entries `walltime`, `cputime` `modeltime` and `evals`. See `automlr` for details.

Value

`numeric(4)` The budget spent during this invocation.

`amoptimize.amdummy` *Reference implementation that exemplifies the backend interface.*

Description

Simulate the spending of budget.

optimization progress should be saved to 'env'.

Usage

```
## S3 method for class 'amdummy'
amoptimize(env, stepbudget)
```

Arguments

`env`	`[environment]` The private data of this backend.
`stepbudget`	`[numeric]` The budget for this optimization step with one or several of the entries `walltime`, `cputime` `modeltime` and `evals`. See `automlr` for details.

Value

`numeric(4)` The budget spent during this invocation.

`amresult` *Give information about the optimum found.*

Description

This function can do some backend bound work to generate more information about the found optimum.

Usage

```
amresult(env)
```

Arguments

`env`	`[environment]` The private data of this backend.

Value

`list` A named list which will be inserted into the result object. Required elements are:

learner [`Learner` **]** The (constructed) learner that achieved the optimum.

opt.point [`list` **]** Optimal hyperparameters for learner.

opt.val [`numeric` **]** Optimum reached for the `AMState`'s `measure`.

opt.path [`OptPath` **]** Information about all the evaluations performed.

result [any] Furthe backend-dependent information about the optimum.

Further elements are possible, but names should not collide with `AMState` / `AMResult` slot names.

`amresult.amdummy` *Reference implementation that exemplifies the backend interface.*

Description

For the dummy backend, the result will be a random point in the search space.

Usage

```
## S3 method for class 'amdummy'
amresult(env)
```

Arguments

`env`	`[environment]` The private data of this backend.

Value

`list` A named list which will be inserted into the result object. Required elements are:

learner [`Learner`] The (constructed) learner that achieved the optimum.

opt.point [`list`] Optimal hyperparameters for learner.

opt.val [`numeric`] Optimum reached for the `AMState`'s `measure`.

opt.path [`OptPath`] Information about all the evaluations performed.

result [**any**] Furthe backend-dependent information about the optimum.

Further elements are possible, but names should not collide with `AMState` / `AMResult` slot names.

`amsetup` *Set up the backend private data.*

Description

This function is called once in the lifetime of the backend private data. All arguments except 'env' will not be passed to amoptimize, so they should be saved in 'env'.

Usage

```
amsetup(env, prior, learner, task, measure)
```

Arguments

`env`	[`environment`] The private data of this backend.
`prior`	[any] The prior as passed to the `automlr` invocation.
`learner`	[`Learner`] The learner object that was built from the declared search space.
`task`	[`Task`] The task to optimize the `Learner` over.
`measure`	[`Measure`] The measure to optimize.

Value

NULL

amsetup.amdummy — *Reference implementation that exemplifies the backend interface.*

Description

The dummy backend here just saves the information necessary to give some bogus result in amresult.amdummy.

Usage

```
## S3 method for class 'amdummy'
amsetup(env, prior, learner, task, measure)
```

Arguments

env	[environment] The private data of this backend.
prior	[any] The prior as passed to the automlr invocation.
learner	[Learner] The learner object that was built from the declared search space.
task	[Task] The task to optimize the Learner over.
measure	[Measure] The measure to optimize.

Value

NULL

autolearner — *Create* Autolearner *objects*

Description

Autolearner objects wrap mlr Learner objects as well as preprocessing functions and provide additional meta information. This is used to define the automlr searchspace.

Usage

```
autolearner(learner, searchspace = list(), stacktype = "learner")
```

Arguments

`learner`	[`Learner`|list] An mlr `Learner` object for a `"learner"` stacktype, otherwise an object returned by `autoWrapper`. If a `Learner` is required, it may also be the ID of the `Learner` which postpones allocation of memory and loading of packages and may therefore be preferred.
`searchspace`	[list of `searchparam`] List of `searchparams` reqpresenting the relevant parameter search space.
`stacktype`	[`character(1)`] Describing how this object can be connected with other learners. Must be one of `"wrapper"`, `"requiredwrapper"` (e.g. feature selection) or `"learner"`.

`automlr` *Automatically choose a model with parameters to fit.*

Description

This is the main entry point of automlr.

Usage

```
automlr(task, ...)

## S3 method for class 'Task'
automlr(task, measure = NULL, budget = 0,
  searchspace = mlrLearners, prior = NULL, savefile = NULL,
  save.interval = default.save.interval, backend, verbose = FALSE, ...)

## S3 method for class 'character'
automlr(task, budget = NULL, prior = NULL,
  savefile = task, save.interval = default.save.interval,
  new.seed = FALSE, verbose = FALSE, ...)

## S3 method for class 'AMState'
automlr(task, budget = NULL, prior = NULL,
  savefile = NULL, save.interval = default.save.interval,
  new.seed = FALSE, verbose = FALSE, ...)
```

Arguments

`task`	[`Task`|`AMState`|`character(1)`] Either: The mlr `Task` object to fit a model on. Or: An `AMState` object, or a `character(1)` containing the file name of an `.rds` file containing such an object. The AMState usually contains progress of a past optimization run that was aborted by the user, a crash, or which ran out of budget.
`...`	No further arguments should be given.
`measure`	[`Measure`] The mlr `Measure` object to optimize for. If this is not given and the first argument is a `Task` object, uses the task's default measure.

`budget`	[`numeric` | list of `numeric`] A named list or named vector with one or several of the entries `walltime` time since invocation `cputime` total cpu time of optimization process `modeltime` time spent executing model fits `evals` number of model fit evaluations (Time is always given in seconds.) When any of the budget criteria is exceeded, the optimization process will halt at the next possible point and return. In the current implementation, this is only checked between model fits and evaluations, so the time budgets may be exceeded, in some cases substantially. If the `taks` argument is an `AMState` object or a `character(1)` referring to an `AMState` rds-file, this is optional and defaults to the referenced `AMState`'s budget *minus the already used up budget*. To continue an already finished run, therefore, one needs to pass a higher budget than the `$spent` slot indicates. Passing `0` will return an AMState object without performing any optimization or touching the file system.
`searchspace`	[list of `Autolearner`] Declaration of the searchspace: The mlr `Learners` to use and the parameter domains to consider for optimization. `Learners` can be chosen manually, either by creating custom `Autolearner` objects using `autolearner`, using elements of the provided `link{mlrLearners}` list, searching all implemented `Learners` by using `link{mlrLearners}` (default), or searching all `Learners` without considering preprocessing using `link{mlrLearnersNoWrap}`. From the provided list, only `Learners` that fit the task characteristics and type will be used, the others will be ignored without notification.
`prior`	[any] A black box that contains some form of knowledge about the world at large that may help speed up the optimization process. Effect of this parameter depends on the backend implementation. Currently, this is ignored by all backends.
`savefile`	[`character(1)`] Name of a file or folder in which intermediate progress will be saved. This is will prevent data getting lost in case of a crash. The data written is an `AMState` object in an `.rds` file that can be read and run with another `automlr` call to resume optimization. If `savefile` ends with a forward slash (`/`), it is assumed to refer to a directory in which a new file will be created. Otherwise it is assumed to refer to a specific file name, in which case the file will be created or overwritten *without warning*. If the `task` argument is an `AMState` object, `savefile` will *not* default to the `AMState`'s *savefile* but must be supplied again; this is to prevent accidental file overwrites. If the first argument is a character, `savefile` defaults to `amstate` and therefore offers to seamlessly continue optimization runs.
`save.interval`	[codenumeric(1)] The inteval, in seconds, in between which to save the intermediate result to `savefile`. Ignored if `savefile` is `NULL`; set to `0` to only save at the end of optimization runs.
`backend`	[`character(1)`] Refers to the back end used for optimization. Currently implemented and provided by automlr are `"random"`, `"irace"` and `"mbo"`. To list all backends, run `lsambackends`.

`verbose`	`[logical(1)]` Give detailed warnings and messages that would otherwise be suppressed.
`new.seed`	`[logical(1)]` If `TRUE`, the random seed saved in the AMState object will not be used; instead the RNG state at time of the invocation will be used. The default behaviour (`FALSE`) is to use the saved rng state so that invocations with the same AMState object give a more deterministic result (insofar as execution time does not influence behaviour). *Warning*: This is not yet tested and likely does not work with `Learners` that use external RNGs.

Value

`AMState` Object containing the result as well as info about the run. Use `amfinish` to extract the results.

Object members:

task [`Task`] The task being trained for.

measure [`Measure`] The measure for which is being ptimized.

budget [`numeric`] The budget of the current run.

spent [`numeric(4)`] The budget already spent.

searchspace [list of `Autolearner`] The `Learners` being considered for optimization.

prior [any] The prior of the current run. If the backend supports this, the prior is being updated during a run and can be given to another `automlr` invocation as an argument.

backend [`character(1)`] The backend of the optimization run.

creation.time [`numeric(1)`] The time at which the object was created.

finish.time [`numeric(1)`] The time at which the object was last touched by `automlr`; either by saving it on disk or by returning a result.

previous.versions [list of `AMState`] Backlog of previous invocations of `automlr` using this object. The objects in this list are reduced instances of `AMState`.

seed [`numeric`] The value of `.Random.seed` which to use for continuation.

Examples

```
## Not run:
library(mlr)
# almost minimal invocation. Will save progress to './iris.rds'.
automlr(iris.task, budget = c(evals = 1000), backend = "random",
  savefile = "iris")
> SOME RESULT YOU GUYS

# optimize for another 1000 evaluations, loading the 'iris.rds' savefile
# automatically and saving back to it during evaluation.
automlr("iris", budget = c(evals = 2000))
> MORE RESULTS

## End(Not run)
```

autoWrapper *Create a wrapper object for* autolearner.

Description

Build a wrapper object that can be plugged into `autolearner` to give wrapper functions to `buildLearners`.

Usage

```
autoWrapper(name, constructor, conversion)
```

Arguments

name	`[character(1)]` the name of the wrapper. Must not contain a `$` character.
constructor	`[function]` The function that will be called with the learner as a single argument and construct another `Learner`.
conversion	`[function]` A function giving information about the data conversion this wrapper performs. Takes one of `"factors"`, `"ordered"`, `"numerics"`, `"missing"` as an argument and returns a character vector containing a subset of these and optionally the empty string `""`. A function returning a set B when giving the argument A indicates that the wrapper is able to convert data from format A to any of the format B. The wrappers parameter set must then adhere to automlr.remove.XXX (for XXX being each element of B) pseudo parameters in their parameter requirements (see `makeAMExoWrapper`). *Currently, only removal of features is supported; therefore, this should only return its argument and possibly* `""`. *Otherwise, the behaviour is buggy.*

buildLearners *Take a list of autolearners and create a (big) mlr Learner object*

Description

Take a list of autolearners and create a (big) mlr Learner object

Usage

```
buildLearners(searchspace, task, verbose = FALSE)
```

Arguments

`searchspace`	[list of `Autolearner`] List of autolearners.
`task`	[`Task`] The task that the searchspace is being created. `buildLearners` respects the task type, presence of NAs and type of covariates. Currently not supported: weights, request of probability assignment instead of class.
`verbose`	[`logical(1)`] Give detailed warnings.

`isambackend` *Check if the given name is a valid automlr backend*

Description

Test whether whether all necessary functions for a backend are implemented.

Usage

```
isambackend(name)
```

Arguments

`name`	[`character(1)`] Name the name of the backend to check for.

`lsambackends` *List the backends currently implemented which can be passed to `automlr`.*

Description

This lists all the backends that can be used by `automlr`, even user defined ones.

Usage

```
lsambackends(fenv = NULL)
```

Arguments

`fenv`	[`NULL`] Should be NULL

Examples

```
lsambackends()
```

`makeAMExoWrapper` *Create an mlr learner object that smoothes out the parameter space.*

Description

So what are we doing here? The AMExoWrapper mostly handles parameter space magic. Specifically, this is:

- introduce a parameter that chooses which wrapper is used, and in which sequence
- introduce parameters that control whether the wrapper(s) are used to facilitate compatibility of learners with the data, i.e. remove missing values, remove or convert data types that are not supportet
- set some special variable values that the 'requires'-expressions of other parameters can use to specify that some parameters are only relevant in the presence of certain data types.
- remove parameters that always take on a single value (maybe dependent on a 'requires' from the search space and set them internally before calling the wrapped model.
  - bonus: if the fixed parameter name contains a suffix of .AMLRFIX#, where # is a number, it will be stripped from the parameter name. This way it is possible to define alternative "defaults" for a parameter in case of certain requirements being given.

The parameters that are introduced and exposed to the outside are:

- automlr.wrappersetup: which wrapper is used. It has the format `outermostWrapper$wrapper...$wrappe`
- automlr.remove.XXX where XXX is one of missings, factors, ordered. It controls whether one of the wrapper will be set up to remove the property in question even if the underlying learner is able to use the data type.
  - NOTE: this may or may not be sensible for the data type in question and it is possible to set the respective value to FALSE via setHyperPars() & exclude it from the search space. Who is AMExoWrapper to decide?
  - NOTE2: Only wrappers marked as "requiredwrapper" can respond to this; This is because otherwise the search space gets too confusing.
- automlr.wremoving.XXX where XXX is one of missings, factors, ordered. If more than one wrapper is present with the capability of removing XXX from the data, this parameter chooses which one wrapper is responsible for removing it.

The following parameters can be used by wrappers and learners in their `$requires`-parameter; they will be replaced here:

- automlr.remove.XXX: Only used by wrappers, may only be used if the respective `$conversion()` call with "XXX" as parameter returns a vector containing at least "numerics" or "". It indicates that this wrapper is responsible for removing the XXX type. If XXX does not occurr in the data, autmlr.remove.XXX is always FALSE.
- automlr.has.XXX: May be used by all wrappers and all learners: Indicates that the XXX type is present in the data. Care is taken e.g. that when wrapperA comes before wrapperB which comes before wrapperC, and wrapperB removes missings, that inside wrapperA (and wrapperB) the value of automlr.has.missings is TRUE, but for wrapperC (and all the learners) automlr.has.missings evaluates to FALSE.

Usage

```
makeAMExoWrapper(modelmultiplexer, wrappers, taskdesc, idRef, canHandleX,
  allLearners)
```

Arguments

`modelmultiplexer`	`[ModelMultiplexer]` A modelmultiplexer object that should have a `$searchspace` slot.
`wrappers`	`[list]` A named list of wrappers that have a `$required` element; names must not contain `$`-character. Also: `$conversion`, `$searchspace`, `$constructor`.
`taskdesc`	`[TaskDesc]` The taskDesc object of the task for which to build the learner.
`idRef`	`[list]` Object that indexes different parameter's IDs. FIXME: This is to be implemented.
`canHandleX`	`[list]` A named list that maps "missings", "factors", and "ordered" to a vector of learner names that can handle the respective data.
`allLearners`	`[character]` The list of all learner names.

Value

`AMExoWrapper` A `Learner` that incorporates the wrappers and learners suitable for mlr learners.

The slot `$searchspace` should be used as `ParamSet` to tune parameters over.

`makeModelMultiplexerParamSetEx` *Like mlr's* `makeModelMultiplexerParamSet`*, but respecting requirements.*

Description

This is necessary since `makeModelMultiplexerParamSet` does not handle parameter requirements correctly.

Usage

```
makeModelMultiplexerParamSetEx(multiplexer, modelParsets, origParamNames)
```

Arguments

`multiplexer`	`[ModelMultiplexer]` The model multiplexer to use to create the ParamSet.
`modelParsets`	`[ParamSet]` The list of param sets that are used to create the mmps.

`origParamNames`	[list of `character`] Maps the IDs of the individual `Learners` to the set of parameters that are actually used and need to be multiplexed.

Value

`ModelMultiplexer`	The `ModelMultiplexer` with repaired requirements.

`makePreprocWrapperAm` *Wrap learner with preProcess function*

Description

Fuse the learner with the automlr preProcess function.

NOTE 1:
some of these arguments are useless dependent on the format of the data: If there are no numeric columns, the `*numeric-arguments` have no effect etc.

Note 2:
setting `ppa.nzv.cutoff.*` to their maximum values would effectively remove all numeric / factor columns, therefore allowing conversion for learners that don't support some types.

Usage

```
makePreprocWrapperAm(learner, ...)
```

Arguments

`learner`	[`Learner`] The mlr learner object
`...`	[any] Additional parameters passed to `preProcess`.

`mlrLearners` *A list of learners with corresponding `par.sets` that can be searched over.*

Description

This is the complete search space as suggested by the automlr package. It includes learners and wrappers.

Usage

```
mlrLearners
```

Format

An object of class `list` of length 0.

See Also

Other searchspace: `mlrLearnersNoWrap`, `mlrLightweightNoWrap`, `mlrLightweight`, `mlrWrappers`

`mlrLearnersNoWrap` *A list of learners with corresponding* `par.sets` *that can be searched over.*

Description

This is a list of learners that the infinitely wise developer of this package collected *himself* that work well with automlr. `mlrLearnersNoWrap` does not include wrappers and therefore does not perform preprocessing and does not include meta-methods.

Usage

```
mlrLearnersNoWrap
```

Format

An object of class `list` of length 58.

See Also

Other searchspace: `mlrLearners`, `mlrLightweightNoWrap`, `mlrLightweight`, `mlrWrappers`

`mlrLightweight` *A list of fast learners with corresponding* `par.sets` *that can be earched over.*

Description

This is a list similar to `mlrLearners`, only it excludes the slowest learners. This decreases average evaluation time significantly and also makes the search space small enough for the `mbo` backend.

Usage

```
mlrLightweight
```

Format

An object of class `list` of length 0.

See Also

Other searchspace: `mlrLearnersNoWrap`, `mlrLearners`, `mlrLightweightNoWrap`, `mlrWrappers`

`mlrLightweightNoWrap` *A list of fast learners with corresponding `par.set`s that can be earched over.*

Description

This is a list similar to `mlrLearnersNoWrap`, only it excludes the slowest learners. This decreases average evaluation time significantly and also makes the search space small enough for the `mbo` backend.

Usage

```
mlrLightweightNoWrap
```

Format

An object of class `list` of length 49.

See Also

Other searchspace: `mlrLearnersNoWrap`, `mlrLearners`, `mlrLightweight`, `mlrWrappers`

`mlrWrappers` *A list of wrappers with corresponding `par.set`s that can be searched over.*

Description

This is a list of wrappers that can be used as part of a searchspace. Currently this only includes a preprocessing wrapper, but may in future also include some meta-methods.

Usage

```
mlrWrappers
```

Format

An object of class `list` of length 0.

See Also

Other searchspace: `mlrLearnersNoWrap`, `mlrLearners`, `mlrLightweightNoWrap`, `mlrLightweight`

`myCheckLearner` *Turn learner id string into learner object, if necessary*

Description

If the given learner is a `character(1)` identifying a learner, the corresponding learner is generated using `makeLearner`.

Usage

```
myCheckLearner(learner)
```

Arguments

`learner`	`[character(1)|Learner]` A character scalar or learner object.

`preProcess` *preprocess the given data set*

Description

Do a set of transformation on the dataset depending on given parameters.

Usage

```
preProcess(data, target = NULL, nzv.cutoff.numeric = 0,
  nzv.cutoff.factor = 0, univariate.trafo = "off", impute.numeric = "off",
  impute.factor = "off", multivariate.trafo = "off",
  feature.filter = "off", feature.filter.thresh = 0, keep.data = FALSE)
```

Arguments

`data`	`[data.frame]` The dataset
`target`	`[character]` The name(s) of the target column(s) of the dataset; may be NULL if there is no data column.
`nzv.cutoff.numeric`	`[numeric]` Exclude numeric columns if their variance goes below this threshold.
`nzv.cutoff.factor`	`[numeric]` Exclude factorial columns if the frequency of its most frequent level is above 1-`nzv.cutoff.factor`.
`univariate.trafo`	`[character(1)]` The transformation to perform on numeric columns. Must be one of `"off"` (no trafo), `"center"`, `"scale"`, `"centerscale"`, `"range"` (scaling so that all values lie between `0` and `1`.

`impute.numeric`	`[character(1)]` If and how to impute numeric missing values. Must be one of `"off"`, `"remove.na"` (deleting rows with NAs), `"mean"`, `"median"`, `"hist"`.
`impute.factor`	`[character(1)]` If and how to impute factorial missing values. Must be one of `"off"`, `"remove.na"`, `"distinct"` (introduce new factor level), `"mode"`, `"hist"`.
`multivariate.trafo`	`[character(1)]` The multivariate transformation of numeric parameters to perform. Must be one of `"off"`, `"pca"`, `"ica"`.
`feature.filter`	`[character(1)]` How to do feature filtering. Must be one of `"off"`, `"information.gain"`, `"chi.squared"`, `"rf.importance"`.
`feature.filter.thresh`	`[numeric]` The threshold at which to exclude features when performing feature filtering. Has no effect if `feature.filter` is set to `"off"`.
`keep.data`	`[logical(1)]` Whether to include the transformed data in the returned object.

Value

`ampreproc` An object that can be used with `predict` to transform new data. If `keep.data` is `TRUE`, the slot `$data` will contain the transformed data.

`print.AMObject` *Give some cute info about a given AMState*

Description

Optionally give a little or a lot (if `verbose == TRUE`) of info.

Usage

```
## S3 method for class 'AMObject'
print(x, verbose = FALSE, ...)
```

Arguments

`x`	`[AMState|AMResult]` What to print
`verbose`	`[logical(1)]` Print detailed info
`...`	ignored

sp — *Define the searchspace parameter in a short form*

Description

This function is used to define the search space related to a given learner. The priority here is that the function should be saving space: Much of the information about a parameter is inferred from the learner itself; Only the name, the type, and a range of a parameter are needed.

However, to get notice about changes in the mlr package, also all the parameters that are not given should be referenced with an `sp()` of type `"def"`; otherwise, a warning will be given upon instantiation of the learner.

Usage

```
sp(name, type = "real", values = NULL, trafo = NULL, id = NULL,
  special = NULL, req = NULL, dim = 1)
```

Arguments

`name`	`[character(1)]` The name of the parameter which must match the id of the `Param` it refers to. May be suffixed with `.AMLRFIX#`, where `#` is a number, to expose one varible to the outside with different search space depending on requirements.
`type`	`[character(1)]` The type of the parameter. One of: `real` (numeric), `int` (integer), `cat` (discrete), `bool` (logical), `fix` (fixed value of whatever type), `def` (using default value / not setting the value).
`values`	`[numeric|character]` If `type` is `real` or `int`, this gives the lower and upper bound. If `type` is `cat`, it is a character vector of possible values. If `type` is `fix`, only one value (the one to be fixed) must be given. If `type` is codebool or `def`, it must be `NULL`.
`trafo`	`[character|function]` May be "exp" for exponential transformation. Transforms integer parameters in a smart way. Only applies for `real` and `int` values. May also be an R function.
`id`	`[character(1)]` May be given to identify parameters of different learners having the same function. This currently has no effect beyond warnings if the `id` is only given once in a search space.
`special`	`[character(1)]` May be `NULL`, `"dummy"` or `"inject"`. Set this to `"dummy"` if this is a dummy variable that will be hidden from the learner itself but visible to the outside. Set this to `"inject"` to create the parameter in the learners `ParamSet` if it does not exist.
`req`	`[language]` A requirement for the variable to have effect.
`dim`	`[integer(1)]` The number of dimensions of this variable.

Index