Package 'mlr3pipelines'

April 6, 2020

Title Preprocessing Operators and Pipelines for 'mlr3'

Version 0.1.3

Description Dataflow programming toolkit that enriches 'mlr3' with a diverse set of pipelining operators ('PipeOps') that can be composed into graphs. Operations exist for data preprocessing, model fitting, and ensemble learning. Graphs can themselves be treated as 'mlr3' 'Learners' and can therefore be resampled, benchmarked, and tuned.

License LGPL-3

URL https://mlr3pipelines.mlr-org.com, https://github.com/mlr-org/mlr3pipelines

BugReports https://github.com/mlr-org/mlr3pipelines/issues

Depends R (>= 3.1.0)

Imports backports, checkmate, data.table, digest, mlr3 (>= 0.1.4), mlr3misc (>= 0.1.4), paradox, R6, withr

Suggests ggplot2, glmnet, igraph, knitr, lgr, lme4, mlbench, mlr3filters (>= 0.1.1), mlr3learners, mlr3measures, nloptr, rmarkdown, rpart, testthat, visNetwork, bestNormalize, fastICA, kernlab, smotefamily, evaluate

ByteCompile true

Encoding UTF-8

LazyData true

NeedsCompilation no

RoxygenNote 7.0.2

Collate 'Graph.R' 'GraphLearner.R' 'mlr_pipeops.R' 'utils.R' 'PipeOp.R' 'PipeOpEnsemble.R' 'LearnerAvg.R' 'NO_OP.R' 'PipeOpTaskPreproc.R' 'PipeOpBoxCox.R' 'PipeOpBranch.R' 'PipeOpChunk.R' 'PipeOpClassBalancing.R' 'PipeOpClassWeights.R' 'PipeOpClassifAvg.R' 'PipeOpColApply.R' 'PipeOpCollapseFactors.R' 'PipeOpCopy.R' 'PipeOpEncode.R' 'PipeOpEncodeImpact.R' 'PipeOpEncodeLmer.R' 'PipeOpFeatureUnion.R' 'PipeOpFilter.R' 'PipeOpFixFactors.R'

'PipeOpHistBin.R' 'PipeOpICA.R' 'PipeOpImpute.R'
'PipeOpImputeHist.R' 'PipeOpImputeMean.R'
'PipeOpImputeMedian.R' 'PipeOpImputeNewlvl.R'
'PipeOpImputeSample.R' 'PipeOpKernelPCA.R' 'PipeOpLearner.R'
'PipeOpLearnerCV.R' 'PipeOpMissingIndicators.R'
'PipeOpModelMatrix.R' 'PipeOpMutate.R' 'PipeOpNOP.R'
'PipeOpPCA.R' 'PipeOpQuantileBin.R' 'PipeOpRegrAvg.R'
'PipeOpRemoveConstants.R' 'PipeOpScale.R' 'PipeOpScaleMaxAbs.R'
'PipeOpScaleRange.R' 'PipeOpSelect.R' 'PipeOpSmote.R'
'PipeOpSpatialSign.R' 'PipeOpSubsample.R' 'PipeOpUnbranch.R'
'PipeOpYeoJohnson.R' 'Selector.R' 'assert_graph.R'
'greplicate.R' 'gunion.R' 'operators.R' 'po.R' 'reexports.R'
'typecheck.R' 'zzz.R'

Author Martin Binder [aut, cre],
Florian Pfisterer [aut] (<https://orcid.org/0000-0001-8867-762X>),
Bernd Bischl [aut] (<https://orcid.org/0000-0001-6002-6980>),
Michel Lang [aut] (<https://orcid.org/0000-0001-9754-0393>),
Susanne Dandl [aut]

Maintainer Martin Binder <mlr.developer@mb706.com>

Repository CRAN

Date/Publication 2020-04-06 09:00:18 UTC

R topics documented:

`mlr3pipelines-package` *mlr3pipelines: Preprocessing Operators and Pipelines for 'mlr3'*

Description

Dataflow programming toolkit that enriches 'mlr3' with a diverse set of pipelining operators ('PipeOps') that can be composed into graphs. Operations exist for data preprocessing, model fitting, and ensemble learning. Graphs can themselves be treated as 'mlr3' 'Learners' and can therefore be resampled, benchmarked, and tuned.

Author(s)

Maintainer: Martin Binder <mlr.developer@mb706.com>

Authors:

- Florian Pfisterer <pfistererf@googlemail.com> (ORCID)
- Bernd Bischl <bernd_bischl@gmx.net> (ORCID)
- Michel Lang <michellang@gmail.com> (ORCID)
- Susanne Dandl <dandl.susanne@googlemail.com>

See Also

Useful links:

- https://mlr3pipelines.mlr-org.com
- https://github.com/mlr-org/mlr3pipelines
- Report bugs at https://github.com/mlr-org/mlr3pipelines/issues

---

`add_class_hierarchy_cache` *Add a Class Hierarchy to the Cache*

Description

Add a class hierarchy to the class hierarchy cache. This is necessary whenever an S3 class's class hierarchy is important when inferring compatibility between types.

Usage

```
add_class_hierarchy_cache(hierarchy)
```

Arguments

`hierarchy`	`character` the class hierarchy to add; should correspond to the `class()` of the lowest object in the hierarchy.

Value

NULL

See Also

Other class hierarchy operations: register_autoconvert_function(), reset_autoconvert_register(), reset_class_hierarchy_cache()

Examples

```
# This lets mlr3pipelines handle "data.table" as "data.frame".
# This is an example and not necessary, because mlr3pipelines adds it by default.

add_class_hierarchy_cache(c("data.table", "data.frame"))
```

---

assert_graph *Assertion for mlr3pipeline Graph*

Description

Function that checks that a given object is a Graph and throws an error if not.

Usage

```
assert_graph(x)
```

Arguments

x	(any) Object to check.

Value

Graph invisible(x)

See Also

Other Graph operators: %>>%(), as_graph(), as_pipeop(), assert_pipeop(), greplicate(), gunion()

assert_pipeop *Assertion for mlr3pipeline PipeOp*

Description

Function that checks that a given object is a PipeOp and throws an error if not.

Usage

```
assert_pipeop(x)
```

Arguments

x	(any) Object to check.

Value

PipeOp invisible(x)

See Also

Other Graph operators: %>>%(), as_graph(), as_pipeop(), assert_graph(), greplicate(), gunion()

as_graph *Conversion to mlr3pipeline Graph*

Description

The argument is turned into a Graph if possible. If clone is TRUE, a deep copy is made if the incoming object is a Graph to ensure the resulting object is a different reference from the incoming object.

as_graph() is an S3 method and can therefore be implemented by other packages that may add objects that can naturally be converted to Graphs.

By default, as_graph() tries to

- apply gunion() to x if it is a list, which recursively applies as_graph() to all list elements first
- create a Graph with only one element if x is a PipeOp or can be converted to one using as_pipeop().

Usage

```
as_graph(x, clone = FALSE)
```

Arguments

x	(any) Object to convert.
clone	(logical(1)) Whether to return a (deep copied) clone if x is a Graph.

Value

Graph x or a deep clone of it.

See Also

Other Graph operators: %>>%(), as_pipeop(), assert_graph(), assert_pipeop(), greplicate(), gunion()

---

as_pipeop *Conversion to mlr3pipeline PipeOp*

Description

The argument is turned into a PipeOp if possible. If clone is TRUE, a deep copy is made if the incoming object is a PipeOp to ensure the resulting object is a different reference from the incoming object.

as_pipeop() is an S3 method and can therefore be implemented by other packages that may add objects that can naturally be converted to PipeOps. Objects that can be converted are for example Learner (using PipeOpLearner) or Filter (using PipeOpFilter).

Usage

```
as_pipeop(x, clone = FALSE)
```

Arguments

x	(any) Object to convert.
clone	(logical(1)) Whether to return a (deep copied) clone if x is a PipeOp.

Value

PipeOp x or a deep clone of it.

See Also

Other Graph operators: %>>%(), as_graph(), assert_graph(), assert_pipeop(), greplicate(), gunion()

branch *Branch Between Alternative Paths*

Description

Create a multiplexed graph.

Usage

```
branch(..., .graphs = NULL, .prefix_branchops = "", .prefix_paths = FALSE)
```

Arguments

`...`	Multiple graphs, possibly named. They all must have exactly one output. If any of the arguments are named, then all must have unique names.
`.graphs`	([list of Graph]): Named list of Graphs, additionally to the graphs given in `...`.
`.prefix_branchops`	([character(1)]): Optional id prefix to prepend to `PipeOpBranch` and `PipeOpUnbranch` id. Their resulting IDs will be `"[.prefix_branchops]branch"` and `"[.prefix_branchops]unbranch"`. Default is `""`.
`.prefix_paths`	([logical(1) | character(1)]): Whether to add prefixes to graph IDs when performing gunion. Can be helpful to avoid ID clashes in resulting graph. Default `FALSE`. If this is `TRUE`, the prefixes are taken from the names of the input arguments if present or `"poX"` where X counts up. If this is a `character(1)`, it is a prefix that is added to the `PipeOp` IDs *additionally* to the input argument list.

Examples

```
library("mlr3")

po_pca = po("pca")
po_nop = po("nop")

branches = branch(pca = po_pca, nothing = po_nop)
# gives the same as
branches = c("pca", "nothing")
po("branch", branches) %>>%
  gunion(list(po_pca, po_nop)) %>>%
  po("unbranch", branches)

branch(pca = po_pca, nothing = po_nop,
  .prefix_branchops = "br_", .prefix_paths = "xy_")
# gives the same as
po("branch", branches, id = "br_branch") %>>%
  gunion(list(xy_pca = po_pca, xy_nothing = po_nop)) %>>%
  po("unbranch", branches, id = "br_unbranch")
```

filter_noop *Remove NO_OPs from a List*

Description

Remove all NO_OP elements from a list.

Usage

```
filter_noop(x)
```

Arguments

x	list List to filter.

Value

list: The input list, with all NO_OP elements removed.

See Also

Other Path Branching: NO_OP, is_noop(), mlr_pipeops_branch, mlr_pipeops_unbranch

Graph *Graph*

Description

A Graph is a representation of a machine learning pipeline graph. It can be *trained*, and subsequently used for *prediction*.

A Graph is most useful when used together with Learner objects encapsulated as PipeOpLearner. In this case, the Graph produces Prediction data during its $predict() phase and can be used as a Learner itself (using the GraphLearner wrapper). However, the Graph can also be used without Learner objects to simply perform preprocessing of data, and, in principle, does not even need to handle data at all but can be used for general processes with dependency structure (although the PipeOps for this would need to be written).

Format

R6Class Graph

Construction

Graph$new()

Internals

A Graph is made up of a list of PipeOps, and a data.table of edges. Both for training and prediction, the Graph performs topological sorting of the PipeOps and executes their respective $train() or $predict() functions in order, moving the PipeOp results along the edges as input to other PipeOps.

Fields

- pipeops :: named list of PipeOp
  Contains all PipeOps in the Graph, named by the PipeOp's $ids.
- edges :: data.table with columns src_id (character), src_channel (character), dst_id (character), dst_channel (character)
  Table of connections between the PipeOps. A data.table. src_id and dst_id are $ids of PipeOps that must be present in the $pipeops list. src_channel and dst_channel must respectively be $output and $input channel names of the respective PipeOps.
- is_trained :: logical(1)
  Is the Graph, i.e. are all of its PipeOps, trained, and can the Graph be used for prediction?
- lhs :: character
  Ids of the 'left-hand-side' PipeOps that have some unconnected input channels and therefore act as Graph input layer.
- rhs :: character
  Ids of the 'right-hand-side' PipeOps that have some unconnected output channels and therefore act as Graph output layer.
- input :: data.table with columns name (character), train (character), predict (character), op.id (character), channel.name (character)
  Input channels of the Graph. For each channel lists the name, input type during training, input type during prediction, PipeOp $id of the PipeOp the channel pertains to, and channel name as the PipeOp knows it.
- output :: data.table with columns name (character), train (character), predict (character), op.id (character), channel.name (character)
  Output channels of the Graph. For each channel lists the name, output type during training, output type during prediction, PipeOp $id of the PipeOp the channel pertains to, and channel name as the PipeOp knows it.
- packages :: character
  Set of all required packages for the various methods in the Graph, a set union of all required packages of all contained PipeOp objects.
- state :: named list
  Get / Set the $state of each of the members of PipeOp.
- param_set :: ParamSet
  Parameters and parameter constraints. Parameter values are in $param_set$values. These are the union of $param_sets of all PipeOps in the Graph. Parameter names as seen by the Graph have the naming scheme <PipeOp$id>.<PipeOp original parameter name>. Changing $param_set$values also propagates the changes directly to the contained PipeOps and is an alternative to changing a PipeOps $param_set$values directly.

- hash :: character(1)
  Stores a checksum calculated on the Graph configuration, which includes all PipeOp hashes (and therefore their $param_set$values) and a hash of $edges.
- keep_results :: logical(1)
  Whether to store intermediate results in the PipeOp's $.result slot, mostly for debugging purposes. Default FALSE.

Methods

- ids(sorted = FALSE)
  (logical(1)) -> character
  Get IDs of all PipeOps. This is in order that PipeOps were added if sorted is FALSE, and topologically sorted if sorted is TRUE.
- add_pipeop(op)
  (PipeOp | Learner | Filter | ...) -> self
  Mutates Graph by adding a PipeOp to the Graph. This does not add any edges, so the new PipeOp will not be connected within the Graph at first.
  Instead of supplying a PipeOp directly, an object that can naturally be converted to a PipeOp can also be supplied, e.g. a Learner or a Filter; see as_pipeop().
- add_edge(src_id,dst_id,src_channel = NULL,dst_channel = NULL)
  (character(1), character(1), character(1) | numeric(1) | NULL, character(1) | numeric(1) | NULL) -> self
  Add an edge from PipeOp src_id, and its channel src_channel (identified by its name or number as listed in the PipeOp's $output), to PipeOp dst_id's channel dst_channel (identified by its name or number as listed in the PipeOp's $input). If source or destination PipeOp have only one input / output channel and src_channel / dst_channel are therefore unambiguous, they can be omitted (i.e. left as NULL).
- plot(html)
  (logical(1)) -> NULL
  Plot the Graph, using either the **igraph** package (for html = FALSE, default) or the visNetwork package for html = TRUE producing a htmlWidget. The htmlWidget can be rescaled using visOptions.
- print()
  () -> NULL
  Print a representation of the Graph on the console. Output is a table with one row for each contained PipeOp and columns ID ($id of PipeOp), State (short representation of $state of PipeOp), sccssors (PipeOps that take their input directly from the PipeOp on this line), and prdcssors (the PipeOps that produce the data that is read as input by the PipeOp on this line).
- set_names(old,new)
  (character, character) -> self
  Rename PipeOps: Change ID of each PipeOp as identified by old to the corresponding item in new. This should be used instead of changing a PipeOp's $id value directly!
- train(input,single_input = TRUE)
  (any, logical(1)) -> named list
  Train Graph by traversing the Graphs' edges and calling all the PipeOp's $train methods in turn. Return a named list of outputs for each unconnected PipeOp out-channel, named according to the Graph's $output name column. During training, the $state member of each

PipeOps will be set and the $is_trained slot of the Graph (and each individual PipeOp) will consequently be set to TRUE.
If single_input is TRUE, the input value will be sent to each unconnected PipeOp's input channel (as listed in the Graph's $input). Typically, input should be a Task, although this is dependent on the PipeOps in the Graph. If single_input is FALSE, then input should be a list with the same length as the Graph's $input table has rows; each list item will be sent to a corresponding input channel of the Graph. If input is a named list, names must correspond to input channel names ($input$name) and inputs will be sent to the channels by name; otherwise they will be sent to the channels in order in which they are listed in $input.

- predict(input,single_input = TRUE)
(any, logical(1)) -> list of any
Predict with the Graph by calling all the PipeOp's $train methods. Input and output, as well as the function of the single_input argument, are analogous to $train().

See Also

Other mlr3pipelines backend related: PipeOpTaskPreprocSimple, PipeOpTaskPreproc, PipeOp, mlr_pipeops

Examples

```
library("mlr3")

g = Graph$new()$
  add_pipeop(PipeOpScale$new(id = "scale"))$
  add_pipeop(PipeOpPCA$new(id = "pca"))$
  add_edge("scale", "pca")
g$input
g$output

task = tsk("iris")
trained = g$train(task)
trained[[1]]$data()

task$filter(1:10)
predicted = g$predict(task)
predicted[[1]]$data()
```

---

greplicate *Create Disjoint Graph Union of Copies of a Graph*

Description

Create a new Graph containing n copies of the input Graph / PipeOp. To avoid ID collisions, PipeOp IDs are suffixed with _i where i ranges from 1 to n.

Usage

```
greplicate(graph, n)
```

Arguments

graph	Graph Graph to replicate.
n	integer(1) Number of copies to create.

Value

Graph containing n copies of input graph.

See Also

Other Graph operators: %>>%(), as_graph(), as_pipeop(), assert_graph(), assert_pipeop(), gunion()

---

gunion *Disjoint Union of Graphs*

Description

Takes an arbitrary amount of Graphs or PipeOps (or objects that can be automatically converted into Graphs or PipeOps, see as_graph() and as_pipeop()) as inputs and joins them in a new Graph.

The PipeOps of the input Graphs are not joined with new edges across Graphs, so if length(graphs) > 1 the resulting Graph will be disconnected.

Usage

```
gunion(graphs)
```

Arguments

graphs	list of (Graph | PipeOp) List of elements with one of the types defined above, which are the Graphs to be joined.

Value

Graph the resulting Graph.

See Also

Other Graph operators: %>>%(), as_graph(), as_pipeop(), assert_graph(), assert_pipeop(), greplicate()

is_noop *Test for NO_OP*

Description

Test whether a given object is a NO_OP.

Usage

```
is_noop(x)
```

Arguments

x	any Object to test.

Value

logical(1): Whether x is a NO_OP.

See Also

Other Path Branching: NO_OP, filter_noop(), mlr_pipeops_branch, mlr_pipeops_unbranch

mlr_learners_avg *Optimized Weighted Average of Features for Classification and Regression*

Description

Computes a weighted average of inputs. Used in the context of computing weighted averages of predictions.

Predictions are averaged using weights (in order of appearance in the data) which are optimized using nonlinear optimization from the package "nloptr" for a measure provided in measure (defaults to classif.acc for LearnerClassifAvg and regr.mse for LearnerRegrAvg). Learned weights can be obtained from $model. Using non-linear optimization is implemented in the SuperLearner R package. For a more detailed analysis the reader is referred to *LeDell, 2015: Scalable Ensemble Learning and Computationally Efficient Variance Estimation*.

Usage

```
mlr_learners_classif.avg

mlr_learners_regr.avg
```

Format

R6Class object inheriting from mlr3::LearnerClassif/mlr3::Learner.

Parameter Set

- measure :: character(1) | Measure
  Measure to optimized weights for. The Measure is either obtained from mlr_measures or directly supplied. Defaults to classif.acc for LearnerClassifAvg and regr.mse for LearnerRegrAvg
- algorithm :: character(1)
  Several nonlinear optimization methods from nloptr are available. See nloptr::nloptr.print.options() for a list of possible options. Note that we only allow for derivative free local or global algorithms, i.e. NLOPT_(G|L)N_.

Methods

- LearnerClassifAvg$new(), id = "classif.avg")
  (chr) -> self
  Constructor.
- LearnerRegrAvg$new(), id = "regr.avg")
  (chr) -> self
  Constructor.

See Also

Other Learners: mlr_learners_graph

Other Ensembles: PipeOpEnsemble, mlr_pipeops_classifavg, mlr_pipeops_regravg

---

mlr_learners_graph *GraphLearner*

Description

A Learner that encapsulates a Graph to be used in mlr3 resampling and benchmarks.

The Graph must return a single Prediction on its $predict() call. The result of the $train() call is discarded, only the internal state changes during training are used.

Note the predict_type of a GraphLearner does currently not track the predict_type of any Learner encapsulated within the Graph. Therefore, when requesting e.g. "prob" predictions, the predict_type of *both* the encapsulated Learner and the wrapping GraphLearner need to be set to "prob".

Format

R6Class object inheriting from mlr3::Learner.

See Also

Other Learners: mlr_learners_avg

mlr_pipeops *Dictionary of PipeOps*

Description

A simple Dictionary storing objects of class PipeOp. Each PipeOp has an associated help page, see mlr_pipeops_[id].

Format

R6Class object inheriting from mlr3misc::Dictionary.

Fields

Fields inherited from Dictionary, as well as:

- metainf :: environment
  Environment that stores the metainf argument of the $add() method. Only for internal use.

Methods

Methods inherited from Dictionary, as well as:

- add(key,value,metainf = NULL)
  (character(1), R6ClassGenerator, NULL | list)
  Adds constructor value to the dictionary with key key, potentially overwriting a previously stored item. If metainf is not NULL (the default), it must be a list of arguments that will be given to the value constructor (i.e. value$new()) when it needs to be constructed for as.data.table PipeOp listing.

S3 methods

- as.data.table(dict)
  Dictionary -> data.table::data.table
  Returns a data.table with columns key (character), packages (character), input.num (integer), output.num (integer), input.type.train (character), input.type.predict (character), output.type.train (character), output.type.predict (character).

See Also

Other mlr3pipelines backend related: Graph, PipeOpTaskPreprocSimple, PipeOpTaskPreproc, PipeOp

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample,

mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson

Examples

```
library("mlr3")

mlr_pipeops$get("learner", lrn("classif.rpart"))

# equivalent:
po("learner", learner = lrn("classif.rpart"))

# all PipeOps currently in the dictionary:
as.data.table(mlr_pipeops)[, c("key", "input.num", "output.num", "packages")]
```

---

mlr_pipeops_boxcox *PipeOpBoxCox*

Description

Conducts a Box-Cox transformation on numeric features. The lambda parameter of the transformation is estimated during training and used for both training and prediction transformation. See bestNormalize::boxcox() for details.

Format

R6Class object inheriting from PipeOpTaskPreproc/PipeOp.

Construction

```
PipeOpBoxCox$new(id = "boxcox", param_vals = list())
```

- id :: character(1)
  Identifier of resulting object, default "boxcox".
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

Input and Output Channels

Input and output channels are inherited from PipeOpTaskPreproc.

The output is the input Task with all affected numeric features replaced by their transformed versions.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as a list of class `boxcox` for each column, which is transformed.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `standardize` :: `logical(1)`
  Whether to center and scale the transformed values to attempt a standard normal distribution. For details see `boxcox()`.
- `eps` :: `numeric(1)`
  Tolerance parameter to identify if lambda parameter is equal to zero. For details see `boxcox()`.
- `lower` :: `numeric(1)`
  Lower value for estimation of lambda parameter. For details see `boxcox()`.
- `upper` :: `numeric(1)`
  Upper value for estimation of lambda parameter. For details see `boxcox()`.

Internals

Uses the `bestNormalize::boxcox` function.

Methods

Only methods inherited from `PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("boxcox")

task$data()
pop$train(list(task))[[1]]$data()
```

```
pop$state
```

`mlr_pipeops_branch` *PipeOpBranch*

Description

Perform alternative path branching: `PipeOpBranch` has multiple output channels that connect to different paths in a `Graph`. At any time, only one of these paths will be taken for execution. At the end of the different paths, the `PipeOpUnbranch` `PipeOp` must be used to indicate the end of alternative paths.

Not to be confused with `PipeOpCopy`, the naming scheme is a bit unfortunate.

Format

`R6Class` object inheriting from `PipeOp`.

Construction

```
PipeOpBranch$new(options, id = "branch", param_vals = list())
```

- `options` :: `numeric(1)` | `character`
  If `options` is an integer number, it determines the number of output channels / options that are created, named `output1`...output<n>. The $selection parameter will then be a `ParamInt`. If `options` is a `character`, it determines the names of channels directly. The $selection parameter will then be a `ParamFct`.
- `id` :: `character(1)`
  Identifier of resulting object, default `"branch"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output

`PipeOpBranch` has one input channel named `"input"`, taking any input (`"*"`) both during training and prediction.

`PipeOpBranch` has multiple output channels depending on the `options` construction argument, named `"output1"`, `"output2"`, ... if `options` is `numeric`, and named after each `options` value if `options` is a `character`. All output channels produce the object given as input (`"*"`) or `NO_OP`, both during training and prediction.

State

The $state is left empty (`list()`).

Parameters

- `selection` :: `numeric(1)` | `character(1)`
  Selection of branching path to take. Is a `ParamInt` if the `options` parameter during construction was a `numeric(1)`, and ranges from 1 to `options`. Is a `ParamFct` if the `options` parameter was a `character` and its possible values are the `options` values. Initialized to either 1 (if the `options` construction argument is `numeric(1)`) or the first element of `options` (if it is `character`).

Internals

Alternative path branching is handled by the `PipeOp` backend. To indicate that a path should not be taken, `PipeOpBranch` returns the `NO_OP` object on its output channel. The `PipeOp` handles each `NO_OP` input by automatically returning a `NO_OP` output without calling $train_internal() or $predict_internal(), until `PipeOpUnbranch` is reached. `PipeOpUnbranch` will then take multiple inputs, all except one of which must be a `NO_OP`, and forward the only non-`NO_OP` object on its output.

Fields

Only fields inherited from `PipeOp`.

Methods

Only methods inherited from `PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Other Path Branching: `NO_OP`, `filter_noop()`, `is_noop()`, `mlr_pipeops_unbranch`

Examples

```
library("mlr3")

pca = po("pca")
nop = po("nop")
choices = c("pca", "nothing")
gr = po("branch", choices) %>>%
  gunion(list(pca, nop)) %>>%
  po("unbranch", choices)
```

```
gr$param_set$values$branch.selection = "pca"
gr$train(tsk("iris"))

gr$param_set$values$branch.selection = "nothing"
gr$train(tsk("iris"))
```

`mlr_pipeops_chunk` *PipeOpChunk*

Description

Chunks its input into `outnum` chunks. Creates `outnum` `Task`s during training, and simply passes on the input during `outnum` times during prediction.

Format

`R6Class` object inheriting from `PipeOp`.

Construction

```
PipeOpChunk$new(outnum, id = "chunk", param_vals = list())
```

- `outnum` :: `numeric(1)`
  Number of output channels, and therefore number of chunks created.
- `id` :: `character(1)`
  Identifier of resulting object, default `"chunk"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output

`PipeOpChunk` has one input channel named `"input"`, taking a `Task` both during training and prediction.

`PipeOpChunk` has multiple output channels depending on the `options` construction argument, named `"output1"`, `"output2"`, ... All output channels produce (respectively disjoint, random) subsets of the input `Task` during training, and pass on the original `Task` during prediction.

State

The $state is left empty (`list()`).

Parameters

- `shuffle` :: `logical(1)`
  Should the data be shuffled before chunking? Initialized to `TRUE`.

Internals

Uses the `mlr3misc::chunk_vector()` function.

Fields

Only fields inherited from `PipeOp`.

Methods

Only methods inherited from `PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Examples

```
library("mlr3")

task = tsk("wine")
opc = mlr_pipeops$get("chunk", 2)

# watch the row number: 89 during training (task is chunked)...
opc$train(list(task))

# ... 178 during predict (task is copied)
opc$predict(list(task))
```

---

mlr_pipeops_classbalancing

PipeOpClassBalancing

Description

Both undersamples a `Task` to keep only a fraction of the rows of the majority class, as well as oversamples (repeats data points) rows of the minority class.

Sampling happens only during training phase. Class-balancing a `Task` by sampling may be beneficial for classification with imbalanced training data.

Format

R6Class object inheriting from PipeOpTaskPreproc/PipeOp.

Construction

```
PipeOpClassBalancing$new(id = "classbalancing", param_vals = list())
```

- id :: character(1) Identifier of the resulting object, default "classbalancing"
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

Input and Output Channels

Input and output channels are inherited from PipeOpTaskPreproc. Instead of a Task, a TaskClassif is used as input and output during training and prediction.

The output during training is the input Task with added or removed rows to balance target classes. The output during prediction is the unchanged input.

State

The $state is a named list with the $state elements inherited from PipeOpTaskPreproc.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc; however, the affect_columns parameter is *not* present. Further parameters are:

- ratio :: numeric(1)
  Ratio of number of rows of classes to keep, relative to the $reference value. Initialized to 1.
- reference :: numeric(1)
  What the $ratio value is measured against. Can be "all" (mean instance count of all classes), "major" (instance count of class with most instances), "minor" (instance count of class with fewest instances), "nonmajor" (average instance count of all classes except the major one), "nonminor" (average instance count of all classes except the minor one), and "one" ($ratio determines the number of instances to have, per class). Initialized to "all".
- adjust :: numeric(1)
  Which classes to up / downsample. Can be "all" (up and downsample all to match required instance count), "major", "minor", "nonmajor", "nonminor" (see respective values for $reference), "upsample" (only upsample), and "downsample". Initialized to "all".
- shuffle :: logical(1)
  Whether to shuffle the result. Otherwise, the resulting task will have the original items that were not removed in downsampling in-order, followed by all newly sampled items ordered by target class. Initialized to TRUE.

Internals

Up / downsampling happens as follows: At first, a "target class count" is calculated, by taking the mean class count of all classes indicated by the `reference` parameter (e.g. if `reference` is `"nonmajor"`: the mean class count of all classes that are not the "major" class, i.e. the class with the most samples) and multiplying this with the value of the `ratio` parameter. If `reference` is `"one"`, then the "target class count" is just the value of `ratio` (i.e. `1 * ratio`).

Then for each class that is referenced by the `adjust` parameter (e.g. if `adjust` is `"nonminor"`: each class that is not the class with the fewest samples), `PipeOpClassBalancing` either throws out samples (downsampling), or adds additional rows that are equal to randomly chosen samples (upsampling), until the number of samples for these classes equals the "target class count".

Uses `task$filter()` to remove rows. When identical rows are added during upsampling, then the `task$row_roles$use` can *not* be used to duplicate rows because of [inaudible]; instead the `task$rbind()` function is used, and a new `data.table` is attached that contains all rows that are being duplicated exactly as many times as they are being added.

Fields

Only fields inherited from `PipeOpTaskPreproc`/`PipeOp`.

Methods

Only methods inherited from `PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Examples

```
library("mlr3")

task = tsk("spam")
opb = po("classbalancing")

# target class counts
table(task$truth())

# double the instances in the minority class (spam)
opb$param_set$values = list(ratio = 2, reference = "minor",
```

```
  adjust = "minor", shuffle = FALSE)
result = opb$train(list(task))[[1L]]
table(result$truth())

# up or downsample all classes until exactly 20 per class remain
opb$param_set$values = list(ratio = 20, reference = "one",
  adjust = "all", shuffle = FALSE)
result = opb$train(list(task))[[1]]
table(result$truth())
```

`mlr_pipeops_classifavg` *PipeOpClassifAvg*

Description

Perform (weighted) majority vote prediction from classification `Predictions` by connecting `PipeOpClassifAvg` to multiple `PipeOpLearner` outputs.

If the incoming `Learner`'s $predict_type is set to `"response"`, the prediction obtained is also a `"response"` prediction with each instance predicted to the prediction from incoming `Learners` with the highest total weight. If the `Learner`'s $predict_type is set to `"prob"`, the prediction obtained is also a `"prob"` type prediction with the probability predicted to be a weighted average of incoming predictions.

All incoming `Learner`'s $predict_type must agree.

Weights can be set as a parameter; if none are provided, defaults to equal weights for each prediction. Defaults to equal weights for each model.

Format

`R6Class` inheriting from `PipeOpEnsemble`/`PipeOp`.

Construction

```
PipeOpClassifAvg$new(innum = 0, id = "classifavg", param_vals = list())
```

- `innum` :: `numeric(1)`
  Determines the number of input channels. If `innum` is 0 (default), a vararg input channel is created that can take an arbitrary number of inputs.
- `id` :: `character(1)` Identifier of the resulting object, default `"classifavg"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpEnsemble`. Instead of a `Prediction`, a `PredictionClassif` is used as input and output during prediction.

State

The $state is left empty (`list()`).

Parameters

The parameters are the parameters inherited from the `PipeOpEnsemble`.

Internals

Inherits from `PipeOpEnsemble` by implementing the `private$weighted_avg_predictions()` method.

Fields

Only fields inherited from `PipeOpEnsemble`/`PipeOp`.

Methods

Only methods inherited from `PipeOpEnsemble`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Other Ensembles: `PipeOpEnsemble`, `mlr_learners_avg`, `mlr_pipeops_regravg`

Examples

```
library("mlr3")

# Simple Bagging
gr = greplicate(n = 5,
  po("subsample") %>>%
  po("learner", lrn("classif.rpart"))
) %>>%
  po("classifavg")

 mlr3::resample(tsk("iris"), GraphLearner$new(gr), rsmp("holdout"))
```

`mlr_pipeops_classweights` *PipeOpClassWeights*

Description

Adds a class weight column to the `Task` that different `Learner`s may be able to use for sample weighting. Sample weights are added to each sample according to the target class.

Only binary classification tasks are supported.

Caution: when constructed naively without parameter, the weights are all set to 1. The `minor_weight` parameter must be adjusted for this `PipeOp` to be useful.

Format

`R6Class` object inheriting from `PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpClassWeights$new(id = "classweights", param_vals = list())
```

- `id` :: `character(1)` Identifier of the resulting object, default `"classweights"`
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`. Instead of a `Task`, a `TaskClassif` is used as input and output during training and prediction.

The output during training is the input `Task` with added weights column according to target class. The output during prediction is the unchanged input.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`; however, the `affect_columns` parameter is *not* present. Further parameters are:

- `minor_weight` :: `numeric(1)`
  Weight given to samples of the minor class. Major class samples have weight 1. Initialized to 1.

Internals

Introduces, or overwrites, the "weights" column in the Task. However, the Learner method needs to respect weights for this to have an effect.

The newly introduced column is named .WEIGHTS; there will be a naming conflict if this column already exists and is *not* a weight column itself.

Fields

Only fields inherited from PipeOpTaskPreproc/PipeOp.

Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("spam")
opb = po("classweights")

# task weights
task$weights

# double the instances in the minority class (spam)
opb$param_set$values$minor_weight = 2
result = opb$train(list(task))[[1L]]
result$weights
```

`mlr_pipeops_colapply` *PipeOpColApply*

Description

Applies a function to each column of a task. Use the `affect_columns` parameter inherited from `PipeOpTaskPreproc` to limit the columns this function should be applied to. This can be used for simple parameter transformations or type conversions (e.g. `as.numeric`).

The same function is applied during training and prediction. One important relationship for machine learning preprocessing is that during the prediction phase, the preprocessing on each data row should be independent of other rows. Therefore, the `applicator` function should always return a vector / list where each result component only depends on the corresponding input component and not on other components. As a rule of thumb, if the function `f` generates output different from `Vectorize(f)`, it is not a function that should be used for `applicator`.

Format

`R6Class` object inheriting from `PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpColApply$new(id = "colapply", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"colapply"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with features changed according to the `applicator` parameter.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `emptydt` :: `data.table`
  An empty `data.table` with columns of names and types from *output* features after training. This is used to produce a correct type conversion during prediction, even when the input has zero length and `applicator` is therefore not called.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as:

- `applicator` :: `function`
  Function to apply to each column of the task. The return value must have the same length as the input, i.e. vectorize over the input. A typical example would be `as.numeric`. Use `Vectorize` to create a vectorizing function from any function that ordinarily only takes one element input.
  The `applicator` is not called during prediction if the input task has no rows; instead the types of affected features are changed to the result types of the `applicator` call during training. Initialized to the `identity()`-function.

Internals

PipeOpColApply can not inherit from PipeOpTaskPreprocSimple, because if `applicator` is given and the prediction data has 0 rows, then the resulting `data.table` does not know what the column types should be. Column type conformity between training and prediction is enforced by simply saving a copy of an empty `data.table` in the $state$emptydt slot.

Fields

Only fields inherited from PipeOpTaskPreproc/PipeOp.

Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
poca = po("colapply", applicator = as.character)
poca$train(list(task))[[1]]  # types are converted

# function that does not vectorize
```

```
f = function(x) {
  # we could use `ifelse` here, but that is not the point
  if (x > 1) {
    "a"
  } else {
    "b"
  }
}
poca$param_set$values$applicator = Vectorize(f)
poca$train(list(task))[[1]]$data()

# only affect Petal.* columns:
poca$param_set$values$affect_columns = selector_grep("^Petal")
poca$train(list(task))[[1]]$data()
```

`mlr_pipeops_collapsefactors` *PipeOpCollapseFactors*

Description

Collapses factors of type `factor`, `ordered`: Collapses the rarest factors in the training samples, until `target_level_count` levels remain. Levels that have prevalence above `no_collapse_above_prevalence` are retained, however. For `factor` variables, these are collapsed to the next larger level, for `ordered` variables, rare variables are collapsed to the neighbouring class, whichever has fewer samples.

Levels not seen during training are not touched during prediction; Therefore it is useful to combine this with the `PipeOpFixFactors`.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpCollapseFactors$new(id = "collapsefactors", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"collapsefactors"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with rare affected `factor` and `ordered` feature levels collapsed.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `collapse_map` :: named `list` of named `list` of `character`
  List of factor level maps. For each factor, `collapse_map` contains a named `list` that indicates what levels of the input task get mapped to what levels of the output task. If `collapse_map` has an entry `feat_1` with an entry `a = c("x","y")`, it means that levels `"x"` and `"y"` get collapsed to level `"a"` in feature `"feat_1"`.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `no_collapse_above_prevalence` :: `numeric(1)`
  Fraction of samples below which factor levels get collapsed. Default is 1, which causes all levels to be collapsed until `target_level_count` remain.
- `target_level_count` :: `integer(1)`
  Number of levels to retain. Default is 2.

Internals

Makes use of the fact that `levels(fact_var) = list(target1 = c("source1","source2"),target2 = "source2")` causes renaming of level `"source1"` and `"source2"` both to `"target1"`, and also `"source2"` to `"target2"`.

Methods

Only methods inherited from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Examples

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

`mlr_pipeops_copy` *PipeOpCopy*

Description

Copies its input `outnum` times. This PipeOp usually not needed, because copying happens automatically when one `PipeOp` is followed by multiple different `PipeOps`. However, when constructing big Graphs using the `%>>%`-operator, `PipeOpCopy` can be helpful to specify which `PipeOp` gets connected to which.

Format

`R6Class` object inheriting from `PipeOp`.

Construction

```
PipeOpEnsemble$new(outnum, id = "copy", param_vals = list())
```

- `outnum` :: `numeric(1)`
  Number of output channels, and therefore number of copies being made.
- `id` :: `character(1)`
  Identifier of resulting object, default `"copy"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

`PipeOpCopy` has one input channel named `"input"`, taking any input (`"*"`) both during training and prediction.

`PipeOpCopy` has multiple output channels depending on the `outnum` construction argument, named `"output1"`, `"output2"`, ... All output channels produce the object given as input (`"*"`).

State

The $state is left empty (`list()`).

Parameters

`PipeOpCopy` has no parameters.

Internals

Note that copies are not clones, but only reference copies. This affects R6-objects: If R6 objects are copied using PipeOpCopy, they must be cloned before

Fields

Only fields inherited from PipeOp.

Methods

Only methods inherited from PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Other Placeholder Pipeops: mlr_pipeops_nop

Examples

```
# The following copies the output of 'scale' automatically to both
# 'pca' and 'nop'
po("scale") %>>%
  gunion(list(
    po("pca"),
    po("nop")
  ))

# The following would not work: the '%>>%'-operator does not know
# which output to connect to which input
# > gunion(list(
# >   po("scale"),
# >   po("select")
# > )) %>>%
# >   gunion(list(
# >     po("pca"),
# >     po("nop"),
# >     po("imputemean")
# >   ))
# Instead, the 'copy' operator makes clear which output gets copied.
gunion(list(
  po("scale") %>>% mlr_pipeops$get("copy", outnum = 2),
  po("select")
)) %>>%
  gunion(list(
    po("pca"),
```

```
  po("nop"),
  po("imputemean")
))
```

`mlr_pipeops_encode` *PipeOpEncode*

Description

Encodes columns of type `factor`, `character` and `ordered`.

Possible encodings are `"one-hot"` encoding, as well as encoding according to `stats::contr.helmert()`, `stats::contr.poly()`, `stats::contr.sum()` and `stats::contr.treatment()`. Newly created columns are named via pattern [column-name].[x] where x is the respective factor level for `"one-hot"` and `"treatment"` encoding, and an integer sequence otherwise.

Use the `PipeOpTaskPreproc` $affect_columns functionality to only encode a subset of columns, or only encode columns of a certain type.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpEncode$new(id = "encode", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"encode"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with all affected `factor`, `character` or `ordered` parameters encoded according to the `method` parameter.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `constrasts` :: named `list` of `matrix`
  List of contrast matrices, one for each affected discrete feature. The rows of each matrix correspond to (training task) levels, the the columns to the new columns that replace the old discrete feature. See `stats::contrasts`.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as:

- method :: character(1)
  Initialized to "one-hot". One of:
  - "one-hot": create a new column for each factor level.
  - "treatment": create $n-1$ columns leaving out the first factor level of each factor variable (see stats::contr.treatment()).
  - "helmert": create columns according to Helmert contrasts (see stats::contr.helmert()).
  - "poly": create columns with contrasts based on orthogonal polynomials (see stats::contr.poly()).
  - "sum": create columns with contrasts summing to zero, (see stats::contr.sum()).

Internals

Uses the stats::contrasts functions. This is relatively inefficient for features with a large number of levels.

Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

data = data.table::data.table(x = factor(letters[1:3]), y = factor(letters[1:3]))
task = TaskClassif$new("task", data, "x")

poe = po("encode")

# poe is initialized with encoding: "one-hot"
poe$train(list(task))[[1]]$data()

# other kinds of encoding:
poe$param_set$values$method = "treatment"
```

```
poe$train(list(task))[[1]]$data()

poe$param_set$values$method = "helmert"
poe$train(list(task))[[1]]$data()

poe$param_set$values$method = "poly"
poe$train(list(task))[[1]]$data()

poe$param_set$values$method = "sum"
poe$train(list(task))[[1]]$data()
```

`mlr_pipeops_encodeimpact`

Conditional Target Value Impact Encoding

Description

Encodes columns of type `factor`, `character` and `ordered`.

Impact coding for classification Tasks converts factor levels of each (factorial) column to the difference between each target level's conditional log-likelihood given this level, and the target level's global log-likelihood.

Impact coding for regression Tasks converts factor levels of each (factorial) column to the difference between the target's conditional mean given this level, and the target's global mean.

Treats new levels during prediction like missing values.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpEncodeImpact$new(id = "encodeimpact", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"encodeimpact"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with all affected `factor`, `character` or `ordered` parameters encoded.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `impact` :: a named `list`
  A list with an element for each affected feature:
  For regression each element is a single column matrix of impact values for each level of that feature.
  For classification, it is a list with an element for each *feature level*, which is a vector giving the impact of this feature level on each *outcome level*.

Parameters

- `smoothing` :: `numeric(1)`
  A finite positive value used for smoothing. Mostly relevant for classification Tasks if a factor does not coincide with a target factor level (and would otherwise give an infinite logit value). Initialized to `1e-4`.
- `impute_zero` :: `logical(1)`
  If `TRUE`, impute missing values as impact 0; otherwise the respective impact is coded as `NA`. Default `FALSE`.

Internals

Uses laplace smoothing, mostly to avoid infinite values for classification Task.

Methods

Only methods inherited `PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Examples

```
library("mlr3")
poe = po("encodeimpact")

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

```
  data.table::data.table(
    x = factor(c("a", "a", "a", "b", "b")),
    y = factor(c("a", "a", "b", "b", "b"))),
  "x")

poe$train(list(task))[[1]]$data()

poe$state
```

`mlr_pipeops_encodelmer` *Impact Encoding with Random Intercept Models*

Description

Encodes columns of type `factor`, `character` and `ordered`.

PipeOpEncodeLmer() converts factor levels of each factorial column to the estimated coefficients of a simple random intercept model. Models are fitted with the glmer function of the lme4 package and are of the type `target ~ 1 + (1 | factor)`. If the task is a regression task, the numeric target variable is used as dependent variable and the factor is used for grouping. If the task is a classification task, the target variable is used as dependent variable and the factor is used for grouping. If the target variable is multiclass, for each level of the multiclass target variable, binary "one vs. rest" models are fitted.

For training, multiple models can be estimated in a cross-validation scheme to ensure that the same factor level does not always result in identical values in the converted numerical feature. For prediction, a global model (which was fitted on all observations during training) is used for each factor. New factor levels are converted to the value of the intercept coefficient of the global model for prediction. NAs are ignored by the CPO.

Use the `PipeOpTaskPreproc` $affect_columns functionality to only encode a subset of columns, or only encode columns of a certain type.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpEncodeLmer$new(id = "encodelmer", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"encodelmer"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from PipeOpTaskPreproc.

The output is the input Task with all affected factor, character or ordered parameters encoded according to the method parameter.

State

The $state is a named list with the $state elements inherited from PipeOpTaskPreproc, as well as:

- target_levels :: character
  Levels of the target columns.
- control :: a named list
  List of coefficients learned via glmer

Parameters

- fast_optim :: logical(1)
  Initialized to TRUE. If "fast_optim" is TRUE (default), a faster (up to 50 percent) optimizer from the nloptr package is used when fitting the lmer models. This uses additional stopping criteria which can give suboptimal results.

Internals

Uses the lme4::glmer. This is relatively inefficient for features with a large number of levels.

Methods

Only methods inherited PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")
poe = po("encodelmer")
```

```
task = TaskClassif$new("task",
  data.table::data.table(
    x = factor(c("a", "a", "a", "b", "b")),
    y = factor(c("a", "a", "b", "b", "b"))),
  "x")

poe$train(list(task))[[1]]$data()

poe$state
```

`mlr_pipeops_featureunion` *PipeOpFeatureUnion*

Description

Aggregates features from all input tasks by `cbind()`ing them together into a single `Task`.

`DataBackend` primary keys and `Task` targets have to be equal across all `Task`s. Only the target column(s) of the first `Task` are kept.

If `assert_targets_equal` is `TRUE` then target column names are compared and an error is thrown if they differ across inputs.

Format

`R6Class` object inheriting from `PipeOp`.

Construction

```
PipeOpFeatureUnion$new(innum = 0, id = "featureunion", param_vals = list(), assert_targets_equal = TRUE
```

- `innum` :: `numeric(1)` | `character`
  Determines the number of input channels. If `innum` is 0 (default), a vararg input channel is created that can take an arbitrary number of inputs. If `innum` is a `character` vector, the number of input channels is the length of `innum`, and the columns of the result are prefixed with the values.
- `id` :: `character(1)` Identifier of the resulting object, default `"featureunion"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.
- `assert_targets_equal` :: `logical(1)`
  If `assert_targets_equal` is `TRUE` (Default), task target column names are checked for agreement. Disagreeing target column names are usually a bug, so this should often be left at the default.

Input and Output Channels

PipeOpFeatureUnion has multiple input channels depending on the innum construction argument, named "input1", "input2", ... if innum is nonzero; if innum is 0, there is only one *vararg* input channel named "...". All input channels take a Task both during training and prediction.

PipeOpFeatureUnion has one output channel named "output", producing a Task both during training and prediction.

The output is a Task constructed by cbind()ing all features from all input Tasks, both during training and prediction.

State

The $state is left empty (list()).

Parameters

PipeOpFeatureUnion has no Parameters.

Internals

PipeOpFeatureUnion uses the Task $cbind() method to bind the input values beyond the first input to the first Task. This means if the Tasks are database-backed, all of them except the first will be fetched into R memory for this. This behaviour may change in the future.

Fields

Only fields inherited from PipeOp.

Methods

Only methods inherited from PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
gr = gunion(list(
  po("nop"),
  po("pca")
)) %>>% po("featureunion")

gr$train(task)

po = po("featureunion", innum = c("a", "b"))

po$train(list(task, task))
```

---

mlr_pipeops_filter *PipeOpFilter*

Description

Feature filtering using a `mlr3filters::Filter` object, see the **mlr3filters** package.

If a `Filter` can only operate on a subset of columns based on column type, then only these features are considered and filtered. `nfeat` and `frac` will count for the features of the type that the `Filter` can operate on; this means e.g. that setting `nfeat` to 0 will only remove features of the type that the `Filter` can work with.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpFilter$new(filter, id = filter$id, param_vals = list())
```

- `filter` :: `Filter`
  `Filter` used for feature filtering.
- `id` :: `character(1)` Identifier of the resulting object, defaulting to the `id` of the `Filter` being used.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with features removed that were filtered out.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `scores` :: named `numeric`
  Scores calculated for all features of the training `Task` which are being used as cutoff for feature filtering. If `frac` or `nfeat` is given, the underlying `Filter` may choose to not calculate scores for all features that are given. This only includes features on which the `Filter` can operate; e.g. if the `Filter` can only operate on numeric features, then scores for factorial features will not be given.
- `features` :: `character`
  Names of features that are being kept. Features of types that the `Filter` can not operate on are always being kept.

Parameters

The parameters are the parameters inherited from the `PipeOpTaskPreproc`, as well as the parameters of the `Filter` used by this object. Besides, parameters introduced are:

- `filter.nfeat` :: `numeric(1)`
  Number of features to select. Mutually exclusive with `frac` and `cutoff`.
- `filter.frac` :: `numeric(1)`
  Fraction of features to keep. Mutually exclusive with `nfeat` and `cutoff`.
- `filter.cutoff` :: `numeric(1)`
  Minimum value of filter heuristic for which to keep features. Mutually exclusive with `nfeat` and `frac`.

Note that at least one of `filter.nfeat`, `filter.frac`, or `filter.cutoff` must be given.

Internals

This does *not* use the $select_cols feature of `PipeOpTaskPreproc` to select only features compatible with the `Filter`; instead the whole `Task` is used by $get_state() and subset internally.

Fields

Fields inherited from `PipeOpTaskPreproc`, as well as:

- `filter` :: `Filter`
  `Filter` that is being used for feature filtering. Do *not* use this slot to get to the feature filtering scores after training; instead, use $state$scores. Read-only.

Methods

Methods inherited from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")
library("mlr3filters")

# setup PipeOpFilter to keep the 5 most important
# features of the spam task w.r.t. their AUC
task = tsk("spam")
filter = flt("auc")
po = po("filter", filter = filter)
po$param_set
po$param_set$values$filter.nfeat = 5

# filter the task
filtered_task = po$train(list(task))[[1]]

# filtered task + extracted AUC scores
filtered_task$feature_names
head(po$state$scores, 10)

# feature selection embedded in a 3-fold cross validation
# keep 30% of features based on their AUC score
task = tsk("spam")
gr = po("filter", filter = flt("auc"), filter.frac = 0.5) %>>%
  po("learner", lrn("classif.rpart"))
learner = GraphLearner$new(gr)
rr = resample(task, learner, rsmp("holdout"), store_models = TRUE)
rr$learners[[1]]$model$auc$scores
```

---

mlr_pipeops_fixfactors

PipeOpFixFactors

Description

Fixes factors of type `factor`, `ordered`: Makes sure the factor levels during prediction are the same as during training; possibly dropping empty training factor levels before.

Note this may introduce *missing values* during prediction if unseen factor levels are found.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpFixFactors$new(id = "fixfactors", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"fixfactors"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with all affected `factor` and `ordered` feature levels fixed.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `levels` :: named `list` of `character`
  List of factor levels of each affected `factor` or `ordered` feature that will be fixed.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `droplevels` :: `logical(1)`
  Whether to drop empty factor levels of the training task. Default `TRUE`

Internals

Changes factor levels of columns and attaches them with a new `data.table` backend and the virtual `cbind()` backend.

Methods

Only methods inherited from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

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

`mlr_pipeops_histbin` *PipeOpHistBin*

Description

Splits numeric features into equally spaced bins. See `graphics::hist()` for details.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpHistBin$new(id = "histbin", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"histbin"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with all affected numeric features replaced by their binned versions.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `bins` :: `list`
  List of intervals representing the bins for each numeric feature.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `bins` :: `character(1)` | `numeric` | `function`
  Either a `character(1)` string naming an algorithm to compute the number of cells, a `numeric(1)` giving the number of breaks for the histogram, a vector `numeric` giving the breakpoints between the histogram cells, or a `function` to compute the vector of breakpoints or to compute the number of cells. Default is algorithm `"Sturges"` (see `grDevices::nclass.Sturges()`). For details see `hist()`.

Internals

Uses the `graphics::hist` function.

Methods

Only methods inherited from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("histbin")

task$data()
pop$train(list(task))[[1]]$data()

pop$state
```

`mlr_pipeops_ica` *PipeOpICA*

Description

Extracts statistically independent components from data. Only affects numerical features. See fastICA::fastICA for details.

Format

`R6Class` object inheriting from `PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpICA$new(id = "ica", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"ica"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with all affected numeric parameters replaced by independent components.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as the elements of the function `fastICA::fastICA()`, with the exception of the $X and $S slots. These are in particular:

- `K` :: `matrix`
  Matrix that projects data onto the first `n.comp` principal components. See `fastICA()`.
- `W` :: `matrix`
  Estimated un-mixing matrix. See `fastICA()`.
- `A` :: `matrix`
  Estimated mixing matrix. See `fastICA()`.
- `center` :: `numeric`
  The mean of each numeric feature during training.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as the following parameters based on fastICA():

- n.comp :: numeric(1)
  Number of components to extract. Default is NULL, which sets it to the number of available numeric columns.
- alg.typ:: character(1)
  Algorithm type. One of "parallel" (default) or "deflation".
- fun :: character(1)
  One of "logcosh" (default) or "exp".
- alpha :: numeric(1)
  In range [1, 2], Used for negentropy calculation when fun is "logcosh". Default is 1.0.
- method :: character(1)
  Internal calculation method. "C" (default) or "R". See fastICA().
- row.norm :: logical(1)
  Logical value indicating whether rows should be standardized beforehand. Default is FALSE.
- maxit :: numeric(1)
  Maximum number of iterations. Default is 200.
- tol :: numeric(1)
  Tolerance for convergence, default is 1e-4.
- verbose logical(1)
  Logical value indicating the level of output during the run of the algorithm. Default is FALSE.
- w.init:: matrix
  Initial un-mixing matrix. See fastICA(). Default is NULL.

Internals

Uses the fastICA() function.

Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("ica")

task$data()
pop$train(list(task))[[1]]$data()

pop$state
```

`mlr_pipeops_imputehist` *PipeOpImputeHist*

Description

Impute numerical features by histogram.

Format

`R6Class` object inheriting from `PipeOpImpute`/`PipeOp`.

Construction

```
PipeOpImputeHist$new(id = "imputehist", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"imputehist"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImputeHist`.

The output is the input `Task` with all affected numeric features missing values imputed by (column-wise) histogram.

State

The $state is a named `list` with the $state elements inherited from `PipeOpImpute`.

The $state$model is a named `list` of `lists` containing elements $counts and $breaks.

Parameters

The parameters are the parameters inherited from `PipeOpImpute`.

Internals

Uses the `graphics::hist()` function. Features that are entirely `NA` are imputed as `0`.

Methods

Only methods inherited from `PipeOpImpute`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Other Imputation PipeOps: `PipeOpImpute`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`

Examples

```
library("mlr3")

task = tsk("pima")
task$missings()

po = po("imputehist")
new_task = po$train(list(task = task))[[1]]
new_task$missings()

po$state$model
```

---

mlr_pipeops_imputemean

PipeOpImputeMean

Description

Impute numerical features by their mean.

Format

`R6Class` object inheriting from `PipeOpImpute`/`PipeOp`.

Construction

```
PipeOpImputeMean$new(id = "imputemean", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"imputemean"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImputeMean`.

The output is the input `Task` with all affected numeric features missing values imputed by (column-wise) mean.

State

The $state is a named `list` with the $state elements inherited from `PipeOpImpute`.

The $state$model is a named `list` of `numeric(1)` indicating the mean of the respective feature.

Parameters

The parameters are the parameters inherited from `PipeOpImpute`.

Internals

Uses the `mean()` function. Features that are entirely `NA` are imputed as `0`.

Methods

Only methods inherited from `PipeOpImpute`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Other Imputation PipeOps: `PipeOpImpute`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`

Examples

```
library("mlr3")

task = tsk("pima")
task$missings()

po = po("imputemean")
new_task = po$train(list(task = task))[[1]]
new_task$missings()

po$state$model
```

`mlr_pipeops_imputemedian` *PipeOpImputeMedian*

Description

Impute numerical features by their median.

Format

`R6Class` object inheriting from `PipeOpImpute`/`PipeOp`.

Construction

```
PipeOpImputeMedian$new(id = "imputemedian", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"imputemedian"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImputeMedian`.

The output is the input `Task` with all affected numeric features missing values imputed by (column-wise) median.

State

The $state is a named `list` with the $state elements inherited from `PipeOpImpute`.

The $state$model is a named `list` of `numeric(1)` indicating the median of the respective feature.

Parameters

The parameters are the parameters inherited from `PipeOpImpute`.

Internals

Uses the `stats::median()` function. Features that are entirely `NA` are imputed as `0`.

Methods

Only methods inherited from `PipeOpImpute`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Other Imputation PipeOps: `PipeOpImpute`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`

Examples

```
library("mlr3")

task = tsk("pima")
task$missings()

po = po("imputemedian")
new_task = po$train(list(task = task))[[1]]
new_task$missings()

po$state$model
```

---

mlr_pipeops_imputenewlvl

PipeOpImputeNewlvl

Description

Impute factorial features by adding a new level `".MISSING"`.

Format

`R6Class` object inheriting from `PipeOpImpute`/`PipeOp`.

Construction

```
PipeOpImputeNewlvl$new(id = "imputenewlvl", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"imputenewlvl"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImputeNewlvl`.

The output is the input `Task` with all affected factorial features missing values imputed by a new level.

State

The $state is a named `list` with the $state elements inherited from `PipeOpImpute`.

The $state$model contains only `NULL` elements.

Parameters

The parameters are the parameters inherited from `PipeOpImpute`.

Internals

Adds an explicit new `level()` to `factor` and `ordered` features, but not to `character` features.

Methods

Only methods inherited from `PipeOpImpute`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Other Imputation PipeOps: `PipeOpImpute`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputesample`

Examples

```
library("mlr3")

task = tsk("pima")
task$missings()

po = po("imputenewlvl")
new_task = po$train(list(task = task))[[1]]
new_task$missings()
```

`mlr_pipeops_imputesample` *PipeOpImputeSample*

Description

Impute features by sampling from non-missing training data.

Format

`R6Class` object inheriting from `PipeOpImpute`/`PipeOp`.

Construction

```
PipeOpImputeSample$new(id = "imputesample", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"imputesample"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImputeSample`.

The output is the input `Task` with all affected numeric features missing values imputed by values sampled (column-wise) from training data.

State

The $state is a named `list` with the $state elements inherited from `PipeOpImpute`.

The $state$model is a named `list` of training data with missings removed.

Parameters

The parameters are the parameters inherited from `PipeOpImpute`.

Internals

Uses the `sample()` function. Features that are entirely `NA` are imputed as the values given by `vector()` of their type.

Methods

Only methods inherited from `PipeOpImpute`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Other Imputation PipeOps: `PipeOpImpute`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`

Examples

```
library("mlr3")

task = tsk("pima")
task$missings()

po = po("imputesample")
new_task = po$train(list(task = task))[[1]]
new_task$missings()
```

---

mlr_pipeops_kernelpca *PipeOpKernelPCA*

Description

Extracts kernel principle components from data. Only affects numerical features. See kernlab::kpca for details.

Format

`R6Class` object inheriting from `PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpKernelPCA$new(id = "kernelpca", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"kernelpca"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with all affected numeric parameters replaced by their principal components.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as the returned `S4` object of the function `kernlab::kpca()`.

The @rotated slot of the `"kpca"` object is overwritten with an empty matrix for memory efficiency.

The slots of the `S4` object can be accessed by accessor function. See kernlab::kpca.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `kernel` :: `character(1)`
  The standard deviations of the principal components. See `kpca()`.
- `kpar` :: `list`
  List of hyper-parameters that are used with the kernel function. See `kpca()`.
- `features` :: `numeric(1)`
  Number of principal components to return. Default 0 means that all principal components are returned. See `kpca()`.
- `th` :: `numeric(1)`
  The value of eigenvalue under which principal components are ignored. Default is 0.0001. See `kpca()`.
- `na.action` :: `function`
  Function to specify NA action. Default is `na.omit`. See `kpca()`.

Internals

Uses the `kpca()` function.

Methods

Only methods inherited from `PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("kernelpca", features = 3)  # only keep top 3 components

task$data()
pop$train(list(task))[[1]]$data()
```

---

mlr_pipeops_learner *PipeOpLearner*

Description

Wraps an mlr3::Learner into a PipeOp.

Inherits the $param_set (and therefore $param_set$values) from the Learner it is constructed from.

Using PipeOpLearner, it is possible to embed mlr3::Learners into Graphs, which themselves can be turned into Learners using GraphLearner. This way, preprocessing and ensemble methods can be included into a machine learning pipeline which then can be handled as singular object for resampling, benchmarking and tuning.

Format

R6Class object inheriting from PipeOp.

Construction

```
PipeOpLearner$new(learner, id = if (is.character(learner)) learner else learner$id, param_vals = list()
```

- learner :: Learner | character(1) Learner to wrap, or a string identifying a Learner in the mlr3::mlr_learners Dictionary.
- id :: character(1) Identifier of the resulting object, defaulting to the id of the Learner being wrapped.

- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

Input and Output Channels

PipeOpLearner has one input channel named "input", taking a Task specific to the Learner type given to learner during construction; both during training and prediction.

PipeOpLearner has one output channel named "output", producing NULL during training and a Prediction subclass during prediction; this subclass is specific to the Learner type given to learner during construction.

The output during prediction is the Prediction on the prediction input data, produced by the Learner trained on the training input data.

State

The $state is set to the $state slot of the Learner object. It is a named list with members:

- model :: any
  Model created by the Learner's $train_internal() function.
- train_log :: data.table with columns class (character), msg (character)
  Errors logged during training.
- train_time :: numeric(1)
  Training time, in seconds.
- predict_log :: NULL | data.table with columns class (character), msg (character)
  Errors logged during prediction.
- predict_time :: NULL | numeric(1) Prediction time, in seconds.

Parameters

The parameters are exactly the parameters of the Learner wrapped by this object.

Internals

The $state is currently not updated by prediction, so the $state$predict_log and $state$predict_time will always be NULL.

Fields

Fields inherited from PipeOp, as well as:

- learner :: Learner
  Learner that is being wrapped. Read-only.

Methods

Methods inherited from PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Other Meta PipeOps: mlr_pipeops_learner_cv

Examples

```
library("mlr3")

task = tsk("iris")
learner = lrn("classif.rpart", cp = 0.1)
lrn_po = mlr_pipeops$get("learner", learner)

lrn_po$train(list(task))
lrn_po$predict(list(task))
```

---

mlr_pipeops_learner_cv

PipeOpLearnerCV

Description

Wraps an mlr3::Learner into a PipeOp.

Returns cross-validated predictions during training as a Task and stores a model of the Learner trained on the whole data in $state. This is used to create a similar Task during prediction.

The Task gets features depending on the capsuled Learner's $predict_type. If the Learner's $predict.type is "response", a feature <ID>.response is created, for $predict.type "prob" the <ID>.prob.<CLASS> features are created, and for $predict.type "se" the new columns are <ID>.response and <ID>.se. <ID> denotes the $id of the PipeOpLearnerCV object.

Inherits the $param_set (and therefore $param_set$values) from the Learner it is constructed from.

PipeOpLearnerCV can be used to create "stacking" or "super learning" Graphs that use the output of one Learner as feature for another Learner. Because the PipeOpLearnerCV erases the original input features, it is often useful to use PipeOpFeatureUnion to bind the prediction Task to the original input Task.

Format

R6Class object inheriting from PipeOpTaskPreproc/PipeOp.

Construction

```
PipeOpLearnerCV$new(learner, id = if (is.character(learner)) learner else learner$id, param_vals = list
```

- learner :: Learner
  Learner to use for cross validation / prediction, or a string identifying a Learner in the mlr3::mlr_learners Dictionary.
- id :: character(1) Identifier of the resulting object, defaulting to the id of the Learner being wrapped.
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

Input and Output Channels

PipeOpLearnerCV has one input channel named "input", taking a Task specific to the Learner type given to learner during construction; both during training and prediction.

PipeOpLearnerCV has one output channel named "output", producing a Task specific to the Learner type given to learner during construction; both during training and prediction.

The output is a task with the same target as the input task, with features replaced by predictions made by the Learner. During training, this prediction is the out-of-sample prediction made by resample, during prediction, this is the ordinary prediction made on the data by a Learner trained on the training phase data.

State

The $state is set to the $state slot of the Learner object, together with the $state elements inherited from the PipeOpTaskPreproc. It is a named list with the inherited members, as well as:

- model :: any
  Model created by the Learner's $train_internal() function.
- train_log :: data.table with columns class (character), msg (character)
  Errors logged during training.
- train_time :: numeric(1)
  Training time, in seconds.
- predict_log :: NULL | data.table with columns class (character), msg (character)
  Errors logged during prediction.
- predict_time :: NULL | numeric(1) Prediction time, in seconds.

Parameters

The parameters are the parameters inherited from the PipeOpTaskPreproc, as well as the parameters of the Learner wrapped by this object. Besides that, parameters introduced are:

- `resampling.method` :: `character(1)`
  Which resampling method do we want to use. Currently only supports `"cv"`.
- `resampling.folds` :: `numeric(1)`
  Number of cross validation folds. Initialized to 3.
- `keep_response` :: `logical(1)`
  Only effective during `"prob"` prediction: Whether to keep response values, if available. Initialized to `FALSE`.

Internals

The $state is currently not updated by prediction, so the $state$predict_log and $state$predict_time will always be `NULL`.

Fields

Fields inherited from `PipeOpTaskPreproc`/`PipeOp`, as well as:

- `learner` :: `Learner`
  `Learner` that is being wrapped. Read-only.

Methods

Methods inherited from `PipeOpTaskPreproc`/`PipeOp`.

See Also

Other Meta PipeOps: `mlr_pipeops_learner`

Examples

```
library("mlr3")

task = tsk("iris")
learner = lrn("classif.rpart")

lrncv_po = po("learner_cv", learner)
lrncv_po$learner$predict_type = "response"

nop = mlr_pipeops$get("nop")

graph = gunion(list(
  lrncv_po,
  nop
)) %>>% po("featureunion")

graph$train(task)

graph$pipeops$classif.rpart$learner$predict_type = "prob"

graph$train(task)
```

`mlr_pipeops_missind` *PipeOpMissInd*

Description

Add missing indicator columns ("dummy columns") to the `Task`. Drops original features; should probably be used in combination with `PipeOpFeatureUnion` and imputation `PipeOp`s (see examples).

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpMissInd$new(id = "missind", param_vals = list())
```

- `id` :: `character(1)` Identifier of the resulting object, defaulting to `"missind"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

State

$state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `indicand_cols` :: `character`
  Names of columns for which indicator columns are added. If the `which` parameter is `"all"`, this is just the names of all features, otherwise it is the names of all features that had missing values during training.

Parameters

The parameters are the parameters inherited from the `PipeOpTaskPreproc`, as well as:

- `which` :: `character(1)`
  Determines for which features the indicator columns are added. Can either be `"missing_train"` (default), adding indicator columns for each feature that actually has missing values, or `"all"`, adding indicator columns for all features.
- `type` :: `character(1)`
  Determines the type of the newly created columns. Can be one of `"numeric"`, `"factor"` (default), `"logical"`.

Internals

This PipeOp should cover most cases where "dummy columns" or "missing indicators" are desired. Some edge cases:

- If imputation for factorial features is performed and only numeric features should gain missing indicators, the `affect_columns` parameter can be set to `selector_type("numeric")`.
- If missing indicators should only be added for features that have more than a fraction of `x` missing values, the PipeOpRemoveConstants can be used with `affect_columns = selector_grep("^missing_")` and `ratio = x`.

Fields

Fields inherited from PipeOpTaskPreproc/PipeOp.

Methods

Methods inherited from PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("pima")$select(c("insulin", "triceps"))
sum(complete.cases(task$data()))
task$missings()
tail(task$data())

po = po("missind")
new_task = po$train(list(task))[[1]]

tail(new_task$data())

# proper imputation + missing indicators

impgraph = list(
  po("imputesample"),
```

```
  po("missind")
) %>>% po("featureunion")

tail(impgraph$train(task)[[1]]$data())
```

`mlr_pipeops_modelmatrix` *PipeOpModelMatrix*

Description

Transforms columns using a given `formula` using the `stats::model.matrix()` function.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpModelMatrix$new(id = "modelmatrix", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"modelmatrix"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with transformed columns according to the used `formula`.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `formula` :: `formula`
  Formula to use. Higher order interactions can be created using constructs like `~. ^ 2`. By default, an `(Intercept)` column of all `1`s is created, which can be avoided by adding `0 +` to the term. See `model.matrix()`.

Internals

Uses the `model.matrix()` function.

Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("modelmatrix", formula = ~ .  ^ 2)

task$data()
pop$train(list(task))[[1]]$data()

pop$param_set$values$formula = ~ 0 + . ^ 2

pop$train(list(task))[[1]]$data()
```

---

mlr_pipeops_mutate *PipeOpMutate*

Description

Adds features according to expressions given as formulas that may depend on values of other features. This can add new features, or can change existing features.

Format

R6Class object inheriting from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

Construction

```
PipeOpMutate$new(id = "mutate", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"mutate"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with added and/or mutated features according to the `mutation` parameter.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `mutation` :: named `list` of `formula`
  Expressions for new features to create (or present features to change), in the form of `formula`. Each element of the list is a `formula` with the name of the element naming the feature to create or change, and the formula expression determining the result. This expression may reference other features, as well as variables visible at the creation of the `formula` (see examples). Initialized to `list()`.
- `delete_originals` :: `logical(1)`
  Whether to delete original features. Even when this is `FALSE`, present features may still be overwritten. Initialized to `FALSE`.

Internals

A `formula` created using the `~` operator always contains a reference to the `environment` in which the `formula` is created. This makes it possible to use variables in the `~`-expressions that both reference either column names or variable names.

Fields

Only fields inherited from `PipeOpTaskPreproc`/`PipeOp`.

Methods

Only methods inherited from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

pom = po("mutate")
pom$param_set$values$mutation = list(
  Sepal.Area = ~ Sepal.Width * Sepal.Length,
  Petal.Area = ~ Petal.Width * Petal.Length
)

pom$train(list(tsk("iris")))[[1]]$data()
```

---

mlr_pipeops_nop *PipeOpNOP*

Description

Simply pushes the input forward. Can be useful during Graph construction using the %>>%-operator to specify which PipeOp gets connected to which.

Format

R6Class object inheriting from PipeOp.

Construction

```
PipeOpNOP$new(id = "nop", param_vals = list())
```

- id :: character(1)
  Identifier of resulting object, default "nop".
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

Input and Output Channels

PipeOpNOP has one input channel named "input", taking any input ("*") both during training and prediction.

PipeOpNOP has one output channel named "output", producing the object given as input ("*") without changes.

State

The $state is left empty (list()).

Parameters

PipeOpNOP has no parameters.

Internals

PipeOpNOP is a useful "default" stand-in for a PipeOp/Graph that does nothing.

Fields

Only fields inherited from PipeOp.

Methods

Only methods inherited from PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Other Placeholder Pipeops: mlr_pipeops_copy

Examples

```
library("mlr3")

nop = po("nop")

nop$train(list(1))
```

```
# use `gunion` and `%>>%` to create a "bypass"
# next to "pca"
gr = gunion(list(
  po("pca"),
  nop
)) %>>% po("featureunion")

gr$train(tsk("iris"))[[1]]$data()
```

`mlr_pipeops_pca` *PipeOpPCA*

Description

Extracts principle components from data. Only affects numerical features. See `stats::prcomp()` for details.

Format

`R6Class` object inheriting from `PipeOpTaskPreproc`/`PipeOp`.

Construction

`PipeOpPCA$new(id = "pca", param_vals = list())`

- `id` :: `character(1)`
  Identifier of resulting object, default `"pca"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with all affected numeric features replaced by their principal components.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as the elements of the class stats::prcomp, with the exception of the $x slot. These are in particular:

- `sdev` :: `numeric`
  The standard deviations of the principal components.
- `rotation` :: `matrix`
  The matrix of variable loadings.
- `center` :: `numeric` | `logical(1)`
  The centering used, or `FALSE`.
- `scale` :: `numeric` | `logical(1)`
  The scaling used, or `FALSE`.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as:

- center :: logical(1)
  Indicating whether the features should be centered. Default is FALSE. See prcomp().
- scale. :: logical(1)
  Whether to scale features to unit variance before analysis. Default is FALSE, but scaling is advisable. See prcomp().
- rank. :: integer(1)
  Maximal number of principal components to be used. Default is NULL: use all components. See prcomp().

Internals

Uses the prcomp() function.

Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("pca")

task$data()
pop$train(list(task))[[1]]$data()

pop$state
```

`mlr_pipeops_quantilebin` *PipeOpQuantileBin*

Description

Splits numeric features into quantile bins.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpQuantileBin$new(id = "quantilebin", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"quantilebin"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with all affected numeric features replaced by their binned versions.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `bins` :: `list`
  List of intervals representing the bins for each numeric feature.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `numsplits` :: `numeric(1)`
  Number of bins to create. Default is `2`.

Internals

Uses the `stats::quantile` function.

Methods

Only methods inherited from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("quantilebin")

task$data()
pop$train(list(task))[[1]]$data()

pop$state
```

---

mlr_pipeops_regravg *PipeOpRegrAvg*

Description

Perform (weighted) prediction averaging from regression Predictions by connecting PipeOpRegrAvg to multiple PipeOpLearner outputs.

The resulting "response" prediction is a weighted average of the incoming "response" predictions. "se" prediction is currently not aggregated but discarded if present.

Weights can be set as a parameter; if none are provided, defaults to equal weights for each prediction. Defaults to equal weights for each model.

Format

R6Class inheriting from PipeOpEnsemble/PipeOp.

Construction

```
PipeOpRegrAvg$new(innum = 0, id = "regravg", param_vals = list())
```

- innum :: numeric(1)
  Determines the number of input channels. If innum is 0 (default), a vararg input channel is created that can take an arbitrary number of inputs.

- `id` :: `character(1)` Identifier of the resulting object, default `"regravg"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpEnsemble`. Instead of a `Prediction`, a `PredictionRegr` is used as input and output during prediction.

State

The $state is left empty (`list()`).

Parameters

The parameters are the parameters inherited from the `PipeOpEnsemble`.

Internals

Inherits from `PipeOpEnsemble` by implementing the `private$weighted_avg_predictions()` method.

Fields

Only fields inherited from `PipeOpEnsemble`/`PipeOp`.

Methods

Only methods inherited from `PipeOpEnsemble`/`PipeOp`.

See Also

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTaskPreproc`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_removeconstan`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

Other Ensembles: `PipeOpEnsemble`, `mlr_learners_avg`, `mlr_pipeops_classifavg`

Examples

```
library("mlr3")

# Simple Bagging
gr = greplicate(n = 5,
  po("subsample") %>>%
  po("learner", lrn("classif.rpart"))
) %>>%
  po("classifavg")

resample(tsk("iris"), GraphLearner$new(gr), rsmp("holdout"))
```

---

mlr_pipeops_removeconstants *PipeOpRemoveConstants*

Description

Remove constant features from a mlr3::Task. For each feature, calculates the ratio of features which differ from their mode value. All features which a ratio below a settable threshold are removed from the task. Missing values can be ignored or treated as a regular value distinct from non-missing values.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpRemoveConstants$new(id = "removeconstants")
```

- `id` :: `character(1)` Identifier of the resulting object, defaulting to `"removeconstants"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

State

$state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `features` :: `character`
  Names of features that are being kept. Features of types that the `Filter` can not operate on are always being kept.

Parameters

The parameters are the parameters inherited from the PipeOpTaskPreproc, as well as:

- ratio :: numeric(1)
  Ratio of values which must be different from the mode value in order to keep a feature in the task. Default is 0, which means only constant features with exactly one observed level are removed.
- rel_tol :: numeric(1)
  Relative tolerance within which to consider a numeric feature constant. Set to 0 to disregard relative tolerance. Default is 1e-8.
- abs_tol :: numeric(1)
  Absolute tolerance within which to consider a numeric feature constant. Set to 0 to disregard absolute tolerance. Default is 1e-8.
- na_ignore :: logical(1)
  If TRUE, the ratio is calculated after removing all missing values first. Default is FALSE.

Fields

Fields inherited from PipeOpTaskPreproc/PipeOp.

Methods

Methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")
data = data.table::data.table(y = runif(10), a = 1:10, b = rep(1, 10), c = rep(1:2, each = 5))

task = TaskRegr$new("example", data, target = "y")

po = po("removeconstants")

po$train(list(task = task))[[1]]$data()

po$state
```

`mlr_pipeops_scale` *PipeOpScale*

Description

Centers all numeric features to mean = 0 (if `center` parameter is `TRUE`) and scales them by dividing them by their root-mean-square (if `scale` parameter is `TRUE`).

The root-mean-square here is defined as `sqrt(sum(x^2)/(length(x)-1))`. If the `center` parameter is `TRUE`, this corresponds to the `sd()`.

Format

`R6Class` object inheriting from `PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpScale$new(id = "scale", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"scale"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with all affected numeric parameters centered and/or scaled.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as:

- `center` :: `numeric`
  The mean of each numeric feature during training, or 0 if `center` is `FALSE`. Will be subtracted during the predict phase.
- `scale` :: `numeric`
  The root mean square, defined as `sqrt(sum(x^2)/(length(x)-1))`, of each feature during training, or 1 if `scale` is FALSE. During predict phase, features are divided by this.
  This is 1 for features that are constant during training if `center` is `TRUE`, to avoid division-by-zero.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as:

- `center` :: `logical(1)`
  Whether to center features, i.e. subtract their `mean()` from them. Default `TRUE`.
- `scale` :: `logical(1)`
  Whether to scale features, i.e. divide them by `sqrt(sum(x^2)/(length(x)-1))`. Default `TRUE`.

Internals

Uses the `scale()` function.

Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
pos = po("scale")

pos$train(list(task))[[1]]$data()

one_line_of_iris = task$filter(13)

one_line_of_iris$data()

pos$predict(list(one_line_of_iris))[[1]]$data()
```

`mlr_pipeops_scalemaxabs` *PipeOpScaleMaxAbs*

Description

Scales the numeric data columns so their maximum absolute value is `maxabs`, if possible. `NA`, `Inf` are ignored, and features that are constant 0 are not scaled.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpScaleMaxAbs$new(id = "scalemaxabs", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"scalemaxabs"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output is the input `Task` with scaled numeric features.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`, as well as the maximum absolute values of each numeric feature.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `maxabs` :: `numeric(1)`
  The maximum absolute value for each column after transformation. Default is 1.

Methods

Only methods inherited from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("scalemaxabs")

task$data()
pop$train(list(task))[[1]]$data()

pop$state
```

---

mlr_pipeops_scalerange

PipeOpScaleRange

Description

Linearly transforms numeric data columns so they are between `lower` and `upper`. The formula for this is $x' = a + x * b$, where b is $(upper - lower)/(max(x) - min(x))$ and a is $-min(x) * b + lower$.

Format

`R6Class` object inheriting from `PipeOpTaskPreprocSimple`/`PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpScaleRange$new(id = "scalerange", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"scalerange"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from PipeOpTaskPreproc.

The output is the input Task with scaled numeric features.

State

The $state is a named list with the $state elements inherited from PipeOpTaskPreproc, as well as the two transformation parameters a and b for each numeric feature.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as:

- lower :: numeric(1)
  Target value of smallest item of input data. Default is 0.
- upper :: numeric(1)
  Target value of greatest item of input data. Default is 1.

Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("scalerange", param_vals = list(lower = -1, upper = 1))

task$data()
pop$train(list(task))[[1]]$data()

pop$state
```

mlr_pipeops_select *PipeOpSelect*

Description

Removes features from Task depending on a Selector function: The selector parameter gives the features to keep. See Selector for selectors that are provided and how to write custom Selectors.

Format

R6Class object inheriting from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

Construction

```
PipeOpSelect$new(id = "select", param_vals = list())
```

- id :: character(1)
  Identifier of resulting object, default "select".
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

Input and Output Channels

Input and output channels are inherited from PipeOpTaskPreproc.

The output is the input Task with features removed that were not selected by the Selector/function in selector.

State

The $state is a named list with the $state elements inherited from PipeOpTaskPreproc, as well as:

- selection :: character
  A vector of all feature names that are kept (i.e. not dropped) in the Task. Initialized to selector_all()

Parameters

- selector :: function | Selector
  Selector function, takes a Task as argument and returns a character of features to keep. See Selector for example functions. Defaults to selector_all().

Internals

Uses task$select().

Fields

Only fields inherited from PipeOpTaskPreproc/PipeOp.

Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Other Selectors: Selector

Examples

```
library("mlr3")

task = tsk("boston_housing")
pos = po("select")

pos$param_set$values$selector = selector_all()
pos$train(list(task))[[1]]$feature_names

pos$param_set$values$selector = selector_type("factor")
pos$train(list(task))[[1]]$feature_names

pos$param_set$values$selector = selector_invert(selector_type("factor"))
pos$train(list(task))[[1]]$feature_names

pos$param_set$values$selector = selector_grep("^r")
pos$train(list(task))[[1]]$feature_names
```

`mlr_pipeops_smote` *PipeOpSmote*

Description

Generates a more balanced data set by creating synthetic instances of the minority class using the SMOTE algorithm. The algorithm samples for each minority instance a new data point based on the K nearest neighbors of that data point. It can only be applied to tasks with numeric features. See `smotefamily::SMOTE` for details.

Format

`R6Class` object inheriting from `PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpSmote$new(id = "smote", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"smote"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output during training is the input `Task` with added synthetic rows for the minority class. The output during prediction is the unchanged input.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as:

- `K` :: `numeric(1)`
  The number of nearest neighbors used for sampling new values. See `SMOTE()`.
- `dup_size` :: `numeric`
  Desired times of synthetic minority instances over the original number of majority instances. See `SMOTE()`.

Internals

For details see:
Chawla, N., Bowyer, K., Hall, L. and Kegelmeyer, W. 2002.
SMOTE: Synthetic minority oversampling technique.
Journal of Artificial Intelligence Research. 16, 321-357.

Fields

Only fields inherited from `PipeOpTaskPreproc`/`PipeOp`.

Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

# Create example task
data = smotefamily::sample_generator(1000, ratio = 0.80)
data$result = factor(data$result)
task = TaskClassif$new(id = "example", backend = data, target = "result")
task$data()
table(task$data()$result)

# Generate synthetic data for minority class
pop = po("smote")
smotedata = pop$train(list(task))[[1]]$data()
table(smotedata$result)
```

---

mlr_pipeops_spatialsign

PipeOpSpatialSign

Description

Normalizes the data row-wise. This is a natural generalization of the "sign" function to higher dimensions.

Format

R6Class object inheriting from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

Construction

```
PipeOpSpatialSign$new(id = "spatialsign", param_vals = list())
```

- id :: character(1)
  Identifier of resulting object, default "spatialsign".
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

Input and Output Channels

Input and output channels are inherited from PipeOpTaskPreproc.

The output is the input Task with all affected numeric features replaced by their normalized versions.

State

The $state is a named list with the $state elements inherited from PipeOpTaskPreproc.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as:

- length :: numeric(1)
  Length to scale rows to. Default is 1.
- norm :: numeric(1)
  Norm to use. Rows are scaled to sum(x^norm)^(1/norm) == length for finite norm, or to max(abs(x)) == length if norm is Inf. Default is 2.

Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")

task$data()

pop = po("spatialsign")

pop$train(list(task))[[1]]$data()
```

`mlr_pipeops_subsample` *PipeOpSubsample*

Description

Subsamples a `Task` to use a fraction of the rows.

Sampling happens only during training phase. Subsampling a `Task` may be beneficial for training time at possibly (depending on original `Task` size) negligible cost of predictive performance.

Format

`R6Class` object inheriting from `PipeOpTaskPreproc`/`PipeOp`.

Construction

```
PipeOpSubsample$new(id = "classbalancing", param_vals = list())
```

- `id` :: `character(1)` Identifier of the resulting object, default `"subsample"`
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output Channels

Input and output channels are inherited from `PipeOpTaskPreproc`.

The output during training is the input `Task` with added or removed rows according to the sampling. The output during prediction is the unchanged input.

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc; however, the affect_columns parameter is *not* present. Further parameters are:

- frac :: numeric(1)
  Fraction of rows in the Task to keep. May only be greater than 1 if replace is TRUE. Initialized to (1 -exp(-1)) == 0.6321.
- stratify :: logical(1)
  Should the subsamples be stratified by target? Initialized to FALSE. May only be TRUE for TaskClassif input.
- replace :: logical(1)
  Sample with replacement? Initialized to FALSE.

Internals

Uses task$filter() to remove rows. If replace is TRUE and identical rows are added, then the taskrow_rolesuse can *not* be used to duplicate rows because of [inaudible]; instead the task$rbind() function is used, and a new data.table is attached that contains all rows that are being duplicated exactly as many times as they are being added.

Fields

Only fields inherited from PipeOpTaskPreproc/PipeOp.

Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
library("mlr3")

pos = mlr_pipeops$get("subsample")

pos$train(list(tsk("iris")))
```

```
# simple bagging:
gr = greplicate(pos %>>% mlr_pipeops$get("learner", lrn("classif.rpart")), 5) %>>%
  mlr_pipeops$get("classifavg")
```

`mlr_pipeops_unbranch` *PipeOpUnbranch*

Description

Used to bring together different paths created by `PipeOpBranch`.

Format

`R6Class` object inheriting from `PipeOp`.

Construction

```
PipeOpUnbranch$new(options, id = "unbranch", param_vals = list())
```

- `options` :: `numeric(1)` | `character`
  If `options` is 0, a vararg input channel is created that can take any number of inputs. If `options` is a nonzero integer number, it determines the number of input channels / options that are created, named `input1`...input<n>. The If `options` is a `character`, it determines the names of channels directly. The difference between these three is purely cosmetic if the user chooses to produce channel names matching with the corresponding `PipeOpBranch`. However, it is not necessary to have matching names and the *vararg* option is always viable.
- `id` :: `character(1)`
  Identifier of resulting object, default `"unbranch"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

Input and Output

`PipeOpUnbranch` has multiple input channels depending on the `options` construction argument, named `"input1"`, `"input2"`, ... if `options` is a nonzero integer and named after each `options` value if `options` is a `character`; if `options` is 0, there is only one *vararg* input channel named `"..."`. All input channels take any argument (`"*"`) both during training and prediction.

`PipeOpUnbranch` has one output channel named `"output"`, producing the only `NO_OP` object received as input (`"*"`), both during training and prediction.

State

The $state is left empty (`list()`).

Parameters

PipeOpUnbranch has no parameters.

Internals

See PipeOpBranch Internals on how alternative path branching works.

Fields

Only fields inherited from PipeOp.

Methods

Only methods inherited from PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_yeojohnson, mlr_pipeops

Other Path Branching: NO_OP, filter_noop(), is_noop(), mlr_pipeops_branch

Examples

```
# See PipeOpBranch for a complete branching example
pou = po("unbranch")

pou$train(list(NO_OP, NO_OP, "hello", NO_OP, NO_OP))
```

---

mlr_pipeops_yeojohnson *PipeOpYeoJohnson*

Description

Conducts a Yeo-Johnson transformation on numeric features. It therefore estimates the optimal value of lambda for the transformation. See bestNormalize::yeojohnson() for details.

Format

R6Class object inheriting from PipeOpTaskPreproc/PipeOp.

Construction

```
PipeOpYeoJohnson$new(id = "yeojohnson", param_vals = list())
```

- id :: character(1)
  Identifier of resulting object, default "yeojohnson".
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

Input and Output Channels

Input and output channels are inherited from PipeOpTaskPreproc.

The output is the input Task with all affected numeric features replaced by their transformed versions.

State

The $state is a named list with the $state elements inherited from PipeOpTaskPreproc, as well as a list of class yeojohnson for each column, which is transformed.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as:

- eps :: numeric(1)
  Tolerance parameter to identify the lambda parameter as zero. For details see yeojohnson().
- standardize :: logical
  Whether to center and scale the transformed values to attempt a standard normal distribution. For details see yeojohnson().
- lower :: numeric(1)
  Lower value for estimation of lambda parameter. For details see yeojohnson().
- upper :: numeric(1)
  Upper value for estimation of lambda parameter. For details see yeojohnson().

Internals

Uses the bestNormalize::yeojohnson function.

Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

See Also

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops

Examples

```
library("mlr3")

task = tsk("iris")
pop = po("yeojohnson")

task$data()
pop$train(list(task))[[1]]$data()

pop$state
```

NO_OP *No-Op Sentinel Used for Alternative Branching*

Description

Special data type for no-ops. Distinct from NULL for easier debugging and distinction from unintentional NULL returns.

Usage

```
NO_OP
```

Format

R6 object.

See Also

Other Path Branching: filter_noop(), is_noop(), mlr_pipeops_branch, mlr_pipeops_unbranch

PipeOp *PipeOp*

Description

A PipeOp represents a transformation of a given "input" into a given "output", with two stages: "training" and "prediction". It can be understood as a generalized function that not only has multiple inputs, but also multiple outputs (as well as two stages). The "training" stage is used when training a machine learning pipeline or fitting a statistical model, and the "predicting" stage is then used for making predictions on new data.

To perform training, the $train() function is called which takes inputs and transforms them, while simultaneously storing information in its $state slot. For prediction, the $predict() function is called, where the $state information can be used to influence the transformation of the new data.

A PipeOp is usually used in a Graph object, a representation of a computational graph. It can have multiple **input channels**—think of these as multiple arguments to a function, for example when averaging different models—, and multiple **output channels**—a transformation may return different objects, for example different subsets of a Task. The purpose of the Graph is to connect different outputs of some PipeOps to inputs of other PipeOps.

Input and output channel information of a PipeOp is defined in the $input and $output slots; each channel has a *name*, a required type during training, and a required type during prediction. The $train() and $predict() function are called with a `list` argument that has one entry for each declared channel (with one exception, see next paragraph). The `list` is automatically type-checked for each channel against $input and then passed on to the $train_internal() or $predict_internal() functions. There the data is processed and a result `list` is created. This `list` is again type-checked for declared output types of each channel. The length and types of the result `list` is as declared in $output.

A special input channel name is `"..."`, which creates a *vararg* channel that takes arbitrarily many arguments, all of the same type. If the $input table contains an `"..."`-entry, then the input given to $train() and $predict() may be longer than the number of declared input channels.

This class is an abstract base class that all PipeOps being used in a Graph should inherit from, and is not intended to be instantiated.

Format

Abstract R6Class.

Construction

```
PipeOp$new(id, param_set = ParamSet$new(), param_vals = list(), input, output, packages = character(0))
```

- `id` :: `character(1)`
  Identifier of resulting object. See $id slot.
- `param_set` :: ParamSet | `list` of `expression`
  Parameter space description. This should be created by the subclass and given to `super$initialize()`. If this is a ParamSet, it is used as the PipeOp's ParamSet directly. Otherwise it must be a `list`

of expressions e.g. created by `alist()` that evaluate to `ParamSet`s. These `ParamSet` are combined using a `ParamSetCollection`.

- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings given in `param_set`. The subclass should have its own `param_vals` parameter and pass it on to `super$initialize()`. Default `list()`.
- input :: `data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`)
  Sets the $input slot of the resulting object; see description there.
- output :: `data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`)
  Sets the $output slot of the resulting object; see description there.
- packages :: `character`
  Set of all required packages for the `PipeOp`'s $train and $predict methods. See $packages slot. Default is `character(0)`.

Internals

`PipeOp` is an abstract class with abstract functions $train_internal() and $predict_internal(). To create a functional `PipeOp` class, these two methods must be implemented. Each of these functions receives a named `list` according to the `PipeOp`'s input channels, and must return a `list` (names are ignored) with values in the order of output channels in $output. The $train_internal() and $predict_internal() function should not be called by the user; instead, a $train() and $predict() should be used. The most convenient usage is to add the `PipeOp` to a `Graph` (possibly as singleton in that `Graph`), and using the `Graph`'s $train() / $predict() methods.

Fields

- `id` :: `character`
  ID of the `PipeOp`. IDs are user-configurable, and IDs of `PipeOp`s must be unique within a `Graph`. IDs of `PipeOp`s must not be changed once they are part of a `Graph`, instead the `Graph`'s $set_names() method should be used.
- `packages` :: `character`
  Packages required for the `PipeOp`. Functions that are not in base R should still be called using `::` (or explicitly attached using `require()`) in $train_internal() *and* $predict_internal(), but packages declared here are checked before any (possibly expensive) processing has started within a `Graph`.
- `param_set` :: `ParamSet`
  Parameters and parameter constraints. Parameter values that influence the functioning of $train and / or $predict are in the $param_set$values slot; these are automatically checked against parameter constraints in $param_set.
- `state` :: `any` | `NULL`
  Method-dependent state obtained during training step, and usually required for the prediction step. This is `NULL` if and only if the `PipeOp` has not been trained. The $state is the *only* slot that can be reliably modified during $train(), because $train_internal() may theoretically be executed in a different R-session (e.g. for parallelization).
- input :: `data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`)
  Input channels of `PipeOp`. Column `name` gives the names (and order) of values in the list given

to $train() and $predict(). Column `train` is the (S3) class that an input object must conform to during training, column `predict` is the (S3) class that an input object must conform to during prediction. Types are checked by the `PipeOp` itself and do not need to be checked by $train_internal() / $predict_internal() code.
A special name is `"..."`, which creates a *vararg* input channel that accepts a variable number of inputs.

- output :: `data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`)
Output channels of `PipeOp`, in the order in which they will be given in the list returned by $train and $predict functions. Column `train` is the (S3) class that an output object must conform to during training, column `predict` is the (S3) class that an output object must conform to during prediction. The `PipeOp` checks values returned by $train_internal() and $predict_internal() against these types specifications.
- `innum` :: `numeric(1)`
Number of input channels. This equals nrow($input).
- `outnum` :: `numeric(1)`
Number of output channels. This equals nrow($output).
- `is_trained` :: `logical(1)`
Indicate whether the `PipeOp` was already trained and can therefore be used for prediction.
- `hash` :: `character(1)`
Checksum calculated on the `PipeOp`, depending on the `PipeOp`'s `class` and the slots $id and $param_set (and therefore also $param_set$values). If a `PipeOp`'s functionality may change depending on more than these values, it should inherit the $hash active binding and calculate the hash as digest(list(super$hash, <OTHER THINGS>), algo = "xxhash64").
- `.result` :: `list`
If the `Graph`'s $keep_results flag is set to `TRUE`, then the intermediate Results of $train() and $predict() are saved to this slot, exactly as they are returned by these functions. This is mainly for debugging purposes and done, if requested, by the `Graph` backend itself; it should *not* be done explicitly by $train_internal() or $predict_internal().

Methods

- `train(input)`
(`list`) -> named `list`
Train `PipeOp` on `inputs`, transform it to output and store the learned $state. If the PipeOp is already trained, already present $state is overwritten. Input list is typechecked against the $input `train` column. Return value is a list with as many entries as $output has rows, with each entry named after the $output `name` column and class according to the $output `train` column.
- `train_internal(input)`
(named `list`) -> `list`
Abstract function that must be implemented by concrete subclasses. $train_internal() is called by $train() after typechecking. It must change the $state value to something non-`NULL` and return a list of transformed data according to the $output `train` column. Names of the returned list are ignored.
The $train_internal() method should not be called by a user; instead, the $train() method should be used which does some checking and possibly type conversion.

- `predict(input)`
  (`list`) -> named `list`
  Predict on new data in `input`, possibly using the stored $state. Input and output are specified by $input and $output in the same way as for $train(), except that the `predict` column is used for type checking.
- `predict_internal(input)`
  (named `list`) -> `list`
  Abstract function that must be implemented by concrete subclasses. $predict_internal() is called by $predict() after typechecking and works analogously to $train_internal(). Unlike $train_internal(), $predict_internal() should not modify the PipeOp in any way.
  Just as $train_internal(), $predict_internal() should not be called by a user; instead, the $predict() method should be used.
- `print()`
  () -> NULL
  Prints the PipeOps most salient information: $id, $is_trained, $param_set$values, $input and $output.

See Also

Other mlr3pipelines backend related: Graph, PipeOpTaskPreprocSimple, PipeOpTaskPreproc, mlr_pipeops

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Examples

```
# example (bogus) PipeOp that returns the sum of two numbers during $train()
# as well as a letter of the alphabet corresponding to that sum during $predict().

PipeOpSumLetter = R6::R6Class("sumletter",
  inherit = PipeOp,  # inherit from PipeOp
  public = list(
    initialize = function(id = "posum", param_vals = list()) {
      super$initialize(id, param_vals = param_vals,
        # declare "input" and "output" during construction here
        # training takes two 'numeric' and returns a 'numeric';
        # prediction takes 'NULL' and returns a 'character'.
        input = data.table::data.table(name = c("input1", "input2"),
          train = "numeric", predict = "NULL"),
        output = data.table::data.table(name = "output",
```

```
        train = "numeric", predict = "character")
    )
  },

  # PipeOp deriving classes must implement train_internal and
  # predict_internal; each taking an input list and returning
  # a list as output.
  train_internal = function(input) {
    sum = input[[1]] + input[[2]]
    self$state = sum
    list(sum)
  },

  predict_internal = function(input) {
    list(letters[self$state])
  }
 )
)
posum = PipeOpSumLetter$new()

print(posum)

posum$train(list(1, 2))
# note the name 'output' is the name of the output channel specified
# in the $output data.table.

posum$predict(list(NULL, NULL))
```

`PipeOpEnsemble` *PipeOpEnsemble*

Description

Parent class for PipeOps that aggregate predictions. Implements the $train_internal() and $predict_internal() methods necessary for a `PipeOp` and requires deriving classes to create the `private$weighted_avg_predicti` function.

Format

Abstract `R6Class` inheriting from `PipeOp`.

Construction

Note: This object is typically constructed via a derived class, e.g. `PipeOpClassifAvg` or `PipeOpRegrAvg`.

`PipeOpEnsemble$new(innum = 0, id, param_set = ParamSet$new(), param_vals = list(), packages = character`

- `innum` :: `numeric(1)`
  Determines the number of input channels. If `innum` is 0 (default), a vararg input channel is created that can take an arbitrary number of inputs.

- id :: character(1)
  Identifier of the resulting object.
- param_set :: ParamSet
  ("Hyper"-)Parameters in form of a ParamSet for the resulting PipeOp.
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().
- packages :: character
  Set of packages required for this PipeOp. These packages are loaded during $train() and $predict(), but not attached. Default character(0).
- prediction_type :: character(1)
  The predict entry of the $input and $output type specifications. Should be "Prediction" (default) or one of its subclasses, e.g. "PredictionClassif", and correspond to the type accepted by $train_internal() and $predict_internal().

Input and Output Channels

PipeOpEnsemble has multiple input channels depending on the innum construction argument, named "input1", "input2", ... if innum is nonzero; if innum is 0, there is only one *vararg* input channel named "...". All input channels take only NULL during training and take a Prediction during prediction.

PipeOpEnsemble has one output channel named "output", producing NULL during training and a Prediction during prediction.

The output during prediction is in some way a weighted averaged representation of the input.

State

The $state is left empty (list()).

Parameters

- weights :: numeric
  Relative weights of input predictions. If this has length 1, it is ignored and weighs all inputs equally. Otherwise it must have length equal to the number of connected inputs. Initialized to 1 (equal weights).

Internals

The commonality of ensemble methods using PipeOpEnsemble is that they take a NULL-input during training and save an empty $state. They can be used following a set of PipeOpLearner PipeOps to perform (possibly weighted) prediction averaging. See e.g. PipeOpClassifAvg and PipeOpRegrAvg which both inherit from this class.

Should it be necessary to use the output of preceding Learners during the "training" phase, then PipeOpEnsemble should not be used. In fact, if training time behaviour of a Learner is important, then one should use a PipeOpLearnerCV instead of a PipeOpLearner, and the ensemble can be created with a Learner encapsulated by a PipeOpLearner. See LearnerClassifAvg and LearnerRegrAvg for examples.

Fields

Only fields inherited from PipeOp.

Methods

Methods inherited from PipeOp as well as:

- weighted_avg_prediction(inputs,weights,row_ids,truth)
  (list of Prediction, numeric, integer | character, list) -> NULL
  Create Predictions that correspond to the weighted average of incoming Predictions. This is called by $predict_internal() with cleaned and sanity-checked values: inputs are guaranteed to fit together, row_ids and truth are guaranteed to be the same as each one in inputs, and weights is guaranteed to have the same length as inputs.
  This method is abstract, it must be implemented by deriving classes.

See Also

Other PipeOps: PipeOpImpute, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Other Ensembles: mlr_learners_avg, mlr_pipeops_classifavg, mlr_pipeops_regravg

---

PipeOpImpute *PipeOpImpute*

Description

Abstract base class for feature imputation.

Format

Abstract R6Class object inheriting from PipeOp.

Construction

```
PipeOpImpute$$new(id, param_set = ParamSet$new(), param_vals = list(), whole_task_dependent = FALSE, pa
```

- id :: character(1)
  Identifier of resulting object. See $id slot of PipeOp.

- param_set :: ParamSet
  Parameter space description. This should be created by the subclass and given to super$initialize().
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings given in param_set. The subclass should have its own param_vals parameter and pass it on to super$initialize(). Default list().
- whole_task_dependent :: logical(1)
  Whether the context_columns parameter should be added which lets the user limit the columns that are used for imputation inference. This should generally be FALSE if imputation depends only on individual features (e.g. mode imputation), and TRUE if imputation depends on other features as well (e.g. kNN-imputation).
- packages :: character
  Set of all required packages for the PipeOp's $train and $predict methods. See $packages slot. Default is character(0).
- task_type :: character(1)
  The class of Task that should be accepted as input and will be returned as output. This should generally be a character(1) identifying a type of Task, e.g. "Task", "TaskClassif" or "TaskRegr" (or another subclass introduced by other packages). Default is "Task".

Input and Output Channels

PipeOpImpute has one input channel named "input", taking a Task, or a subclass of Task if the task_type construction argument is given as such; both during training and prediction.

PipeOpImpute has one output channel named "output", producing a Task, or a subclass; the Task type is the same as for input; both during training and prediction.

The output Task is the modified input Task with features imputed according to the $impute() function.

State

The $state is a named list; besides members added by inheriting classes, the members are:

- affect_cols :: character
  Names of features being selected by the affect_columns parameter.
- inference_cols :: character
  Names of features being selected by the context_columns parameter.
- intasklayout :: data.table
  Copy of the training Task's $feature_types slot. This is used during prediction to ensure that the prediction Task has the same features, feature layout, and feature types as during training.
- outtasklayout :: data.table
  Copy of the trained Task's $feature_types slot. This is used during prediction to ensure that the Task resulting from the prediction operation has the same features, feature layout, and feature types as after training.
- model :: named list
  Model used for imputation. This is a list named by Task features, containing the result of the $train_imputer() function for each one.

Parameters

- affect_columns :: function | Selector | NULL
  What columns the PipeOpImpute should operate on. The parameter must be a Selector function, which takes a Task as argument and returns a character of features to use.
  See Selector for example functions. Defaults to NULL, which selects all features.
- context_columns :: function | Selector | NULL
  What columns the PipeOpImpute imputation may depend on. This parameter is only present if the constructor is called with the whole_task_dependent argument set to TRUE.
  The parameter must be a Selector function, which takes a Task as argument and returns a character of features to use.
  See Selector for example functions. Defaults to NULL, which selects all features.

Internals

PipeOpImpute is an abstract class inheriting from PipeOp that makes implementing imputer PipeOps simple.

Fields

Fields inherited from PipeOp.

Methods

Methods inherited from PipeOp, as well as:

- select_cols(task)
  (Task) -> character
  Selects which columns the PipeOp operates on. In contrast to the affect_columns parameter. select_cols is for the *inheriting class* to determine which columns the operator should function on, e.g. based on feature type, while affect_columns is a way for the *user* to limit the columns that a PipeOpTaskPreproc should operate on.
- train_imputer(feature,type,context)
  (atomic, character(1), data.table) -> any
  Called once for each feature selected by affect_columns to create the model entry to be used for $impute().
- impute(feature,type,model,context)
  (atomic, character(1), any, data.table) -> atomic
  Imputes the features. model is the model created by $train_imputer()

See Also

Other PipeOps: PipeOpEnsemble, PipeOpTaskPreproc, PipeOp, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_copy, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_encode, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix,

mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_scale, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_unbranch, mlr_pipeops_yeojohnson, mlr_pipeops

Other Imputation PipeOps: mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputenewlvl, mlr_pipeops_imputesample

---

PipeOpTaskPreproc *PipeOpTaskPreproc*

Description

Base class for handling most "preprocessing" operations. These are operations that have exactly one Task input and one Task output, and expect the column layout of these Tasks during input and output to be the same.

Users must implement $train_task() and $predict_task(), which have a Task input and should return that Task. The Task should, if possible, be manipulated in-place, and should not be cloned.

Alternatively, the $train_dt() and $predict_dt() functions can be implemented, which operate on data.table objects instead. This should generally only be done if all data is in some way altered (e.g. PCA changing all columns to principal components) and not if only a few columns are added or removed (e.g. feature selection) because this should be done at the Task-level with $train_task(). The $select_cols() function can be overloaded for $train_dt() and $predict_dt() to operate only on subsets of the Task's data, e.g. only on numerical columns.

If the `can_subset_cols` argument of the constructor is `TRUE` (the default), then the hyperparameter `affect_columns` is added, which can limit the columns of the Task that is modified by the PipeOpTaskPreproc using a Selector function. Note this functionality is entirely independent of the $select_cols() functionality.

PipeOpTaskPreproc is useful for operations that behave differently during training and prediction. For operations that perform essentially the same operation and only need to perform extra work to build a $state during training, the PipeOpTaskPreprocSimple class can be used instead.

Format

Abstract R6Class inheriting from PipeOp.

Construction

```
PipeOpTaskPreproc$new(id, param_set = ParamSet$new(), param_vals = list(), can_subset_cols = TRUE, pack
```

- `id` :: `character(1)`
  Identifier of resulting object. See $id slot of PipeOp.
- `param_set` :: ParamSet
  Parameter space description. This should be created by the subclass and given to `super$initialize()`.

- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings given in param_set. The subclass should have its own param_vals parameter and pass it on to super$initialize(). Default list().
- can_subset_cols :: logical(1)
  Whether the affect_columns parameter should be added which lets the user limit the columns that are modified by the PipeOpTaskPreproc. This should generally be FALSE if the operation adds or removes rows from the Task, and TRUE otherwise. Default is TRUE.
- packages :: character
  Set of all required packages for the PipeOp's $train and $predict methods. See $packages slot. Default is character(0).
- task_type :: character(1)
  The class of Task that should be accepted as input and will be returned as output. This should generally be a character(1) identifying a type of Task, e.g. "Task", "TaskClassif" or "TaskRegr" (or another subclass introduced by other packages). Default is "Task".

Input and Output Channels

PipeOpTaskPreproc has one input channel named "input", taking a Task, or a subclass of Task if the task_type construction argument is given as such; both during training and prediction.

PipeOpTaskPreproc has one output channel named "output", producing a Task, or a subclass; the Task type is the same as for input; both during training and prediction.

The output Task is the modified input Task according to the overloaded $train_task()/$predict_taks() or $train_dt()/$predict_dt() functions.

State

The $state is a named list; besides members added by inheriting classes, the members are:

- affect_cols :: character
  Names of features being selected by the affect_columns parameter, if present; names of *all* present features otherwise.
- intasklayout :: data.table
  Copy of the training Task's $feature_types slot. This is used during prediction to ensure that the prediction Task has the same features, feature layout, and feature types as during training.
- outtasklayout :: data.table
  Copy of the trained Task's $feature_types slot. This is used during prediction to ensure that the Task resulting from the prediction operation has the same features, feature layout, and feature types as after training.
- dt_columns :: character
  Names of features selected by the $select_cols() call during training. This is only present if the $train_dt() functionality is used, and not present if the $train_task() function is overloaded instead.

Parameters

- affect_columns :: function | Selector | NULL
  What columns the PipeOpTaskPreproc should operate on. This parameter is only present if the constructor is called with the can_subset_cols argument set to TRUE (the default).
  The parameter must be a Selector function, which takes a Task as argument and returns a character of features to use.
  See Selector for example functions. Defaults to NULL, which selects all features.

Internals

PipeOpTaskPreproc is an abstract class inheriting from PipeOp. It implements the $train_internal() and $predict_internal() functions. These functions perform checks and go on to call $train_task() and $predict_task(). A subclass of PipeOpTaskPreproc may implement these functions, or implement $train_dt() and $predict_dt() instead. This works by having the default implementations of $train_task() and $predict_task() call $train_dt() and $predict_dt(), respectively.

The affect_columns functionality works by unsetting columns by removing their "col_role" before processing, and adding them afterwards by setting the col_role to "feature".

Fields

Fields inherited from PipeOp.

Methods

Methods inherited from PipeOp, as well as:

- train_task
  (Task) -> Task
  Called by the PipeOpTaskPreproc's implementation of $train_internal(). Takes a single Task as input and modifies it (ideally in-place without cloning) while storing information in the $state slot. Note that unlike $train_internal(), the argument is *not* a list but a singular Task, and the return object is also *not* a list but a singular Task. Also, contrary to $train_internal(), the $state being generated must be a list, which the PipeOpTaskPreproc will add additional slots to (see Section *State*). Care should be taken to avoid name collisions between $state elements added by $train_task() and PipeOpTaskPreproc.
  By default this function calls the $train_dt() function, but it can be overloaded to perform operations on the Task directly.
- predict_task
  (Task) -> Task
  Called by the PipeOpTaskPreproc's implementation of $predict_internal(). Takes a single Task as input and modifies it (ideally in-place without cloning) while using information in the $state slot. Works analogously to $train_task(). If $predict_task() should only be overloaded if $train_task() is overloaded (i.e. $train_dt() is *not* used).
- train_dt(dt,levels,target)
  (data.table, named list, any) -> data.table | data.frame | matrix
  Train PipeOpTaskPreproc on dt, transform it and store a state in $state. A transformed object must be returned that can be converted to a data.table using as.data.table. dt does not need to be copied deliberately, it is possible and encouraged to change it in-place.

The `levels` argument is a named list of factor levels for factorial or character features. The `target` argument contains the $truth() information of the training `Task`; its type depends on the `Task` type being trained on.
This method can be overloaded when inheriting from `PipeOpTaskPreproc`, together with $predict_dt() and optionally $select_cols(); alternatively, $train_task() and $predict_task() can be overloaded.

- `predict_dt(dt,levels)`
(`data.table`, named `list`) -> `data.table` | `data.frame` | `matrix`
Predict on new data in `dt`, possibly using the stored $state. A transformed object must be returned that can be converted to a `data.table` using `as.data.table`. `dt` does not need to be copied deliberately, it is possible and encouraged to change it in-place.
The `levels` argument is a named list of factor levels for factorial or character features.
This method can be overloaded when inheriting `PipeOpTaskPreproc`, together with $train_dt() and optionally $select_cols(); alternatively, $train_task() and $predict_task() can be overloaded.
- `select_cols(task)`
(`Task`) -> `character`
Selects which columns the `PipeOp` operates on, if $train_dt() and $predict_dt() are overloaded. This function is not called if $train_task() and $predict_task() are overloaded. In contrast to the `affect_columns` parameter. `select_cols` is for the *inheriting class* to determine which columns the operator should function on, e.g. based on feature type, while `affect_columns` is a way for the *user* to limit the columns that a `PipeOpTaskPreproc` should operate on.
This method can optionally be overloaded when inheriting `PipeOpTaskPreproc`, together with $train_dt() and $predict_dt(); alternatively, $train_task() and $predict_task() can be overloaded.
If this method is not overloaded, it defaults to selecting all columns.

See Also

Other mlr3pipelines backend related: `Graph`, `PipeOpTaskPreprocSimple`, `PipeOp`, `mlr_pipeops`

Other PipeOps: `PipeOpEnsemble`, `PipeOpImpute`, `PipeOp`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_copy`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_encode`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputenewlvl`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_mutate`, `mlr_pipeops_nop`, `mlr_pipeops_pca`, `mlr_pipeops_quantilebin`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_scale`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_unbranch`, `mlr_pipeops_yeojohnson`, `mlr_pipeops`

`PipeOpTaskPreprocSimple`
PipeOpTaskPreprocSimple

Description

Base class for handling many "preprocessing" operations that perform essentially the same operation during training and prediction. Instead implementing a $train_task() and a $predict_task() operation, only a $get_state() and a $transform() operation needs to be defined, both of which take one argument: a Task.

Alternatively, analogously to the PipeOpTaskPreproc approach of offering $train_dt()/$predict_dt(), the $get_state_dt() and $transform_dt() functions may be implemented.
$get_state must not change its input value in-place and must return something that will be written into $state (which must not be NULL), `transform()` should modify its argument in-place; it is called both during training and prediction.

This inherits from PipeOpTaskPreproc and behaves essentially the same.

Format

Abstract R6Class inheriting from PipeOpTaskPreproc/PipeOp.

Construction

```
PipeOpTaskPreprocSimple$new(id, param_set = ParamSet$new(), param_vals = list(), can_subset_cols = TRU
```

(Construction is identical to PipeOpTaskPreproc.)

- `id` :: `character(1)`
  Identifier of resulting object. See $id slot of PipeOp.
- `param_set` :: ParamSet
  Parameter space description. This should be created by the subclass and given to `super$initialize()`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings given in `param_set`.
  The subclass should have its own `param_vals` parameter and pass it on to `super$initialize()`.
  Default `list()`.
- `can_subset_cols` :: `logical(1)`
  Whether the `affect_columns` parameter should be added which lets the user limit the columns that are modified by the PipeOpTaskPreprocSimple. This should generally be `FALSE` if the operation adds or removes rows from the Task, and `TRUE` otherwise. Default is `TRUE`.
- packages :: `character`
  Set of all required packages for the PipeOp's $train and $predict methods. See $packages slot.
  Default is `character(0)`.
- `task_type` :: `character(1)`
  The class of Task that should be accepted as input and will be returned as output. This should generally be a `character(1)` identifying a type of Task, e.g. `"Task"`, `"TaskClassif"` or `"TaskRegr"` (or another subclass introduced by other packages). Default is `"Task"`.

Input and Output Channels

Input and output channels are inherited from PipeOpTaskPreproc.

The output during training and prediction is the Task, modified by $transform() or $transform_dt().

State

The $state is a named `list` with the $state elements inherited from `PipeOpTaskPreproc`.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`.

Internals

`PipeOpTaskPreprocSimple` is an abstract class inheriting from `PipeOpTaskPreproc` and implementing the $train_task() and $predict_task() functions. A subclass of `PipeOpTaskPreprocSimple` may implement the functions $get_state() and $transform(), or alternatively the functions $get_state_dt() and $transform_dt() (as well as $select_cols(), in the latter case). This works by having the default implementations of $get_state() and $transform() call $get_state_dt() and $transform_dt().

Fields

Fields inherited from `PipeOp`.

Methods

Methods inherited from `PipeOpTaskPreproc`, as well as:

- `get_state(task)`
  (`Task`) -> named `list`
  Store create something that will be stored in $state during training phase of `PipeOpTaskPreprocSimple`. The state can then influence the $transform() function. Note that $get_state() must *return* the state, and should not store it in $state. It is not strictly necessary to implement either $get_state() or $get_state_dt(); if they are not implemented, the state will be stored as `list()`.
  This method can optionally be overloaded when inheriting from `PipeOpTaskPreprocSimple`, together with $transform(); alternatively, $get_state_dt() (optional) and $transform_dt() (and possibly $select_cols(), from `PipeOpTaskPreproc`) can be overloaded.
- `transform(task)`
  (`Task`) -> `Task`
  Predict on new data in `task`, possibly using the stored $state. `task` should not be cloned, instead it should be changed in-place. This method is called both during training and prediction phase, and should essentially behave the same independently of phase. (If this is incongruent with the functionality to be implemented, then it should inherit from `PipeOpTaskPreproc`, not from `PipeOpTaskPreprocSimple`.)
  This method can be overloaded when inheriting from `PipeOpTaskPreprocSimple`, optionally with $get_state(); alternatively, $get_state_dt() (optional) and $transform_dt() (and possibly $select_cols(), from `PipeOpTaskPreproc`) can be overloaded.
- `get_state_dt(dt)`
  (`data.table`) -> named `list`
  Create something that will be stored in $state during training phase of `PipeOpTaskPreprocSimple`. The state can then influence the $transform_dt() function. Note that $get_state_dt() must *return* the state, and should not store it in $state. If neither $get_state() nor $get_state_dt() are overloaded, the state will be stored as `list()`.
  This method can optionally be overloaded when inheriting from `PipeOpTaskPreprocSimple`,

together with $transform_dt() (and optionally $select_cols(), from PipeOpTaskPreproc); Alternatively, $get_state() (optional) and $transform() can be overloaded.

- `transform_dt(dt)`
  (data.table) -> data.table | data.frame | matrix
  Predict on new data in dt, possibly using the stored $state. A transformed object must be returned that can be converted to a data.table using as.data.table. dt does not need to be copied deliberately, it is possible and encouraged to change it in-place. This method is called both during training and prediction phase, and should essentially behave the same independently of phase. (If this is incongruent with the functionality to be implemented, then it should inherit from PipeOpTaskPreproc, not from PipeOpTaskPreprocSimple.)
  This method can optionally be overloaded when inheriting from PipeOpTaskPreprocSimple, together with $transform_dt() (and optionally $select_cols(), from PipeOpTaskPreproc); Alternatively, $get_state() (optional) and $transform() can be overloaded.

See Also

Other mlr3pipelines backend related: Graph, PipeOpTaskPreproc, PipeOp, mlr_pipeops

---

po *Shorthand PipeOp Constructor*

Description

Create

- a PipeOp from mlr_pipeops from given ID
- a PipeOpLearner from a Learner object
- a PipeOpFilter from a Filter object

The object is initialized with given parameters and param_vals.

Usage

```
po(.obj, ...)
```

Arguments

.obj	[any] The object from which to construct a PipeOp. If this is a character(1), it is looked up in the mlr_pipeops dictionary. Otherwise, it is converted to a PipeOp.
...	any Additional parameters to give to constructed object. This may be an argument of the constructor of the PipeOp, in which case it is given to this constructor; or it may be a parameter value, in which case it is given to the param_vals argument of the constructor.

Examples

```
library("mlr3")

po("learner", lrn("classif.rpart"), cp = 0.3)

po(lrn("classif.rpart"), cp = 0.3)

# is equivalent with:
mlr_pipeops$get("learner", lrn("classif.rpart"),
  param_vals = list(cp = 0.3))
```

---

register_autoconvert_function

Add Autoconvert Function to Conversion Register

Description

Add functions that perform conversion to a desired class.

Whenever a Graph or a PipeOp is called with an object that does not conform to its declared input type, the "autoconvert register" is queried for functions that may turn the object into a desired type.

Usage

```
register_autoconvert_function(cls, fun, packages = character(0))
```

Arguments

cls	character(1) The class that fun converts to.
fun	function The conversion function. Must take one argument and return an object of class cls, or possibly a sub-class as recognized by are_types_compatible().
packages	character The packages required to be loaded for fun to operate.

Value

NULL.

See Also

Other class hierarchy operations: add_class_hierarchy_cache(), reset_autoconvert_register(), reset_class_hierarchy_cache()

Examples

```
# This lets mlr3pipelines automatically try to convert a string into
# a `PipeOp` by querying the [`mlr_pipeops`] [`Dictionary`][mlr3misc::Dictionary].
# This is an example and not necessary, because mlr3pipelines adds it by default.
register_autoconvert_function("PipeOp", function(x) as_pipeop(x), packages = "mlr3pipelines")
```

`reset_autoconvert_register` *Reset Autoconvert Register*

Description

Reset autoconvert register to factory default, thereby undoing any calls to `register_autoconvert_function()` by the user.

Usage

```
reset_autoconvert_register()
```

Value

`NULL`

See Also

Other class hierarchy operations: `add_class_hierarchy_cache()`, `register_autoconvert_function()`, `reset_class_hierarchy_cache()`

`reset_class_hierarchy_cache` *Reset the Class Hierarchy Cache*

Description

Reset the class hierarchy cache to factory default, thereby undoing any calls to `add_class_hierarchy_cache()` by the user.

Usage

```
reset_class_hierarchy_cache()
```

Value

`NULL`

See Also

Other class hierarchy operations: `add_class_hierarchy_cache()`, `register_autoconvert_function()`, `reset_autoconvert_register()`

`Selector` *Selector Functions*

Description

A `Selector` function is used by different `PipeOp`s, most prominently `PipeOpSelect` and many `PipeOp`s inheriting from `PipeOpTaskPreproc`, to determine a subset of `Task`s to operate on.

Even though a `Selector` is a `function` that can be written itself, it is preferable to use the `Selector` constructors shown here. Each of these can be called with its arguments to create a `Selector`, which can then be given to the `PipeOpSelect` `selector` parameter, or many `PipeOpTaskPreproc`s' `affect_columns` parameter. See there for examples of this usage.

Usage

```
selector_all()

selector_none()

selector_type(types)

selector_grep(pattern, ignore.case = FALSE, perl = FALSE, fixed = FALSE)

selector_name(feature_names, assert_present = FALSE)

selector_invert(selector)

selector_intersect(selector_x, selector_y)

selector_union(selector_x, selector_y)

selector_setdiff(selector_x, selector_y)
```

Arguments

`types`	(`character`) Type of feature to select
`pattern`	(`character(1)`) grep pattern
`ignore.case`	(`logical(1)`) ignore case
`perl`	(`logical(1)`) perl regex
`fixed`	(`logical(1)`) fixed pattern instead of regex
`feature_names`	(`character`) Select features by exact name match.

assert_present (logical(1))
Throw an error if feature_names are not all present in the task being operated on.

selector (Selector)
Selector to invert.

selector_x (Selector)
First Selector to query.

selector_y (Selector)
Second Selector to query.

Value

function: A Selector function that takes a Task and returns the feature names to be processed.

Functions

- selector_all: selector_all selects all features.
- selector_none: selector_none selects none of the features.
- selector_type: selector_type selects features according to type. Legal types are listed in mlr_reflections$task_feature_types.
- selector_grep: selector_grep selects features with names matching the grep() pattern.
- selector_name: selector_name selects features with names matching exactly the names listed.
- selector_invert: selector_invert inverts a given Selector: It always selects the features that would be *dropped* by the other Selector, and drops the features that would be kept.
- selector_intersect: selector_intersect selects the intersection of two Selectors: Only features selected by both Selectors are selected in the end.
- selector_union: selector_union selects the union of two Selectors: Features selected by either Selector are selected in the end.
- selector_setdiff: selector_setdiff selects the setdiff of two Selectors: Features selected by selector_x are selected, unless they are also selected by selector_y.

Details

A Selector is a function that has one input argument (commonly named task). The function is called with the Task that a PipeOp is operating on. The return value of the function must be a character vector that is a subset of the feature names present in the Task.

For example, a Selector that selects all columns is

```
function(task) {
  task$feature_names
}
```

(this is the selector_all()-Selector.) A Selector that selects all columns that have names shorter than four letters would be:

```
function(task) {
  task$feature_names[
    nchar(task$feature_names) < 4
  ]
}
```

A Selector that selects only the column "Sepal.Length" (as in the iris task), if present, is

```
function(task) {
  intersect(task$feature_names, "Sepal.Length")
}
```

It is preferable to use the Selector construction functions like select_type, select_grep etc. if possible, instead of writing custom Selectors.

See Also

Other Selectors: mlr_pipeops_select

Examples

```
library("mlr3")

iris_task = tsk("iris")
bh_task = tsk("boston_housing")

sela = selector_all()
sela(iris_task)
sela(bh_task)

self = selector_type("factor")
self(iris_task)
self(bh_task)

selg = selector_grep("a.*i")
selg(iris_task)
selg(bh_task)

selgi = selector_invert(selg)
selgi(iris_task)
selgi(bh_task)

selgf = selector_union(selg, self)
selgf(iris_task)
selgf(bh_task)
```

%>>% *PipeOp Composition Operator*

Description

This operator "pipes" data from the source g1 into the sink g2. Both source and sink can either be a Graph or a PipeOp (or an object that can be automatically converted into a Graph or PipeOp, see as_graph() and as_pipeop()).
%>>% tries to automatically match output channels of g1 to input channels of g2; this is only possible if either

- the number of output channels of g1 (as given by g1$output) is equal to the number of input channels of g2 (as given by g2$input), or
- g1 has only one output channel (i.e. g1$output has one line), or
- g2 has only one input channel, which is a *vararg* channel (i.e. g2$input has one line, with name entry "...").

Connections between channels are created in the order in which they occur in g1 and g2, respectively: g1's output channel 1 is connected to g2's input channel 1, channel 2 to 2 etc.

This operator always created deep copies of its input arguments, so they cannot be modified by reference afterwards. To access individual PipeOps after composition, use the resulting Graph's $pipeops list.

Both arguments of %>>% are automatically converted to Graphs using as_graph(); this means that objects on either side may be objects that can be automatically converted to PipeOps (such as Learners or Filters), or that can be converted to Graphs. This means, in particular, lists of Graphs, PipeOps or objects convertible to that, because as_graph() automatically applies gunion() to lists. See examples.

Usage

```
g1 %>>% g2
```

Arguments

g1	(Graph | PipeOp | Learner | Filter | list | ...) Graph / PipeOp / object-convertible-to-PipeOp to put in front of g2.
g2	(Graph | PipeOp | Learner | Filter | list | ...) Graph / PipeOp / object-convertible-to-PipeOp to put after g1.

Value

Graph: the constructed Graph.

See Also

Other Graph operators: as_graph(), as_pipeop(), assert_graph(), assert_pipeop(), greplicate(), gunion()

Examples

```
o1 = PipeOpScale$new()
o2 = PipeOpPCA$new()
o3 = PipeOpFeatureUnion$new(2)

# The following two are equivalent:
pipe1 = o1 %>>% o2

pipe2 = Graph$new()$
  add_pipeop(o1$clone(deep = TRUE))$
  add_pipeop(o2$clone(deep = TRUE))$
  add_edge(o1$id, o2$id)

# Note automatical gunion() of lists.
# The following three are equivalent:
graph1 = list(o1, o2) %>>% o3

graph2 = gunion(list(o1, o2)) %>>% o3

graph3 = Graph$new()$
  add_pipeop(o1$clone(deep = TRUE))$
  add_pipeop(o2$clone(deep = TRUE))$
  add_pipeop(o3$clone(deep = TRUE))$
  add_edge(o1$id, o3$id, dst_channel = 1)$
  add_edge(o2$id, o3$id, dst_channel = 2)
```

Index