# Package 'mlr3pipelines'

July 1, 2024

**Title** Preprocessing Operators and Pipelines for 'mlr3'

**Version** 0.6.0

**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, lgr, mlr3 (>= 0.20.0), mlr3misc (>= 0.9.0), paradox, R6, withr

**Suggests** ggplot2, glmnet, igraph, knitr, lme4, mlbench, bbotk (>= 0.3.0), mlr3filters (>= 0.1.1), mlr3learners, mlr3measures, nloptr, quanteda, rmarkdown, rpart, stopwords, testthat, visNetwork, bestNormalize, fastICA, kernlab, smotefamily, evaluate, NMF, MASS, kknn, GenSA, methods, vtreat, future, htmlwidgets

**ByteCompile** true

**Encoding** UTF-8

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**NeedsCompilation** no

**RoxygenNote** 7.3.2

**VignetteBuilder** knitr

**Collate** 'Graph.R' 'GraphLearner.R' 'mlr_pipeops.R' 'multiplicity.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'
'PipeOpColRoles.R' 'PipeOpCollapseFactors.R' 'PipeOpCopy.R'
'PipeOpDateFeatures.R' 'PipeOpEncode.R' 'PipeOpEncodeImpact.R'
'PipeOpEncodeLmer.R' 'PipeOpFeatureUnion.R' 'PipeOpFilter.R'
'PipeOpFixFactors.R' 'PipeOpHistBin.R' 'PipeOpICA.R'
'PipeOpImpute.R' 'PipeOpImputeConstant.R' 'PipeOpImputeHist.R'
'PipeOpImputeLearner.R' 'PipeOpImputeMean.R'
'PipeOpImputeMedian.R' 'PipeOpImputeMode.R' 'PipeOpImputeOOR.R'
'PipeOpImputeSample.R' 'PipeOpKernelPCA.R' 'PipeOpLearner.R'
'PipeOpLearnerCV.R' 'PipeOpMissingIndicators.R'
'PipeOpModelMatrix.R' 'PipeOpMultiplicity.R' 'PipeOpMutate.R'
'PipeOpNMF.R' 'PipeOpNOP.R' 'PipeOpOVR.R' 'PipeOpPCA.R'
'PipeOpProxy.R' 'PipeOpQuantileBin.R'
'PipeOpRandomProjection.R' 'PipeOpRandomResponse.R'
'PipeOpRegrAvg.R' 'PipeOpRemoveConstants.R'
'PipeOpRenameColumns.R' 'PipeOpScale.R' 'PipeOpScaleMaxAbs.R'
'PipeOpScaleRange.R' 'PipeOpSelect.R' 'PipeOpSmote.R'
'PipeOpSpatialSign.R' 'PipeOpSubsample.R'
'PipeOpTextVectorizer.R' 'PipeOpThreshold.R' 'PipeOpTrafo.R'
'PipeOpTuneThreshold.R' 'PipeOpUnbranch.R' 'PipeOpVtreat.R'
'PipeOpYeoJohnson.R' 'Selector.R' 'assert_graph.R'
'bibentries.R' 'greplicate.R' 'gunion.R' 'mlr_graphs.R'
'operators.R' 'pipeline_bagging.R' 'pipeline_branch.R'
'pipeline_convert_types.R' 'pipeline_greplicate.R'
'pipeline_ovr.R' 'pipeline_robustify.R' 'pipeline_stacking.R'
'pipeline_targettrafo.R' 'po.R' 'ppl.R' 'reexports.R'
'typecheck.R' 'zzz.R'

**Author** Martin Binder [aut, cre],
Florian Pfisterer [aut] (<https://orcid.org/0000-0001-8867-762X>),
Lennart Schneider [aut] (<https://orcid.org/0000-0003-4152-5308>),
Bernd Bischl [aut] (<https://orcid.org/0000-0001-6002-6980>),
Michel Lang [aut] (<https://orcid.org/0000-0001-9754-0393>),
Sebastian Fischer [aut] (<https://orcid.org/0000-0002-9609-3197>),
Susanne Dandl [aut]

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

**Repository** CRAN

**Date/Publication** 2024-07-01 13:30:02 UTC

# Contents

---

`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)
- Lennart Schneider <lennart.sch@web.de> (ORCID)
- Bernd Bischl <bernd_bischl@gmx.net> (ORCID)
- Michel Lang <michellang@gmail.com> (ORCID)
- Sebastian Fischer <sebf.fischer@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"))
```

## as.Multiplicity *Convert an object to a Multiplicity*

### Description

Convert an object to a Multiplicity.

### Usage

```
as.Multiplicity(x)
```

### Arguments

| | |
|---|---|
| x | (any)<br>Object to convert. |

### Value

Multiplicity

---

## assert_graph *Assertion for mlr3pipelines 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)<br>Object to check. |

### Value

Graph invisible(x)

### See Also

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

---

## assert_pipeop *Assertion for mlr3pipelines 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)<br>Object to check. |

### Value

PipeOp invisible(x)

### See Also

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

---

## as_graph *Conversion to mlr3pipelines 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)<br>Object to convert. |
| clone | (logical(1))<br>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(), chain_graphs(), greplicate(), gunion(), mlr_graphs_greplicate

---

as_pipeop *Conversion to mlr3pipelines 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)<br>Object to convert. |
| clone | (logical(1))<br>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(), chain_graphs(), greplicate(), gunion(), mlr_graphs_greplicate

chain_graphs *Chain a Series 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 serial Graph, as if connecting them using %>>%.

Care is taken to avoid unnecessarily cloning of components. A call of chain_graphs(list(g1, g2, g3, g4, ...), in_place = FALSE) is equivalent to g1 %>>% g2 %>>!% g3 %>>!% g4 %>>!% .... A call of chain_graphs(list(g1, g2, g3, g4, ...), in_place = FALSE) is equivalent to g1 %>>!% g2 %>>!% g3 %>>!% g4 %>>!% ... (differing in the first operator being %>>!% as well).

## Usage

```
chain_graphs(graphs, in_place = FALSE)
```

## Arguments

| | |
|---|---|
| graphs | list of (Graph \| PipeOp \| NULL \| ...)<br>List of elements which are the Graphs to be joined. Elements must be convertible to Graph or PipeOp using as_graph() and as_pipeop(). NULL is the neutral element of %>>% and skipped. |
| in_place | (logical(1))<br>Whether to try to avoid cloning the first element of graphs, similar to the difference of %>>!% over %>>%. This can only be avoided if graphs[[1]] is already a Graph. Beware that, if chain_graphs() fails because of id collisions, then graphs[[1]] will possibly be in an incompletely modified state when in_place is TRUE. |

## Value

Graph the resulting Graph, or NULL if there are no non-null values in graphs.

## See Also

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

## filter_noop *Remove NO_OPs from a List*

### Description

Remove all NO_OP elements from a list.

### Usage

```
filter_noop(x)
```

### Arguments

| | |
|---|---|
| x | list<br>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 Base Class*

### 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.

### 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.
- phash :: character(1)
  Stores a checksum calculated on the Graph configuration, which includes all PipeOp hashes *except* 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.
- man :: character(1)
  Identifying string of the help page that shows with help().

## 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, clone = TRUE)
  (PipeOp | Learner | Filter | ..., logical(1)) -> 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(). The argument given as op is cloned if clone is TRUE (default); to access a Graph's PipeOps by-reference, use $pipeops. Note that $add_pipeop() is a relatively low-level operation, it is recommended to build graphs using %>>%.
- 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).
- chain(gs, clone = TRUE)
  (list of Graphs, logical(1)) -> self
  Takes a list 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 serial Graph coming after self, as if connecting them using %>>%.
- 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(dot = FALSE, dotname = "dot", fontsize = 24L)
  (logical(1), character(1), integer(1)) -> NULL

Print a representation of the Graph on the console. If dot is FALSE, 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). If dot is TRUE, print a DOT representation of the Graph on the console. The DOT output can be named via the argument dotname and the fontsize can also be specified.

- 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!
- update_ids(prefix = "", postfix = "")
(character, character) -> self
Pre- or postfix PipeOp's existing ids. Both prefix and postfix default to "", i.e. no changes.
- 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().
- help(help_type)
(character(1)) -> help file
Displays the help file of the concrete PipeOp instance. help_type is one of "text", "html", "pdf" and behaves as the help_type argument of R's help().

## See Also

Other mlr3pipelines backend related: PipeOp, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_graphs, mlr_pipeops, mlr_pipeops_updatetarget

## 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.

This function is deprecated and will be removed in the next version in favor of using pipeline_greplicate / ppl("greplicate").

## Usage

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

## Arguments

| | |
|---|---|
| graph | Graph<br>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(), chain_graphs(), gunion(), mlr_graphs_greplicate

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.

This operation always creates 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.

## Usage

```
gunion(graphs, in_place = FALSE)
```

## Arguments

| | |
|---|---|
| graphs | list of (Graph \| PipeOp \| NULL \| ...)<br>List of elements which are the Graphs to be joined. Elements must be convertible to Graph or PipeOp using as_graph() and as_pipeop(). NULL values automatically get converted to PipeOpNOP with a random ID of the format nop_********. The list can be named, in which case the IDs of the elements are prefixed with the names, separated by a dot (.). |
| in_place | (logical(1) \| logical)<br>Whether to try to avoid cloning the first element of graphs, similar to the difference of %>>!% over %>>%. This can only be avoided if graphs[[1]] is already a Graph.<br>Unlike chain_graphs(), gunion() does all checks *before* mutating graphs[[1]], so it will not leave graphs[[1]] in an incompletely modified state when it fails.<br>in_place may also be of length graph, in which case it determines for each element of graphs whether it is cloned. This is for internal usage and is not recommended. |

## Value

Graph the resulting Graph.

## See Also

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

## is.Multiplicity *Check if an object is a Multiplicity*

### Description

Check if an object is a Multiplicity.

### Usage

```
is.Multiplicity(x)
```

### Arguments

| | |
|---|---|
| x | (any)<br>Object to check. |

### Value

logical(1)

## is_noop *Test for NO_OP*

### Description

Test whether a given object is a NO_OP.

### Usage

```
is_noop(x)
```

### Arguments

| | |
|---|---|
| x | any<br>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_graphs` *Dictionary of (sub-)graphs*

## Description

A simple `Dictionary` storing objects of class `Graph`. The dictionary contains a collection of often-used graph structures, and it's aim is solely to make often-used functions more accessible. Each `Graph` has an associated help page, which can be accessed via `?mlr_graphs_<key>`, i.e. `?mlr_graphs_bagging`.

## Format

`R6Class` object inheriting from `mlr3misc::Dictionary`.

## Methods

Methods inherited from `Dictionary`, as well as:

- `add(key, value)`
  (`character(1)`, `function`)
  Adds constructor `value` to the dictionary with key `key`, potentially overwriting a previously stored item.

## S3 methods

- `as.data.table(dict)`
  `Dictionary` -> `data.table::data.table`
  Returns a `data.table` with column `key` (`character`).

## See Also

Other mlr3pipelines backend related: `Graph`, `PipeOp`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `mlr_pipeops_updatetarget`

Other Dictionaries: `mlr_pipeops`

## Examples

```
library(mlr3)
lrn = lrn("regr.rpart")
task = mlr_tasks$get("boston_housing")

# Robustify the learner for the task.
gr = pipeline_robustify(task, lrn) %>>% po("learner", lrn)
# or equivalently
gr = mlr_graphs$get("robustify", task = task, learner = lrn) %>>% po(lrn)
# or equivalently
gr = ppl("robustify", task, lrn) %>>% po("learner", lrn)

# all Graphs currently in the dictionary:
```

```
as.data.table(mlr_graphs)
```

---

`mlr_graphs_bagging` *Create a bagging learner*

---

## Description

Creates a `Graph` that performs bagging for a supplied graph. This is done as follows:

- `Subsample` the data in each step using `PipeOpSubsample`, afterwards apply `graph`
- Replicate this step `iterations` times (in parallel via multiplicities)
- Average outputs of replicated `graph`s predictions using the `averager` (note that setting `collect_multipliciy = TRUE` is required)

All input arguments are cloned and have no references in common with the returned `Graph`.

## Usage

```
pipeline_bagging(
  graph,
  iterations = 10,
  frac = 0.7,
  averager = NULL,
  replace = FALSE
)
```

## Arguments

| | |
|---|---|
| `graph` | `PipeOp` \| `Graph`<br>A `PipeOpLearner` or `Graph` to create a robustifying pipeline for. Outputs from the replicated `graph`s are connected with the `averager`. |
| `iterations` | `integer(1)`<br>Number of bagging iterations. Defaults to 10. |
| `frac` | `numeric(1)`<br>Percentage of rows to keep during subsampling. See `PipeOpSubsample` for more information. Defaults to 0.7. |
| `averager` | `PipeOp` \| `Graph`<br>A `PipeOp` or `Graph` that averages the predictions from the replicated and subsampled graph's. In the simplest case, `po("classifavg")` and `po("regravg")` can be used in order to perform simple averaging of classification and regression predictions respectively. If `NULL` (default), no averager is added to the end of the graph. Note that setting `collect_multipliciy = TRUE` during construction of the averager is required. |
| `replace` | `logical(1)`<br>Whether to sample with replacement. Default `FALSE`. |

## Value

Graph

## Examples

```
library(mlr3)
lrn_po = po("learner", lrn("regr.rpart"))
task = mlr_tasks$get("boston_housing")
gr = pipeline_bagging(lrn_po, 3, averager = po("regravg", collect_multiplicity = TRUE))
resample(task, GraphLearner$new(gr), rsmp("holdout"))$aggregate()

# The original bagging method uses boosting by sampling with replacement.
gr = ppl("bagging", lrn_po, frac = 1, replace = TRUE,
  averager = po("regravg", collect_multiplicity = TRUE))
resample(task, GraphLearner$new(gr), rsmp("holdout"))$aggregate()
```

---

# mlr_graphs_branch *Branch Between Alternative Paths*

---

## Description

Create a multiplexed graph.

All input arguments are cloned and have no references in common with the returned Graph.

## Usage

```
pipeline_branch(graphs, prefix_branchops = "", prefix_paths = FALSE)
```

## Arguments

| | |
|---|---|
| graphs | list of Graph<br>Multiple graphs, possibly named. They all must have exactly one output. If any of the arguments are named, then all must have unique names. |
| prefix_branchops | character(1)<br>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)<br>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. |

## Value

Graph

## Examples

```
library("mlr3")

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

branches = pipeline_branch(list(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)

pipeline_branch(list(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")
```

---

# mlr_graphs_convert_types

*Convert Column Types*

---

## Description

Converts all columns of type `type_from` to `type_to`, using the corresponding R function (e.g. `as.numeric()`, `as.factor()`). It is possible to further subset the columns that should be affected using the `affect_columns` argument. The resulting Graph contains a PipeOpColApply, followed, if appropriate, by a PipeOpFixFactors.

Unlike R's `as.factor()` function, `ppl("convert_types")` will convert `ordered` types into (unordered) `factor` vectors.

## Usage

```
pipeline_convert_types(
  type_from,
  type_to,
  affect_columns = NULL,
  id = NULL,
  fixfactors = NULL,
  more_args = list()
)
```

## Arguments

| | |
|---|---|
| type_from | character<br>Which column types to convert. May be any combination of "logical", "integer", "numeric", "factor", "ordered", "character", or "POSIXct". |
| type_to | character(1)<br>Which type to convert to. Must be a scalar value, exactly one of the types allowed in type_from. |
| affect_columns | function \| Selector \| NULL<br>Which columns to affect. This argument can further restrict the columns being converted, beyond the type_from argument. Must be a Selector-like function, which takes a Task as argument and returns a character of features to use. |
| id | character(1) \| NULL<br>ID to give to the constructed PipeOps. Defaults to an ID built automatically from type_from and type_to. If a PipeOpFixFactors is appended, its ID will be paste0(id, "_ff"). |
| fixfactors | logical(1) \| NULL<br>Whether to append a PipeOpFixFactors. Defaults to TRUE if and only if type_to is "factor" or "ordered". |
| more_args | list<br>Additional arguments to give to the conversion function. This could e.g. be used to pass the timezone to as.POSIXct. |

## Value

Graph

## Examples

```
library("mlr3")

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

graph = ppl("convert_types", "character", "factor")
str(graph$train(task_chr)[[1]]$data())

graph_z = ppl("convert_types", "character", "factor",
  affect_columns = selector_name("z"))
graph_z$train(task_chr)[[1]]$data()

# `affect_columns` and `type_from` are both applied. The following
# looks for a 'numeric' column with name 'z', which is not present;
# the task is therefore unchanged.
graph_z = ppl("convert_types", "numeric", "factor",
```

```
  affect_columns = selector_name("z"))
graph_z$train(task_chr)[[1]]$data()
```

---

`mlr_graphs_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`.

All input arguments are cloned and have no references in common with the returned `Graph`.

## Usage

```
pipeline_greplicate(graph, n)
```

## Arguments

| | |
|---|---|
| `graph` | `Graph`<br>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()`, `chain_graphs()`, `greplicate()`, `gunion()`

## Examples

```
library("mlr3")

po_pca = po("pca")
pipeline_greplicate(po_pca, n = 2)
```

mlr_graphs_ovr *Create A Graph to Perform "One vs. Rest" classification.*

## Description

Create a new Graph for a classification Task to perform "One vs. Rest" classification.

All input arguments are cloned and have no references in common with the returned Graph.

## Usage

```
pipeline_ovr(graph)
```

## Arguments

| | |
|---|---|
| graph | Graph<br>Graph being wrapped between PipeOpOVRSplit and PipeOpOVRUnite. The Graph should return NULL during training and a classification Prediction during prediction. |

## Value

Graph

## Examples

```
library("mlr3")

task = tsk("wine")

learner = lrn("classif.rpart")
learner$predict_type = "prob"

# Simple OVR
g1 = pipeline_ovr(learner)
g1$train(task)
g1$predict(task)

# Bagged Learners
gr = po("replicate", reps = 3) %>>%
  po("subsample") %>>%
  learner %>>%
  po("classifavg", collect_multiplicity = TRUE)
g2 = pipeline_ovr(gr)
g2$train(task)
g2$predict(task)

# Bagging outside OVR
g3 = po("replicate", reps = 3) %>>%
  pipeline_ovr(po("subsample") %>>% learner) %>>%
```

```
  po("classifavg", collect_multiplicity = TRUE)
g3$train(task)
g3$predict(task)
```

---

`mlr_graphs_robustify` *Robustify a learner*

---

## Description

Creates a `Graph` that can be used to robustify any subsequent learner. Performs the following steps:

- Drops empty factor levels using `PipeOpFixFactors`
- Imputes `numeric` features using `PipeOpImputeHist` and `PipeOpMissInd`
- Imputes `factor` features using `PipeOpImputeOOR`
- Encodes `factors` using `one-hot-encoding`. Factors with a cardinality > max_cardinality are collapsed using `PipeOpCollapseFactors`

The graph is built conservatively, i.e. the function always tries to assure everything works. If a learner is provided, some steps can be left out, i.e. if the learner can deal with factor variables, no encoding is performed.

All input arguments are cloned and have no references in common with the returned `Graph`.

## Usage

```
pipeline_robustify(
  task = NULL,
  learner = NULL,
  impute_missings = NULL,
  factors_to_numeric = NULL,
  max_cardinality = 1000,
  ordered_action = "factor",
  character_action = "factor",
  POSIXct_action = "numeric"
)
```

## Arguments

| | |
|---|---|
| `task` | `Task`<br>A `Task` to create a robustifying pipeline for. Optional, if omitted, the "worst possible" `Task` is assumed and the full pipeline is created. |
| `learner` | `Learner`<br>A learner to create a robustifying pipeline for. Optional, if omitted, the "worst possible" `Learner` is assumed and a more conservative pipeline is built. |

impute_missings
: logical(1) | NULL
Should missing values be imputed? Defaults to NULL: imputes if the task has missing values (or factors that are not encoded to numerics) and the learner can not handle them.

factors_to_numeric
: logical(1) | NULL
Should (ordered and unordered) factors be encoded? Defaults to NULL: encodes if the task has factors (or character columns that get converted to factor) and the learner can not handle factors.

max_cardinality
: integer(1)
Maximum number of factor levels allowed. See above. Default: 1000.

ordered_action
: character(1)
How to handle ordered columns: "factor" (default) or "factor!": convert to factor columns; "numeric" or "numeric!": convert to numeric columns; "integer" or "integer!": convert to integer columns; "ignore" or "ignore!": ignore. When task is given and has no ordered columns, or when learner is given and can handle ordered, then "factor", "numeric" and "integer" are treated like "ignore". This means it is necessary to add the exclamation point to override Task or Learner properties when given. "ignore" and "ignore!" therefore behave completely identically, "ignore!" is only present for consistency.
When ordered features are converted to factor, then they are treated like factor features further down in the pipeline, and are possibly eventually converted to numerics, but in a different way: factors get one-hot encoded, ordered_action = "numeric" converts ordered using as.numeric to their integer-valued rank.

character_action
: character(1)
How to handle character columns: "factor" (default) or "factor!": convert to factor columns; "matrix" or "matrix!": Use PipeOpTextVectorizer. "ignore" or "ignore!": ignore. When task is given and has no character columns, or when learner is given and can handle character, then "factor" and "matrix" are treated like "ignore". This means it is necessary to add the exclamation point to override Task or Learner properties when given. "ignore" and "ignore!" therefore behave completely identically, "ignore!" is only present for consistency.
When character columns are converted to factor, then they are treated like factor further down in the pipeline, and are possibly eventually converted to numerics, using one-hot encoding.

POSIXct_action
: character(1)
How to handle POSIXct columns: "numeric" (default) or "numeric!": convert to numeric columns; "datefeatures" or "datefeatures!": Use PipeOpDateFeatures. "ignore" or "ignore!": ignore. When task is given and has no POSIXct columns, or when learner is given and can handle POSIXct, then "numeric" and "datefeatures" are treated like "ignore". This means it is necessary to add the exclamation point to override Task or Learner properties when given. "ignore" and "ignore!" therefore behave completely identically, "ignore!"

is only present for consistency.

## Value

Graph

## Examples

```
library(mlr3)
lrn = lrn("regr.rpart")
task = mlr_tasks$get("boston_housing")
gr = pipeline_robustify(task, lrn) %>>% po("learner", lrn)
resample(task, GraphLearner$new(gr), rsmp("holdout"))
```

---

# mlr_graphs_stacking *Create A Graph to Perform Stacking.*

---

## Description

Create a new Graph for stacking. A stacked learner uses predictions of several base learners and fits a super learner using these predictions as features in order to predict the outcome.

All input arguments are cloned and have no references in common with the returned Graph.

## Usage

```
pipeline_stacking(
  base_learners,
  super_learner,
  method = "cv",
  folds = 3,
  use_features = TRUE
)
```

## Arguments

| | |
|---|---|
| base_learners | list of Learner<br>A list of base learners. |
| super_learner | Learner<br>The super learner that makes the final prediction based on the base learners. |
| method | character(1)<br>"cv" (default) for building a super learner using cross-validated predictions of the base learners or "insample" for building a super learner using the predictions of the base learners trained on all training data. |

folds — integer(1)
Number of cross-validation folds. Only used for method = "cv". Default 3.

use_features — logical(1)
Whether the original features should also be passed to the super learner. Default TRUE.

## Value

Graph

## Examples

```
if (requireNamespace("kknn")) {
library(mlr3)
library(mlr3learners)

base_learners = list(
  lrn("classif.rpart", predict_type = "prob"),
  lrn("classif.kknn", predict_type = "prob")
)
super_learner = lrn("classif.log_reg")

graph_stack = pipeline_stacking(base_learners, super_learner)
graph_learner = as_learner(graph_stack)
graph_learner$train(tsk("german_credit"))
}
```

---

# mlr_graphs_targettrafo

*Transform and Re-Transform the Target Variable*

---

## Description

Wraps a Graph that transforms a target during training and inverts the transformation during prediction. This is done as follows:

- Specify a transformation and inversion function using any subclass of PipeOpTargetTrafo, defaults to PipeOpTargetMutate, afterwards apply graph.
- At the very end, during prediction the transformation is inverted using PipeOpTargetInvert.
- To set a transformation and inversion function for PipeOpTargetMutate see the parameters trafo and inverter of the param_set of the resulting Graph.
- Note that the input graph is not explicitly checked to actually return a Prediction during prediction.

All input arguments are cloned and have no references in common with the returned Graph.

## Usage

```
pipeline_targettrafo(
  graph,
  trafo_pipeop = PipeOpTargetMutate$new(),
  id_prefix = ""
)
```

## Arguments

| | |
|---|---|
| graph | PipeOpLearner \| Graph<br>A PipeOpLearner or Graph to wrap between a transformation and re-transformation of the target variable. |
| trafo_pipeop | PipeOp<br>A PipeOp that is a subclass of PipeOpTargetTrafo. Default is PipeOpTargetMutate. |
| id_prefix | character(1)<br>Optional id prefix to prepend to PipeOpTargetInvert ID. The resulting ID will be "[id_prefix]targetinvert". Default is "". |

## Value

Graph

## Examples

```
library("mlr3")

tt = pipeline_targettrafo(PipeOpLearner$new(LearnerRegrRpart$new()))
tt$param_set$values$targetmutate.trafo = function(x) log(x, base = 2)
tt$param_set$values$targetmutate.inverter = function(x) list(response = 2 ^ x$response)

# gives the same as
g = Graph$new()
g$add_pipeop(PipeOpTargetMutate$new(param_vals = list(
  trafo = function(x) log(x, base = 2),
  inverter = function(x) list(response = 2 ^ x$response))
  )
)
g$add_pipeop(LearnerRegrRpart$new())
g$add_pipeop(PipeOpTargetInvert$new())
g$add_edge(src_id = "targetmutate", dst_id = "targetinvert",
  src_channel = 1, dst_channel = 1)
g$add_edge(src_id = "targetmutate", dst_id = "regr.rpart",
  src_channel = 2, dst_channel = 1)
g$add_edge(src_id = "regr.rpart", dst_id = "targetinvert",
  src_channel = 1, dst_channel = 2)
```

`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.ce` for `LearnerClassifAvg` and `regr.mse` for `LearnerRegrAvg`). Learned weights can be obtained from `$model`. This Learner implements and generalizes an approach proposed in LeDell (2015) that uses non-linear optimization in order to learn base-learner weights that optimize a given performance metric (e.g `AUC`). The approach is similar but not exactly the same as the one implemented as `AUC` in the **SuperLearner** R package (when `metric` is `"classif.auc"`). For a more detailed analysis and the general idea, the reader is referred to LeDell (2015).

Note, that weights always sum to 1 by division by `sum(weights)` before weighting incoming features.

## Usage

```
mlr_learners_classif.avg

mlr_learners_regr.avg
```

## Format

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

## Parameters

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

- `measure` :: `Measure` | `character`
  `Measure` to optimize for. Will be converted to a `Measure` in case it is `character`. Initialized to `"classif.ce"`, i.e. misclassification error for classification and `"regr.mse"`, i.e. mean squared error for regression.
- `optimizer` :: `Optimizer` | `character(1)`
  `Optimizer` used to find optimal thresholds. If `character`, converts to `Optimizer` via `opt`. Initialized to `OptimizerNLoptr`. Nloptr hyperparameters are initialized to `xtol_rel = 1e-8`, `algorithm = "NLOPT_LN_COBYLA"` and equal initial weights for each learner. For more fine-grained control, it is recommended to supply a instantiated `Optimizer`.
- `log_level` :: `character(1)` | `integer(1)`
  Set a temporary log-level for `lgr::get_logger("bbotk")`. Initialized to: "warn".

### Methods

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

### References

LeDell, Erin (2015). *Scalable Ensemble Learning and Computationally Efficient Variance Estimation*. Ph.D. thesis, UC Berkeley.

### See Also

Other Learners: `mlr_learners_graph`

Other Ensembles: `PipeOpEnsemble`, `mlr_pipeops_classifavg`, `mlr_pipeops_ovrunite`, `mlr_pipeops_regravg`

---

## `mlr_learners_graph` *Encapsulate a Graph as a Learner*

---

### 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.

The `predict_type` of a `GraphLearner` can be obtained or set via it's `predict_type` active binding. Setting a new predict type will try to set the `predict_type` in all relevant `PipeOp` / `Learner` encapsulated within the `Graph`. Similarly, the predict_type of a Graph will always be the smallest denominator in the `Graph`.

A `GraphLearner` is always constructed in an untrained state. When the `graph` argument has a non-`NULL` `$state`, it is ignored.

### Format

`R6Class` object inheriting from `mlr3::Learner`.

### Construction

`GraphLearner$new(graph, id = NULL, param_vals = list(), task_type = NULL, predict_type = NULL)`

- `graph` :: `Graph` | `PipeOp`
  `Graph` to wrap. Can be a `PipeOp`, which is automatically converted to a `Graph`. This argument is usually cloned, unless `clone_graph` is `FALSE`; to access the `Graph` inside `GraphLearner` by-reference, use `$graph`.

- id :: character(1) Identifier of the resulting Learner.
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings . Default list().
- task_type :: character(1)
  What task_type the GraphLearner should have; usually automatically inferred for Graphs that are simple enough.
- predict_type :: character(1)
  What predict_type the GraphLearner should have; usually automatically inferred for Graphs that are simple enough.
- clone_graph :: logical(1)
  Whether to clone graph upon construction. Unintentionally changing graph by reference can lead to unexpected behaviour, so TRUE (default) is recommended. In particular, note that the $state of $graph is set to NULL by reference on construction of GraphLearner, during $train(), and during $predict() when clone_graph is FALSE.

## Fields

Fields inherited from PipeOp, as well as:

- graph :: Graph
  Graph that is being wrapped. This field contains the prototype of the Graph that is being trained, but does *not* contain the model. Use graph_model to access the trained Graph after $train(). Read-only.
- graph_model :: Learner
  Graph that is being wrapped. This Graph contains a trained state after $train(). Read-only.
- internal_tuned_values :: named list() or NULL
  The internal tuned parameter values collected from all PipeOps. NULL is returned if the learner is not trained or none of the wrapped learners supports internal tuning.
- internal_valid_scores :: named list() or NULL
  The internal validation scores as retrieved from the PipeOps. The names are prefixed with the respective IDs of the PipeOps. NULL is returned if the learner is not trained or none of the wrapped learners supports internal validation.
- validate :: numeric(1), "predefined", "test" or NULL
  How to construct the validation data. This also has to be configured for the individual PipeOps such as PipeOpLearner, see set_validate.GraphLearner. For more details on the possible values, see mlr3::Learner.
- marshaled :: logical(1)
  Whether the learner is marshaled.

## Methods

- marshal(...)
  (any) -> self
  Marshal the model.
- unmarshal(...)
  (any) -> self
  Unmarshal the model.

### Internals

as_graph() is called on the graph argument, so it can technically also be a list of things, which is automatically converted to a Graph via gunion(); however, this will usually not result in a valid Graph that can work as a Learner. graph can furthermore be a Learner, which is then automatically wrapped in a Graph, which is then again wrapped in a GraphLearner object; this usually only adds overhead and is not recommended.

### See Also

Other Learners: mlr_learners_avg

### Examples

```
library("mlr3")

graph = po("pca") %>>% lrn("classif.rpart")

lr = GraphLearner$new(graph)
lr = as_learner(graph)  # equivalent

lr$train(tsk("iris"))

lr$graph$state  # untrained version!
# The following is therefore NULL:
lr$graph$pipeops$classif.rpart$learner_model$model

# To access the trained model from the PipeOpLearner's Learner, use:
lr$graph_model$pipeops$classif.rpart$learner_model$model

# Feature importance (of principal components):
lr$graph_model$pipeops$classif.rpart$learner_model$importance()
```

---

## 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, PipeOp, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_graphs, mlr_pipeops_updatetarget

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Dictionaries: mlr_graphs

## 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` *Box-Cox Transformation of Numeric Features*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputeconstant`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscale`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

## Examples

```
library("mlr3")

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

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

pop$state
```

mlr_pipeops_branch *Path Branching*

## 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 an integer. If options is a character, it determines the names of channels directly. The $selection parameter will then be factorial.
- 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 Channels

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 `private$.train()` or `private$.predict()`, 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

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `mlr_pipeops_boxcox`, `mlr_pipeops_chunk`, `mlr_pipeops_classbalancing`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputeconstant`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscale`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

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 *Chunk Input into Multiple Outputs*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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` *Class Balancing*

---

## 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 rows of the resulting task. In case the data is upsampled and `shuffle = FALSE`, the resulting task will have the original rows (which were not removed in downsampling) in the original order, followed by all newly added rows 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

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `mlr_pipeops_boxcox`, `mlr_pipeops_branch`, `mlr_pipeops_chunk`, `mlr_pipeops_classifavg`, `mlr_pipeops_classweights`, `mlr_pipeops_colapply`, `mlr_pipeops_collapsefactors`, `mlr_pipeops_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`,

mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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

*Majority Vote Prediction*

---

## Description

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

Always returns a "prob" prediction, regardless of the incoming Learner's $predict_type. The label of the class with the highest predicted probability is selected as the "response" prediction. 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.

If ‘

## Format

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

## Construction

```
PipeOpClassifAvg$new(innum = 0, collect_multiplicity = FALSE, 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.
- `collect_multiplicity` :: `logical(1)`
  If `TRUE`, the input is a `Multiplicity` collecting channel. This means, a `Multiplicity` input, instead of multiple normal inputs, is accepted and the members are aggregated. This requires `innum` to be 0. Default is `FALSE`.
- `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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Multiplicity PipeOps: Multiplicity(), PipeOpEnsemble, mlr_pipeops_featureunion, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_regravg, mlr_pipeops_replicate

Other Ensembles: PipeOpEnsemble, mlr_learners_avg, mlr_pipeops_ovrunite, mlr_pipeops_regravg

## Examples

```
library("mlr3")

# Simple Bagging
gr = ppl("greplicate",
  po("subsample") %>>%
  po("learner", lrn("classif.rpart")),
  n = 3
) %>>%
  po("classifavg")

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

---

`mlr_pipeops_classweights`

*Class Weights for Sample Weighting*

---

## Description

Adds a class weight column to the Task that different Learners 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_colapply, mlr_pipeops_collapsefactors, mlr_pipeops_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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 *Apply a Function to each Column of a Task*

## Description

Applies a function to each column of a task. Use the `affect_columns` parameter inherited from `PipeOpTaskPreprocSimple` 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 `PipeOpTaskPreprocSimple`/`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 `PipeOpTaskPreprocSimple`.

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 `PipeOpTaskPreprocSimple`.

## Parameters

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

- `applicator` :: `function`
  Function to apply to each column of the task. The return value should be a `vector` of the same length as the input, i.e., the function vectorizes over the input. A typical example would be `as.numeric`.
  The return value can also be a `matrix`, `data.frame`, or `data.table`. In this case, the length

of the input must match the number of returned rows. The names of the resulting features of the output Task is based on the (column) name(s) of the return value of the applicator function, prefixed with the original feature name separated by a dot (.). Use Vectorize to create a vectorizing function from any function that ordinarily only takes one element input.

## Internals

Calls map on the data, using the value of applicator as f. and coerces the output via as.data.table.

## Fields

Only fields inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

## Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_collapsefacto mlr_pipeops_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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
f1 = function(x) {
  # we could use `ifelse` here, but that is not the point
```

```
  if (x > 1) {
    "a"
  } else {
    "b"
  }
}
poca$param_set$values$applicator = Vectorize(f1)
poca$train(list(task))[[1]]$data()

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

# function returning multiple columns
f2 = function(x) {
  cbind(floor = floor(x), ceiling = ceiling(x))
}
poca$param_set$values$applicator = f2
poca$param_set$values$affect_columns = selector_all()
poca$train(list(task))[[1]]$data()
```

---

`mlr_pipeops_collapsefactors`
*Collapse Factors*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, mlr_pipeops_boxcox, mlr_pipeops_branch, mlr_pipeops_chunk, mlr_pipeops_classbalancing, mlr_pipeops_classifavg, mlr_pipeops_classweights, mlr_pipeops_colapply, mlr_pipeops_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin,

mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs,
mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign,
mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale
mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch,
mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

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

---

# mlr_pipeops_colroles *Change Column Roles of a Task*

---

## Description

Changes the column roles of the input Task according to new_role.

## Format

R6Class object inheriting from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

## Construction

```
PipeOpColRoles$new(id = "colroles", param_vals = list())
```

- id :: character(1)
  Identifier of resulting object, default "colroles".
- 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 column roles according to new_role.

## 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:

- new_role :: list
  Named list of new column roles. The names must match the column names of the input task that will later be trained/predicted on. Each entry of the list must contain a character vector with possible values of mlr_reflections$task_col_roles. If the value is given as character(), the column will be dropped from the input task. Changing the role of a column results in this column loosing its previous role(s). Setting a new target variable or changing the role of an existing target variable is not supported.

## Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

task = tsk("boston_housing")
pop = po("colroles", param_vals = list(
  new_role = list(town = c("order", "feature"))
))

pop$train(list(task))
```

mlr_pipeops_copy *Copy Input Multiple Times*

## 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

```
PipeOpCopy$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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

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_datefeatures`

*Preprocess Date Features*

---

## Description

Based on `POSIXct` columns of the data, a set of date related features is computed and added to the feature set of the output task. If no `POSIXct` column is found, the original task is returned unaltered. This functionality is based on the `add_datepart()` and `add_cyclic_datepart()` functions from the `fastai` library. If operation on only particular `POSIXct` columns is requested, use the `affect_columns` parameter inherited from `PipeOpTaskPreprocSimple`.

If `cyclic = TRUE`, cyclic features are computed for the features `"month"`, `"week_of_year"`, `"day_of_year"`, `"day_of_month"`, `"day_of_week"`, `"hour"`, `"minute"` and `"second"`. This means that for each feature `x`, two additional features are computed, namely the sine and cosine transformation of `2 * pi * x / max_x` (here `max_x` is the largest possible value the feature could take on + 1, assuming the lowest possible value is given by 0, e.g., for hours from 0 to 23, this is 24). This is useful to respect the cyclical nature of features such as seconds, i.e., second 21 and second 22 are one second apart, but so are second 60 and second 1 of the next minute.

## Format

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

## Construction

```
PipeOpDateFeatures$new(id = "datefeatures", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"datefeatures"`.
- `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 PipeOpTaskPreprocSimple.

The output is the input Task with date-related features computed and added to the feature set of the output task and the POSIXct columns of the data removed from the feature set (depending on the value of keep_date_var).

### State

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

### Parameters

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

- keep_date_var :: logical(1)
  Should the POSIXct columns be kept as features? Default FALSE.
- cyclic :: logical(1)
  Should cyclic features be computed? See Internals. Default FALSE.
- year :: logical(1)
  Should the year be extracted as a feature? Default TRUE.
- month :: logical(1)
  Should the month be extracted as a feature? Default TRUE.
- week_of_year :: logical(1)
  Should the week of the year be extracted as a feature? Default TRUE.
- day_of_year :: logical(1)
  Should the day of the year be extracted as a feature? Default TRUE.
- day_of_month :: logical(1)
  Should the day of the month be extracted as a feature? Default TRUE.
- day_of_week :: logical(1)
  Should the day of the week be extracted as a feature? Default TRUE.
- hour :: logical(1)
  Should the hour be extracted as a feature? Default TRUE.
- minute :: logical(1)
  Should the minute be extracted as a feature? Default TRUE.
- second :: logical(1)
  Should the second be extracted as a feature? Default TRUE.
- is_day :: logical(1)
  Should a feature be extracted indicating whether it is day time (06:00am - 08:00pm)? Default TRUE.

### Internals

The cyclic feature transformation always assumes that values range from 0, so some values (e.g. day of the month) are shifted before sine/cosine transform.

## Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

## Fields

Only fields inherited from PipeOpTaskPreproc/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")
dat = iris
set.seed(1)
dat$date = sample(seq(as.POSIXct("2020-02-01"), to = as.POSIXct("2020-02-29"), by = "hour"),
 size = 150L)
task = TaskClassif$new("iris_date", backend = dat, target = "Species")
pop = po("datefeatures", param_vals = list(cyclic = FALSE, minute = FALSE, second = FALSE))
pop$train(list(task))
pop$state
```

---

# mlr_pipeops_encode *Factor Encoding*

---

## Description

Encodes columns of type `factor` 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.

`character`-type features can be encoded by converting them `factor` features first, using `ppl("convert_types", "character", "factor")`.

## 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` and `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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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()

# converting character-columns
data_chr = data.table::data.table(x = factor(letters[1:3]), y = letters[1:3])
task_chr = TaskClassif$new("task_chr", data_chr, "x")

goe = ppl("convert_types", "character", "factor") %>>% po("encode")

goe$train(task_chr)[[1]]$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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin,

mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs,
mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign,
mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale
mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch,
mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`,

mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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

*Aggregate Features from Multiple Inputs*

---

## 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.

If input tasks share some feature names but these features are not identical an error is thrown. This check is performed by first comparing the features names and if duplicates are found, also the values of these possibly duplicated features. True duplicated features are only added a single time to the output task.

## Format

R6Class object inheriting from PipeOp.

## Construction

```
PipeOpFeatureUnion$new(innum = 0, collect_multiplicity = FALSE, 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.
- collect_multiplicity :: logical(1)
  If TRUE, the input is a Multiplicity collecting channel. This means, a Multiplicity input, instead of multiple normal inputs, is accepted and the members are aggregated. This requires innum to be 0. Default is FALSE.
- 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Multiplicity PipeOps: Multiplicity(), PipeOpEnsemble, mlr_pipeops_classifavg, mlr_pipeops_multiplicity mlr_pipeops_multiplicityimply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_regravg, mlr_pipeops_replicate

## Examples

```
library("mlr3")

task1 = tsk("iris")
gr = gunion(list(
  po("nop"),
  po("pca")
)) %>>% po("featureunion")

gr$train(task1)
```

```
task2 = tsk("iris")
task3 = tsk("iris")
po = po("featureunion", innum = c("a", "b"))

po$train(list(task2, task3))
```

---

`mlr_pipeops_filter` *Feature Filtering*

---

## 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. This argument is always cloned; to access the `Filter` inside `PipeOpFilter` by-reference, use `$filter`.

- `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, cutoff, and permuted.
- filter.frac :: numeric(1)
  Fraction of features to keep. Mutually exclusive with nfeat, cutoff, and permuted.
- filter.cutoff :: numeric(1)
  Minimum value of filter heuristic for which to keep features. Mutually exclusive with nfeat, frac, and permuted.
- filter.permuted :: integer(1)
  If this parameter is set, a random permutation of each feature is added to the task before applying the filter. All features selected before the permuted-th permuted features is selected are kept. This is similar to the approach in Wu (2007) and Thomas (2017). Mutually exclusive with nfeat, frac, and cutoff.

Note that at least one of filter.nfeat, filter.frac, filter.cutoff, and filter.permuted must be given.

## Internals

This does *not* use the $.select_cols feature of PipeOpTaskPreproc to select only features compatible with the Filter; instead the whole Task is used by private$.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.

## References

Wu Y, Boos DD, Stefanski LA (2007). "Controlling Variable Selection by the Addition of Pseudovariables." *Journal of the American Statistical Association*, **102**(477), 235–243. doi:10.1198/016214506000000843.

Thomas J, Hepp T, Mayr A, Bischl B (2017). "Probing for Sparse and Fast Variable Selection with Model-Based Boosting." *Computational and Mathematical Methods in Medicine*, **2017**, 1–8. doi:10.1155/2017/1421409.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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` *Fix Factor Levels*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

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

---

# mlr_pipeops_histbin *Split Numeric Features into Equally Spaced Bins*

---

## Description

Splits numeric features into equally spaced bins. See `graphics::hist()` for details. Values that fall out of the training data range during prediction are binned with the lowest / highest bin respectively.

### 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:

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

### Parameters

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

- breaks :: 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

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

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

pop$state
```

---

# mlr_pipeops_ica *Independent Component Analysis*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_imputeconstant`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscale`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

## Examples

```
library("mlr3")

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

task$data()
```

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

pop$state
```

---

`mlr_pipeops_imputeconstant` *Impute Features by a Constant*

---

## Description

Impute features by a constant value.

## Format

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

## Construction

```
PipeOpImputeConstant$new(id = "imputeconstant", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"imputeconstant"`.
- `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 `PipeOpImpute`.

The output is the input `Task` with all affected features missing values imputed by the value of the `constant` parameter.

## State

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

The `$state$model` contains the value of the `constant` parameter that is used for imputation.

## Parameters

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

- `constant` :: `atomic(1)`
  The constant value that should be used for the imputation, atomic vector of length 1. The atomic mode must match the type of the features that will be selected by the `affect_columns` parameter and this will be checked during imputation. Initialized to `".MISSING"`.
- `check_levels` :: `logical(1)`
  Should be checked whether the `constant` value is a valid level of factorial features (i.e., it already is a level)? Raises an error if unsuccesful. This check is only performed for factorial features (i.e., `factor`, `ordered`; skipped for `character`). Initialized to `TRUE`.

## Internals

Adds an explicit new level to `factor` and `ordered` features, but not to `character` features, if `check_levels` is `FALSE` and the level is not already present.

## Methods

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

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `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_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityin`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscalerange`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

Other Imputation PipeOps: `PipeOpImpute`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`

## Examples

```
library("mlr3")

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

# impute missing values of the numeric feature "glucose" by the constant value -999
po = po("imputeconstant", param_vals = list(
  constant = -999, affect_columns = selector_name("glucose"))
)
new_task = po$train(list(task = task))[[1]]
new_task$missings()
new_task$data(cols = "glucose")[[1]]
```

`mlr_pipeops_imputehist` *Impute Numerical Features by Histogram*

## Description

Impute numerical features by histogram.

During training, a histogram is fitted using R's `hist()` function. The fitted histogram is then sampled from for imputation. This is an approximation to sampling from the empirical training data distribution (i.e. sampling from training data with replacement), but is much more memory efficient for large datasets, since the `$state` does not need to save the training data.

## 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 `PipeOpImpute`.

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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityir mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Imputation PipeOps: PipeOpImpute, mlr_pipeops_imputeconstant, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, 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_imputelearner`

*Impute Features by Fitting a Learner*

---

## Description

Impute features by fitting a Learner for each feature. Uses the features indicated by the context_columns parameter as features to train the imputation Learner. Note this parameter is part of the PipeOpImpute base class and explained there.

Additionally, only features supported by the learner can be imputed; i.e. learners of type regr can only impute features of type integer and numeric, while classif can impute features of type factor, ordered and logical.

The Learner used for imputation is trained on all context_columns; if these contain missing values, the Learner typically either needs to be able to handle missing values itself, or needs to do its own imputation (see examples).

## Format

R6Class object inheriting from PipeOpImpute/PipeOp.

## Construction

```
PipeOpImputeLearner$new(learner, id = NULL, param_vals = list())
```

- id :: character(1)
  Identifier of resulting object, default "impute.", followed by the id of the Learner.
- learner :: Learner | character(1) Learner to wrap, or a string identifying a Learner in the mlr3::mlr_learners Dictionary. The Learner usually needs to be able to handle missing values, i.e. have the missings property, unless care is taken that context_columns do not contain missings; see examples.
  This argument is always cloned; to access the Learner inside PipeOpImputeLearner by-reference, use $learner.

- 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 PipeOpImpute.

The output is the input Task with missing values from all affected features imputed by the trained model.

## State

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

The $state$models is a named list of models created by the Learner's $.train() function for each column. If a column consists of missing values only during training, the model is 0 or the levels of the feature; these are used for sampling during prediction.

This state is given the class "pipeop_impute_learner_state".

## Parameters

The parameters are the parameters inherited from PipeOpImpute, in addition to the parameters of the Learner used for imputation.

## Internals

Uses the $train and $predict functions of the provided learner. Features that are entirely NA are imputed as 0 or randomly sampled from available (factor / logical) levels.

The Learner does *not* necessarily need to handle missing values in cases where context_columns is chosen well (or there is only one column with missing values present).

## Fields

Fields inherited from PipeOpTaskPreproc/PipeOp, as well as:

- learner :: Learner
  Learner that is being wrapped. Read-only.
- learner_models :: list of Learner | NULL
  Learner that is being wrapped. This list is named by features for which a Learner was fitted, and contains the same Learner, but with different respective models for each feature. If this PipeOp is not trained, this is an empty list. For features that were entirely NA during training, the list contains NULL elements.

## Methods

Only methods inherited from PipeOpImpute/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityin
mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Imputation PipeOps: PipeOpImpute, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample

## Examples

```
library("mlr3")

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

po = po("imputelearner", lrn("regr.rpart"))
new_task = po$train(list(task = task))[[1]]
new_task$missings()

# '$state' of the "regr.rpart" Learner, trained to predict the 'mass' column:
po$state$model$mass

library("mlr3learners")
# to use the "regr.kknn" Learner, prefix it with its own imputation method!
# The "imputehist" PipeOp is used to train "regr.kknn"; predictions of this
# trained Learner are then used to impute the missing values in the Task.
po = po("imputelearner",
  po("imputehist") %>>% lrn("regr.kknn")
)

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

---

# mlr_pipeops_imputemean

*Impute Numerical Features by their Mean*

---

## 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 `PipeOpImpute`.

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

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`
`mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityin`
`mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_renamecolumns`
`mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscalerange`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

Other Imputation PipeOps: PipeOpImpute, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, 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`

*Impute Numerical Features by their Median*

---

## 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 `PipeOpImpute`.

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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityin
mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Imputation PipeOps: PipeOpImpute, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, 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_imputemode` *Impute Features by their Mode*

## Description

Impute features by their mode. Supports factors as well as logical and numerical features. If multiple modes are present then imputed values are sampled randomly from them.

## Format

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

## Construction

```
PipeOpImputeMode$new(id = "imputemode", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"imputemode"`.
- `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 `PipeOpImpute`.

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

## State

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

The `$state$model` is a named `list` of a vector of length one of the type of the feature, indicating the mode of the respective feature.

## Parameters

The parameters are the parameters inherited from `PipeOpImpute`.

## Internals

Features that are entirely `NA` are imputed as the following: For `factor` or `ordered`, random levels are sampled uniformly at random. For logicals, `TRUE` or `FALSE` are sampled uniformly at random. Numerics and integers are imputed as `0`.

Note that every random imputation is drawn independently, so different values may be imputed if multiple values are missing.

## Methods

Only methods inherited from PipeOpImpute/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityin
mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Imputation PipeOps: PipeOpImpute, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputeoor, mlr_pipeops_imputesample

## Examples

```
library("mlr3")

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

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

po$state$model
```

---

`mlr_pipeops_imputeoor` *Out of Range Imputation*

---

## Description

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

Impute numerical features by constant values shifted below the minimum or above the maximum by using $min(x) - offset - multiplier * diff(range(x))$ or $max(x) + offset + multiplier * diff(range(x))$.

This type of imputation is especially sensible in the context of tree-based methods, see also Ding & Simonoff (2010).

## Format

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

## Construction

```
PipeOpImputeOOR$new(id = "imputeoor", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"imputeoor"`.
- `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 `PipeOpImpute`.

The output is the input `Task` with all affected features having missing values imputed as described above.

## State

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

The `$state$model` contains either `".MISSING"` used for `character` and `factor` (also `ordered`) features or `numeric(1)` indicating the constant value used for imputation of `integer` and `numeric` features.

## Parameters

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

- `min` :: `logical(1)`
  Should `integer` and `numeric` features be shifted below the minimum? Initialized to TRUE. If FALSE they are shifted above the maximum. See also the description above.
- `offset` :: `numeric(1)`
  Numerical non-negative offset as used in the description above for `integer` and `numeric` features. Initialized to 1.
- `multiplier` :: `numeric(1)`
  Numerical non-negative multiplier as used in the description above for `integer` and `numeric` features. Initialized to 1.

## Internals

Adds an explicit new `level()` to `factor` and `ordered` features, but not to `character` features. For `integer` and `numeric` features uses the `min`, `max`, `diff` and `range` functions. `integer` and `numeric` features that are entirely `NA` are imputed as `0`.

## Methods

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

## References

Ding Y, Simonoff JS (2010). "An Investigation of Missing Data Methods for Classification Trees Applied to Binary Response Data." *Journal of Machine Learning Research*, **11**(6), 131-170. https://jmlr.org/papers/v11/ding10a.html.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`
`mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityin`
`mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_renamecolumns`
`mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscalerange`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

Other Imputation PipeOps: `PipeOpImpute`, `mlr_pipeops_imputeconstant`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputesample`

## Examples

```
library("mlr3")
set.seed(2409)
data = tsk("pima")$data()
data$y = factor(c(NA, sample(letters, size = 766, replace = TRUE), NA))
data$z = ordered(c(NA, sample(1:10, size = 767, replace = TRUE)))
task = TaskClassif$new("task", backend = data, target = "diabetes")
task$missings()
po = po("imputeoor")
```

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

---

`mlr_pipeops_imputesample`
*Impute Features by Sampling*

---

## 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 `PipeOpImpute`.

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 following: For `factor` or `ordered`, random levels are sampled uniformly at random. For logicals, `TRUE` or `FALSE` are sampled uniformly at random. Numerics and integers are imputed as `0`.

## Methods

Only methods inherited from PipeOpImpute/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityir mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Imputation PipeOps: PipeOpImpute, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor

## 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 *Kernelized Principle Component Analysis*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityir
mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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 *Wrap a Learner into a PipeOp*

---

## 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 = NULL, param_vals = list())
```

- learner :: Learner | character(1) Learner to wrap, or a string identifying a Learner in the mlr3::mlr_learners Dictionary. This argument is always cloned; to access the Learner inside PipeOpLearner by-reference, use $learner.
- id :: character(1) Identifier of the resulting object, internally 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() 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.
- learner_model :: Learner
  Learner that is being wrapped. This learner contains the model if the PipeOp is trained. Read-only.
- validate :: "predefined" or NULL
  This field can only be set for Learners that have the "validation" property. Setting the field to "predefined" means that the wrapped Learner will use the internal validation task, otherwise it will be ignored. Note that specifying *how* the validation data is created is possible via the $validate field of the GraphLearner. For each PipeOp it is then only possible to either use it ("predefined") or not use it (NULL). Also see set_validate.GraphLearner for more information.
- internal_tuned_values :: named list() or NULL
  The internally tuned values if the wrapped Learner supports internal tuning, NULL otherwise.
- internal_valid_scores :: named list() or NULL
  The internal validation scores if the wrapped Learner supports internal validation, NULL otherwise.

## Methods

Methods inherited from PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityin
mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample,

mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

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

*Wrap a Learner into a PipeOp with Cross-validated Predictions as Features*

---

## 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 = NULL, param_vals = list())
```

- learner :: Learner
  Learner to use for cross validation / prediction, or a string identifying a Learner in the mlr3::mlr_learners Dictionary. This argument is always cloned; to access the Learner inside PipeOpLearnerCV by-reference, use $learner.

- id :: character(1) Identifier of the resulting object, internally 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() 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.

This state is given the class "pipeop_learner_cv_state".

### 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"` and `"insample"`. `"insample"` generates predictions with the model trained on all training data.
- `resampling.folds` :: `numeric(1)`
  Number of cross validation folds. Initialized to 3. Only used for `resampling.method = "cv"`.
- `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 `PipeOp`, as well as:

- `learner` :: `Learner`
  `Learner` that is being wrapped. Read-only.
- `learner_model` :: `Learner`
  `Learner` that is being wrapped. This learner contains the model if the `PipeOp` is trained. Read-only.

## Methods

Methods inherited from `PipeOpTaskPreproc`/`PipeOp`.

## See Also

https://mlr-org.com/pipeops.html

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` *Add Missing Indicator Columns*

---

## 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).

Note the `affect_columns` is initialized with `selector_invert(selector_type(c("factor", "ordered", "character")))`, since missing values in factorial columns are often indicated by out-of-range imputation (`PipeOpImputeOOR`).

## 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 "factor" (default), "integer", "logical", "numeric".

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityin
mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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` *Transform Columns by Constructing a Model Matrix*

---

## 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 1s 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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_multiplicityexply *Explicate a Multiplicity*

## Description

Explicate a Multiplicity by turning the input Multiplicity into multiple outputs.

This PipeOp has multiple output channels; the members of the input Multiplicity are forwarded each along a single edge. Therefore, only multiplicities with exactly as many members as outnum are accepted.

Note that Multiplicity is currently an experimental features and the implementation or UI may change.

## Format

R6Class object inheriting from PipeOp.

## Construction

```
PipeOpMultiplicityExply$new(outnum , id = "multiplicityexply", param_vals = list())
```

- outnum :: numeric(1) | character
  Determines the number of output channels.
- id :: character(1)
  Identifier of the resulting object, default "multiplicityexply".
- 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

PipeOpMultiplicityExply has a single input channel named "input", collecting a Multiplicity of type any ("[*]") both during training and prediction.

PipeOpMultiplicityExply has multiple output channels depending on the outnum construction argument, named "output1", "output2" returning the elements of the unclassed input Multiplicity.

## State

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

## Parameters

PipeOpMultiplicityExply has no Parameters.

## Internals

`outnum` should match the number of elements of the unclassed input `Multiplicity`.

## Fields

Only fields inherited from `PipeOp`.

## Methods

Only methods inherited from `PipeOp`.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscalerange`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

Other Multiplicity PipeOps: `Multiplicity()`, `PipeOpEnsemble`, `mlr_pipeops_classifavg`, `mlr_pipeops_featureunion`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_regravg`, `mlr_pipeops_replicate`

Other Experimental Features: `Multiplicity()`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_replicate`

## Examples

```
library("mlr3")
task1 = tsk("iris")
task2 = tsk("mtcars")
po = po("multiplicityexply", outnum = 2)
po$train(list(Multiplicity(task1, task2)))
po$predict(list(Multiplicity(task1, task2)))
```

`mlr_pipeops_multiplicityimply` *Implicate a Multiplicity*

## Description

Implicate a `Multiplicity` by returning the input(s) converted to a `Multiplicity`.

This `PipeOp` has multiple input channels; all inputs are collected into a `Multiplicity` and then are forwarded along a single edge, causing the following `PipeOp`s to be called multiple times, once for each `Multiplicity` member.

Note that `Multiplicity` is currently an experimental features and the implementation or UI may change.

## Format

`R6Class` object inheriting from `PipeOp`.

## Construction

```
PipeOpMultiplicityImply$new(innum = 0, id = "multiplicityimply", param_vals = list())
```

- `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`.
- `id` :: `character(1)`
  Identifier of the resulting object, default `"multiplicityimply"`.
- `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

`PipeOpMultiplicityImply` 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 any input (`"*"`) both during training and prediction.

`PipeOpMultiplicityImply` has one output channel named `"output"`, emitting a `Multiplicity` of type any (`"[*]"`), i.e., returning the input(s) converted to a `Multiplicity` both during training and prediction.

## State

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

## Parameters

PipeOpMultiplicityImply has no Parameters.

## Internals

If innum is not numeric, e.g., a character, the output Multiplicity will be named based on the input channel names

## Fields

Only fields inherited from PipeOp.

## Methods

Only methods inherited from PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Multiplicity PipeOps: Multiplicity(), PipeOpEnsemble, mlr_pipeops_classifavg, mlr_pipeops_featureunior mlr_pipeops_multiplicityexply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_regravg, mlr_pipeops_replicate

Other Experimental Features: Multiplicity(), mlr_pipeops_multiplicityexply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_replicate

## Examples

```
library("mlr3")
task1 = tsk("iris")
task2 = tsk("mtcars")
po = po("multiplicityimply")
```

```
po$train(list(task1, task2))
po$predict(list(task1, task2))
```

---

`mlr_pipeops_mutate` *Add Features According to Expressions*

---

## 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.

Note that the `formulas` in `mutation` are evaluated sequentially. This allows for using variables that were constructed during evaluation of a previous formula. However, if existing features are changed, precedence is given to the original ones before the newly constructed ones.

## Fields

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

## Methods

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

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`
`mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`
`mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscale`
`mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

## Examples

```
library("mlr3")

constant = 1
pom = po("mutate")
pom$param_set$values$mutation = list(
  Sepal.Length_plus_constant = ~ Sepal.Length + constant,
  Sepal.Area = ~ Sepal.Width * Sepal.Length,
  Petal.Area = ~ Petal.Width * Petal.Length,
  Sepal.Area_plus_Petal.Area = ~ Sepal.Area + Petal.Area
)
```

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

---

`mlr_pipeops_nmf` *Non-negative Matrix Factorization*

---

## Description

Extracts non-negative components from data by performing non-negative matrix factorization. Only affects non-negative numerical features. See `nmf()` for details.

## Format

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

## Construction

```
PipeOpNMF$new(id = "nmf", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"nmf"`.
- `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 non-negative components.

## State

The `$state` is a named `list` with the `$state` elements inherited from `PipeOpTaskPreproc`, as well as the elements of the object returned by `nmf()`.

## Parameters

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

- `rank` :: `integer(1)`
  Factorization rank, i.e., number of components. Initialized to `2`. See `nmf()`.
- `method` :: `character(1)`
  Specification of the NMF algorithm. Initialized to `"brunet"`. See `nmf()`.
- `seed` :: `character(1)` | `integer(1)` | `list()` | object of class `NMF` | `function()`
  Specification of the starting point. See `nmf()`.

- nrun :: integer(1)
  Number of runs to performs. Default is 1. More than a single run allows for the computation of a consensus matrix which will also be stored in the $state. See nmf().
- debug :: logical(1)
  Whether to toggle debug mode. Default is FALSE. See nmf().
- keep.all :: logical(1)
  Whether all factorizations are to be saved and returned. Default is FALSE. Only has an effect if nrun > 1. See nmf().
- parallel :: character(1) | integer(1) | logical(1)
  Specification of parallel handling if nrun > 1. Initialized to FALSE, as it is recommended to use mlr3's future-based parallelization. See nmf().
- parallel.required :: character(1) | integer(1) | logical(1)
  Same as parallel, but an error is thrown if the computation cannot be performed in parallel or with the specified number of processors. Initialized to FALSE, as it is recommended to use mlr3's future-based parallelization. See nmf().
- shared.memory :: logical(1)
  Whether shared memory should be enabled. See nmf().
- simplifyCB :: logical(1)
  Whether callback results should be simplified. Default is TRUE. See nmf().
- track :: logical(1)
  Whether error tracking should be enabled. Default is FALSE. See nmf().
- verbose :: integer(1) | logical(1)
  Specification of verbosity. Default is FALSE. See nmf().
- pbackend :: character(1) | integer(1) | NULL
  Specification of the parallel backend. It is recommended to use mlr3's future-based parallelization. See nmf().
- callback | function()
  Callback function that is called after each run (if nrun > 1). See nmf().

## Internals

Uses the nmf() function as well as basis(), coef() and ginv().

## Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons

mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
if (requireNamespace("NMF")) {
library("mlr3")

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

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

pop$state
}
```

---

# mlr_pipeops_nop *Simply Push Input Forward*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale
mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

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_ovrsplit` *Split a Classification Task into Binary Classification Tasks*

---

## Description

Splits a classification Task into several binary classification Tasks to perform "One vs. Rest" classification. This works in combination with `PipeOpOVRUnite`.

For each target level a new binary classification Task is constructed with the respective target level being the positive class and all other target levels being the new negative class `"rest"`.

This `PipeOp` creates a `Multiplicity`, which means that subsequent `PipeOp`s are executed multiple times, once for each created binary Task, until a `PipeOpOVRUnite` is reached.

Note that `Multiplicity` is currently an experimental features and the implementation or UI may change.

## Format

`R6Class` inheriting from `PipeOp`.

## Construction

```
PipeOpOVRSplit$new(id = "ovrsplit", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of the resulting object, default `"ovrsplit"`.
- `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

PipeOpOVRSplit has one input channel named "input" taking a TaskClassif both during training and prediction.

PipeOpOVRSplit has one output channel named "output" returning a Multiplicity of TaskClassifs both during training and prediction, i.e., the newly constructed binary classification Tasks.

### State

The $state contains the original target levels of the TaskClassif supplied during training.

### Parameters

PipeOpOVRSplit has no parameters.

### Internals

The original target levels stored in the $state are also used during prediction when creating the new binary classification Tasks.

The names of the element of the output Multiplicity are given by the levels of the target.

If a target level "rest" is present in the input TaskClassif, the negative class will be labeled as "rest." (using as many "."' postfixes needed to yield a valid label).

Should be used in combination with PipeOpOVRUnite.

### Fields

Only fields inherited from PipeOp.

### Methods

Only methods inherited from PipeOp.

### See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign,

mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale
mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch,
mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Multiplicity PipeOps: Multiplicity(), PipeOpEnsemble, mlr_pipeops_classifavg, mlr_pipeops_featureunior
mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_ovrunite,
mlr_pipeops_regravg, mlr_pipeops_replicate

Other Experimental Features: Multiplicity(), mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply,
mlr_pipeops_ovrunite, mlr_pipeops_replicate

## Examples

```
library(mlr3)
task = tsk("iris")
po = po("ovrsplit")
po$train(list(task))
po$predict(list(task))
```

---

# mlr_pipeops_ovrunite *Unite Binary Classification Tasks*

---

## Description

Perform "One vs. Rest" classification by (weighted) majority vote prediction from classification Predictions. This works in combination with PipeOpOVRSplit.

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

Always returns a "prob" prediction, regardless of the incoming Learner's $predict_type. The label of the class with the highest predicted probability is selected as the "response" prediction.

Missing values during prediction are treated as each class label being equally likely.

This PipeOp uses a Multiplicity input, which is created by PipeOpOVRSplit and causes PipeOps on the way to this PipeOp to be called once for each individual binary Task.

Note that Multiplicity is currently an experimental features and the implementation or UI may change.

## Format

R6Class inheriting from PipeOpEnsemble/PipeOp.

## Construction

```
PipeOpOVRUnite$new(id = "ovrunite", param_vals = list())
```

- id :: character(1)
  Identifier of the resulting object, default "ovrunite".
- 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 and PipeOpEnsemble's collect parameter is initialized with TRUE to allow for collecting a Multiplicity input.

### 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$.predict() method.

Should be used in combination with PipeOpOVRSplit.

### Fields

Only fields inherited from PipeOpEnsemble/PipeOp.

### Methods

Only methods inherited from PipeOpEnsemble/PipeOp.

### See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale
mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Ensembles: PipeOpEnsemble, mlr_learners_avg, mlr_pipeops_classifavg, mlr_pipeops_regravg

Other Multiplicity PipeOps: Multiplicity(), PipeOpEnsemble, mlr_pipeops_classifavg, mlr_pipeops_featureunion
mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_ovrsplit,
mlr_pipeops_regravg, mlr_pipeops_replicate

Other Experimental Features: Multiplicity(), mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply,
mlr_pipeops_ovrsplit, mlr_pipeops_replicate

## Examples

```
library(mlr3)
task = tsk("iris")
gr = po("ovrsplit") %>>% lrn("classif.rpart") %>>% po("ovrunite")
gr$train(task)
gr$predict(task)
gr$pipeops$classif.rpart$learner$predict_type = "prob"
gr$predict(task)
```

---

# mlr_pipeops_pca *Principle Component Analysis*

---

## 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 TRUE. 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_proxy, mlr_pipeops_quantilebin,

mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs,
mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign,
mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale
mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch,
mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

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

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

pop$state
```

---

`mlr_pipeops_proxy` *Wrap another PipeOp or Graph as a Hyperparameter*

---

## Description

Wraps another PipeOp or Graph as determined by the `content` hyperparameter. Input is routed through the `content` and the `contents'` output is returned. The `content` hyperparameter can be changed during tuning, this is useful as an alternative to PipeOpBranch.

## Format

Abstract R6Class inheriting from PipeOp.

## Construction

```
PipeOpProxy$new(innum = 0, outnum = 1, id = "proxy", param_vals = list())
```

- `innum` :: `numeric(1)\cr 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.
- `outnum` :: ‘numeric(1)
  Determines the number of output channels.
- `id` :: `character(1)`
  Identifier of resulting object. See `$id` slot of PipeOp.
- `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

PipeOpProxy 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 "...".

PipeOpProxy has multiple output channels depending on the outnum construction argument, named "output1", "output2", ... The output is determined by the output of the content operation (a PipeOp or Graph).

### State

The $state is the trained content PipeOp or Graph.

### Parameters

- content :: PipeOp | Graph
  The PipeOp or Graph that is being proxied (or an object that is converted to a Graph by as_graph()). Defaults to an instance of PipeOpFeatureUnion (combines all input if they are Tasks).

### Internals

The content will internally be coerced to a graph via as_graph() prior to train and predict.

The default value for content is PipeOpFeatureUnion,

### Fields

Fields inherited from PipeOp.

### Methods

Only methods inherited from PipeOp.

### See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs,

mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")
library("mlr3learners")

set.seed(1234)
task = tsk("iris")

# use a proxy for preprocessing and a proxy for learning, i.e.,
# no preprocessing and classif.kknn
g = po("proxy", id = "preproc", param_vals = list(content = po("nop"))) %>>%
  po("proxy", id = "learner", param_vals = list(content = lrn("classif.kknn")))
rr_kknn = resample(task, learner = GraphLearner$new(g), resampling = rsmp("cv", folds = 3))
rr_kknn$aggregate(msr("classif.ce"))

# use pca for preprocessing and classif.rpart as the learner
g$param_set$values$preproc.content = po("pca")
g$param_set$values$learner.content = lrn("classif.rpart")
rr_pca_rpart = resample(task, learner = GraphLearner$new(g), resampling = rsmp("cv", folds = 3))
rr_pca_rpart$aggregate(msr("classif.ce"))
```

---

# mlr_pipeops_quantilebin

*Split Numeric Features into Quantile Bins*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_randomprojec
mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

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

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

pop$state
```

---

`mlr_pipeops_randomprojection` *Project Numeric Features onto a Randomly Sampled Subspace*

---

## Description

Projects numeric features onto a randomly sampled subspace. All numeric features (or the ones selected by `affect_columns`) are replaced by numeric features `PR1`, `PR2`, ... `PRn`

Samples with features that contain missing values result in all `PR1`..`PRn` being NA for that sample, so it is advised to do imputation *before* random projections if missing values can be expected.

## Format

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

## Construction

```
PipeOpRandomProjection$new(id = "randomprojection", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"randomprojection"`.
- `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 affected numeric features projected onto a random subspace.

## State

The `$state` is a named `list` with the `$state` elements inherited from `PipeOpTaskPreproc`, as well as an element `$projection`, a `matrix`.

## Parameters

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

- rank :: integer(1)
  The dimension of the subspace to project onto. Initialized to 1.

## Internals

If there are n (affected) numeric features in the input Task, then $state$projection is a rank x m matrix. The output is calculated as input %*% state$projection.

The random projection matrix is obtained through Gram-Schmidt orthogonalization from a matrix with values standard normally distributed, which gives a distribution that is rotation invariant, as per Eaton: Multivariate Statistics, A Vector Space Approach, Pg. 234.

## Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

task = tsk("iris")
pop = po("randomprojection", rank = 2)

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

pop$state
```

mlr_pipeops_randomresponse *Generate a Randomized Response Prediction*

## Description

Takes in a Prediction of predict_type "prob" (for PredictionClassif) or "se" (for PredictionRegr) and generates a randomized "response" prediction.

For "prob", the responses are sampled according to the probabilities of the input PredictionClassif. For "se", responses are randomly drawn according to the rdistfun parameter (default is rnorm) by using the original responses of the input PredictionRegr as the mean and the original standard errors of the input PredictionRegr as the standard deviation (sampling is done observation-wise).

## Format

R6Class object inheriting from PipeOp.

## Construction

```
PipeOpRandomResponse$new(id = "randomresponse", param_vals = list(), packages = character(0))
```

- id :: character(1)
  Identifier of the resulting object, default "randomresponse".
- 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 all required packages for the private$.predict() methods related to the rdistfun parameter. Default is character(0).

## Input and Output Channels

PipeOpRandomResponse has one input channel named "input", taking NULL during training and a Prediction during prediction.

PipeOpRandomResponse has one output channel named "output", producing NULL during training and a Prediction with random responses during prediction.

## State

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

## Parameters

- rdistfun :: function
  A function for generating random responses when the predict type is "se". This function must accept the arguments n (integerish number of responses), mean (numeric for the mean), and sd (numeric for the standard deviation), and must *vectorize* over mean and sd. Default is rnorm.

## Internals

If the `predict_type` of the input `Prediction` does not match `"prob"` or `"se"`, the input `Prediction` will be returned unaltered.

## Fields

Only fields inherited from `PipeOp`.

## Methods

Only methods inherited from `PipeOp`.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`
`mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconstants`, `mlr_pipeops_renamecolu`
`mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscalerange`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

## Examples

```
library(mlr3)
library(mlr3learners)

task1 = tsk("iris")
g1 = LearnerClassifRpart$new() %>>% PipeOpRandomResponse$new()
g1$train(task1)
g1$pipeops$classif.rpart$learner$predict_type = "prob"
set.seed(2409)
g1$predict(task1)

task2 = tsk("mtcars")
g2 = LearnerRegrLM$new() %>>% PipeOpRandomResponse$new()
g2$train(task2)
g2$pipeops$regr.lm$learner$predict_type = "se"
set.seed(2906)
```

```
g2$predict(task2)
```

---

`mlr_pipeops_regravg` *Weighted Prediction Averaging*

---

## 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, collect_multiplicity = FALSE, 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.
- `collect_multiplicity` :: `logical(1)`
  If `TRUE`, the input is a `Multiplicity` collecting channel. This means, a `Multiplicity` input, instead of multiple normal inputs, is accepted and the members are aggregated. This requires `innum` to be 0. Default is `FALSE`.
- `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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_removeconstants, mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Multiplicity PipeOps: Multiplicity(), PipeOpEnsemble, mlr_pipeops_classifavg, mlr_pipeops_featureunior mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_replicate

Other Ensembles: PipeOpEnsemble, mlr_learners_avg, mlr_pipeops_classifavg, mlr_pipeops_ovrunite

## Examples

```
library("mlr3")

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

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

`mlr_pipeops_removeconstants` *Remove Constant Features*

## Description

Remove constant features from a mlr3::Task. For each feature, calculates the ratio of features which differ from their mode value. All features with 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. Initialized to 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. Initialized to `1e-8`.
- `abs_tol` :: `numeric(1)`
  Absolute tolerance within which to consider a numeric feature constant. Set to 0 to disregard absolute tolerance. Initialized to `1e-8`.

- na_ignore :: logical(1)
  If TRUE, the ratio is calculated after removing all missing values first, so a column can be "constant" even if some but not all values are NA. Initialized to TRUE.

## Fields

Fields inherited from PipeOpTaskPreproc/PipeOp.

## Methods

Methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_renamecolumn
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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_renamecolumns` *Rename Columns*

## Description

Renames the columns of a `Task` both during training and prediction. Uses the `$rename()` mutator of the `Task`.

## Format

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

## Construction

```
PipeOpRenameColumns$new(id = "renamecolumns", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"renamecolumns"`.
- `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 `PipeOpTaskPreprocSimple`.

The output is the input `Task` with the old column names changed to the new ones.

## State

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

## Parameters

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

- `renaming` :: named `character`
  Named `character` vector. The names of the vector specify the old column names that should be changed to the new column names as given by the elements of the vector. Initialized to the empty character vector.
- `ignore_missing` :: `logical(1)`
  Ignore if columns named in `renaming` are not found in the input `Task`. If this is `FALSE`, then names found in `renaming` not found in the `Task` cause an error. Initialized to `FALSE`.

## Internals

Uses the `$rename()` mutator of the `Task` to set the new column names.

## Fields

Only fields inherited from PipeOpTaskPreprocSimple/PipeOp.

## Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

task = tsk("iris")
pop = po("renamecolumns", param_vals = list(renaming = c("Petal.Length" = "PL")))
pop$train(list(task))
```

---

# mlr_pipeops_replicate *Replicate the Input as a Multiplicity*

---

## Description

Replicate the input as a Multiplicity, causing subsequent PipeOps to be executed multiple reps times.

Note that Multiplicity is currently an experimental features and the implementation or UI may change.

### Format

R6Class object inheriting from PipeOp.

### Construction

```
PipeOpReplicate$new(id = "replicate", param_vals = list())
```

- id :: character(1) Identifier of the resulting object, default "replicate".
- 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

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

PipeOpReplicate has one output channel named "output" returning the replicated input as a Multiplicity of type any ("[*]") both during training and prediction.

### State

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

### Parameters

- reps :: numeric(1)
  Integer indicating the number of times the input should be replicated.

### Fields

Only fields inherited from PipeOp.

### Methods

Only methods inherited from PipeOp.

### See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop,

mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Multiplicity PipeOps: Multiplicity(), PipeOpEnsemble, mlr_pipeops_classifavg, mlr_pipeops_featureunior mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_regravg

Other Experimental Features: Multiplicity(), mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite

## Examples

```
library("mlr3")
task = tsk("iris")
po = po("replicate", param_vals = list(reps = 3))
po$train(list(task))
po$predict(list(task))
```

---

# mlr_pipeops_scale *Center and Scale Numeric Features*

---

## 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 / median (depending on robust) of each numeric feature during training, or 0 if center is FALSE. Will be subtracted during the predict phase.
- scale :: numeric
  The value by which features are divided. 1 if scale is FALSE
  If robust is FALSE, this is the root mean square, defined as sqrt(sum(x^2)/(length(x)-1)), of each feature, possibly after centering. If robust is TRUE, this is the mean absolute deviation multiplied by 1.4826 (see stats::mad of each feature, possibly after centering. 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.
- robust :: logical(1)
  Whether to use robust scaling; instead of scaling / centering with mean / standard deviation, median and median absolute deviation mad are used. Initialized to FALSE.

### Internals

Imitates the scale() function for robust = FALSE and alternatively subtracts the median and divides by mad for robust = TRUE.

### Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

### See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures,

mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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

*Scale Numeric Features with Respect to their Maximum Absolute Value*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`
`mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`
`mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscalerange`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

### Examples

```
library("mlr3")

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

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

pop$state
```

---

`mlr_pipeops_scalerange`

*Linearly Transform Numeric Features to Match Given Boundaries*

---

### Description

Linearly transforms numeric data columns so they are between `lower` and `upper`. The formula for this is $x' = offset + x * scale$, where $scale$ is $(upper - lower)/(max(x) - min(x))$ and $offset$ is $-min(x) * scale + lower$. The same transformation is applied during training and prediction.

### 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 $scale$ and $offset$ 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. Initialized to 0.
- `upper` :: `numeric(1)`
  Target value of greatest item of input data. Initialized to 1.

## Methods

Only methods inherited from PipeOpTaskPreprocSimple/PipeOpTaskPreproc/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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` *Remove Features Depending on a Selector*

## 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 `Selector`s.

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

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` *SMOTE Balancing*

## 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 purely 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()`.

## Fields

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

## Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

## References

Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002). "SMOTE: Synthetic Minority Oversampling Technique." *Journal of Artificial Intelligence Research*, **16**, 321–357. doi:10.1613/jair.953.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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` *Normalize Data Row-wise*

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

task = tsk("iris")

task$data()

pop = po("spatialsign")

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

---

# mlr_pipeops_subsample *Subsampling*

---

## 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 = "subsample", 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$filter()` to remove rows. If `replace` is `TRUE` and identical rows are added, then the `task$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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

pos = mlr_pipeops$get("subsample", param_vals = list(frac = 0.7, stratify = TRUE))

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

---

# mlr_pipeops_targetinvert

*Invert Target Transformations*

---

## Description

Inverts target-transformations done during training based on a supplied inversion function. Typically should be used in combination with a subclass of PipeOpTargetTrafo.

During prediction phase the function supplied through "fun" is called with a list containing the "prediction" as a single element, and should return a list with a single element (a Prediction) that is returned by PipeOpTargetInvert.

## Format

R6Class object inheriting from PipeOp.

### Construction

```
PipeOpTargetInvert$new(id = "targetinvert", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"targetinvert"`.
- `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

`PipeOpTargetInvert` has two input channels named `"fun"` and `"prediction"`. During training, both take `NULL` as input. During prediction, `"fun"` takes a function and `"prediction"` takes a `Prediction`.

`PipeOpTargetInvert` has one output channel named `"output"` and returns `NULL` during training and a `Prediction` during prediction.

### State

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

### Parameters

`PipeOpTargetInvert` has no parameters.

### Internals

Should be used in combination with a subclass of `PipeOpTargetTrafo`.

### Fields

Only fields inherited from `PipeOp`.

### Methods

Only methods inherited from `PipeOp`.

### See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`
`mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`,

mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

---

`mlr_pipeops_targetmutate` *Transform a Target by a Function*

---

## Description

Changes the *target* of a `Task` according to a function given as hyperparameter. An inverter-function that undoes the transformation during prediction must also be given.

## Format

`R6Class` object inheriting from `PipeOpTargetTrafo`/`PipeOp`

## Construction

```
PipeOpTargetMutate$new(id = "targetmutate", param_vals = list(), new_task_type = NULL)
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"targetmutate"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.
- `new_task_type` :: `character(1)` | `NULL`
  The task type to which the output is converted, must be one of `mlr_reflections$task_types$type`. Defaults to `NULL`: no change in task type.

## Input and Output Channels

Input and output channels are inherited from `PipeOpTargetTrafo`.

## State

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

## Parameters

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

- trafo :: function data.table -> data.table
  Transformation function for the target. Should only be a function of the target, i.e., taking a single data.table argument, typically with one column. The return value is used as the new target of the resulting Task. To change target names, change the column name of the data using e.g. setnames().
  Note that this function also gets called during prediction and should thus gracefully handle NA values.
  Initialized to identity().
- inverter :: function data.table -> data.table | named list
  Inversion of the transformation function for the target. Called on a data.table created from a Prediction using as.data.table(), without the $row_ids and $truth columns, and should return a data.table or named list that contains the new relevant slots of a Prediction subclass (e.g., $response, $prob, $se, ...). Initialized to identity().

## Internals

Overloads PipeOpTargetTrafo's .transform() and .invert() functions. Should be used in combination with PipeOpTargetInvert.

## Fields

Fields inherited from PipeOp, as well as:

- new_task_type :: character(1)
  new_task_type construction argument. Read-only.

## Methods

Only methods inherited from PipeOpTargetTrafo/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs,

mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targettrafoscalerange, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library(mlr3)
task = tsk("boston_housing")
po = PipeOpTargetMutate$new("logtrafo", param_vals = list(
  trafo = function(x) log(x, base = 2),
  inverter = function(x) list(response = 2 ^ x$response))
)
# Note that this example is ill-equipped to work with
# `predict_type == "se"` predictions.

po$train(list(task))
po$predict(list(task))

g = Graph$new()
g$add_pipeop(po)
g$add_pipeop(LearnerRegrRpart$new())
g$add_pipeop(PipeOpTargetInvert$new())
g$add_edge(src_id = "logtrafo", dst_id = "targetinvert",
  src_channel = 1, dst_channel = 1)
g$add_edge(src_id = "logtrafo", dst_id = "regr.rpart",
  src_channel = 2, dst_channel = 1)
g$add_edge(src_id = "regr.rpart", dst_id = "targetinvert",
  src_channel = 1, dst_channel = 2)

g$train(task)
g$predict(task)

#syntactic sugar using ppl():
tt = ppl("targettrafo", graph = PipeOpLearner$new(LearnerRegrRpart$new()))
tt$param_set$values$targetmutate.trafo = function(x) log(x, base = 2)
tt$param_set$values$targetmutate.inverter = function(x) list(response = 2 ^ x$response)
```

---

# mlr_pipeops_targettrafoscalerange

*Linearly Transform a Numeric Target to Match Given Boundaries*

---

## Description

Linearly transforms a numeric target of a TaskRegr so it is between lower and upper. The formula for this is $x' = offset + x * scale$, where $scale$ is $(upper - lower)/(max(x) - min(x))$ and $offset$ is $-min(x) * scale + lower$. The same transformation is applied during training and prediction.

### Format

R6Class object inheriting from PipeOpTargetTrafo/PipeOp

### Construction

```
PipeOpTargetTrafoScaleRange$new(id = "targettrafoscalerange", param_vals = list())
```

- id :: character(1)
  Identifier of resulting object, default "targettrafoscalerange".
- 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 PipeOpTargetTrafo.

### State

The $state is a named list containing the slots $offset and $scale.

### Parameters

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

- lower :: numeric(1)
  Target value of smallest item of input target. Initialized to 0.
- upper :: numeric(1)
  Target value of greatest item of input target. Initialized to 1.

### Internals

Overloads PipeOpTargetTrafo's .get_state(), .transform(), and .invert(). Should be used in combination with PipeOpTargetInvert.

### Methods

Only methods inherited from PipeOpTargetTrafo/PipeOp.

### See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian,

mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library(mlr3)
task = tsk("boston_housing")
po = PipeOpTargetTrafoScaleRange$new()

po$train(list(task))
po$predict(list(task))

#syntactic sugar for a graph using ppl():
ttscalerange = ppl("targettrafo", trafo_pipeop = PipeOpTargetTrafoScaleRange$new(),
  graph = PipeOpLearner$new(LearnerRegrRpart$new()))
ttscalerange$train(task)
ttscalerange$predict(task)
ttscalerange$state$regr.rpart
```

---

# mlr_pipeops_textvectorizer

*Bag-of-word Representation of Character Features*

---

## Description

Computes a bag-of-word representation from a (set of) columns. Columns of type `character` are split up into words. Uses the `quanteda::dfm()`, `quanteda::dfm_trim()` from the 'quanteda' package. TF-IDF computation works similarly to `quanteda::dfm_tfidf()` but has been adjusted for train/test data split using `quanteda::docfreq()` and `quanteda::dfm_weight()`

In short:

- Per default, produces a bag-of-words representation
- If `n` is set to values > 1, ngrams are computed
- If `df_trim` parameters are set, the bag-of-words is trimmed.
- The `scheme_tf` parameter controls term-frequency (per-document, i.e. per-row) weighting
- The `scheme_df` parameter controls the document-frequency (per token, i.e. per-column) weighting.

Parameters specify arguments to quanteda's dfm, dfm_trim, docfreq and dfm_weight. What belongs to what can be obtained from each params tags where tokenizer are arguments passed on to quanteda::dfm(). Defaults to a bag-of-words representation with token counts as matrix entries.

In order to perform the *default* dfm_tfidf weighting, set the scheme_df parameter to "inverse". The scheme_df parameter is initialized to "unary", which disables document frequency weighting.

The pipeop works as follows:

1. Words are tokenized using quanteda::tokens.
2. Ngrams are computed using quanteda::tokens_ngrams
3. A document-frequency matrix is computed using quanteda::dfm
4. The document-frequency matrix is trimmed using quanteda::dfm_trim during train-time.
5. The document-frequency matrix is re-weighted (similar to quanteda::dfm_tfidf) if scheme_df is not set to "unary".

## Format

R6Class object inheriting from PipeOpTaskPreproc/PipeOp.

## Construction

```
PipeOpTextVectorizer$new(id = "textvectorizer", param_vals = list())
```

- id :: character(1)
  Identifier of resulting object, default "textvectorizer".
- 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 features converted to a bag-of-words representation.

## State

The $state is a list with element 'cols': A vector of extracted columns.

## Parameters

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

- return_type :: character(1)
  Whether to return an integer representation ("integer-sequence") or a Bag-of-words ("bow"). If set to "integer_sequence", tokens are replaced by an integer and padded/truncated to sequence_length. If set to "factor_sequence", tokens are replaced by a factor and padded/truncated to sequence_length. If set to 'bow', a possibly weighted bag-of-words matrix is returned. Defaults to bow.

- stopwords_language :: character(1)
  Language to use for stopword filtering. Needs to be either "none", a language identifier listed in stopwords::stopwords_getlanguages("snowball") ("de", "en", ...) or "smart". "none" disables language-specific stopwords. "smart" coresponds to stopwords::stopwords(source = "smart"), which contains *English* stopwords and also removes one-character strings. Initialized to "smart".

- extra_stopwords :: character
  Extra stopwords to remove. Must be a character vector containing individual tokens to remove. Initialized to character(0). When n is set to values greater than 1, this can also contain stop-ngrams.
- tolower :: logical(1)
  Convert to lower case? See quanteda::dfm. Default: TRUE.
- stem :: logical(1)
  Perform stemming? See quanteda::dfm. Default: FALSE.
- what :: character(1)
  Tokenization splitter. See quanteda::tokens. Default: word.
- remove_punct :: logical(1)
  See quanteda::tokens. Default: FALSE.
- remove_url :: logical(1)
  See quanteda::tokens. Default: FALSE.
- remove_symbols :: logical(1)
  See quanteda::tokens. Default: FALSE.
- remove_numbers :: logical(1)
  See quanteda::tokens. Default: FALSE.
- remove_separators :: logical(1)
  See quanteda::tokens. Default: TRUE.
- split_hypens :: logical(1)
  See quanteda::tokens. Default: FALSE.
- n :: integer
  Vector of ngram lengths. See quanteda::tokens_ngrams. Initialized to 1, deviating from the base function's default. Note that this can be a *vector* of multiple values, to construct ngrams of multiple orders.
- skip :: integer
  Vector of skips. See quanteda::tokens_ngrams. Default: 0. Note that this can be a *vector* of multiple values.
- sparsity :: numeric(1)
  Desired sparsity of the 'tfm' matrix. See quanteda::dfm_trim. Default: NULL.
- max_termfreq :: numeric(1)
  Maximum term frequency in the 'tfm' matrix. See quanteda::dfm_trim. Default: NULL.
- min_termfreq :: numeric(1)
  Minimum term frequency in the 'tfm' matrix. See quanteda::dfm_trim. Default: NULL.
- termfreq_type :: character(1)
  How to asess term frequency. See quanteda::dfm_trim. Default: "count".

- scheme_df :: character(1)
  Weighting scheme for document frequency: See quanteda::docfreq. Initialized to "unary" (1 for each document, deviating from base function default).
- smoothing_df :: numeric(1)
  See quanteda::docfreq. Default: 0.
- k_df :: numeric(1)
  k parameter given to quanteda::docfreq (see there). Default is 0.
- threshold_df :: numeric(1)
  See quanteda::docfreq. Default: 0. Only considered for scheme_df = "count".
- base_df :: numeric(1)
  The base for logarithms in quanteda::docfreq (see there). Default: 10.
- scheme_tf :: character(1)
  Weighting scheme for term frequency: See quanteda::dfm_weight. Default: "count".
- k_tf :: numeric(1)
  k parameter given to quanteda::dfm_weight (see there). Default behaviour is 0.5.
- base_df :: numeric(1)
  The base for logarithms in quanteda::dfm_weight (see there). Default: 10.

#' * sequence_length :: integer(1)
The length of the integer sequence. Defaults to Inf, i.e. all texts are padded to the length of the longest text. Only relevant for "return_type" : "integer_sequence"

## Internals

See Description. Internally uses the quanteda package. Calls quanteda::tokens, quanteda::tokens_ngrams and quanteda::dfm. During training, quanteda::dfm_trim is also called. Tokens not seen during training are dropped during prediction.

## Methods

Only methods inherited from PipeOpTaskPreproc/PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta

mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")
library("data.table")
# create some text data
dt = data.table(
  txt = replicate(150, paste0(sample(letters, 3), collapse = " "))
)
task = tsk("iris")$cbind(dt)

pos = po("textvectorizer", param_vals = list(stopwords_language = "en"))

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_threshold *Change the Threshold of a Classification Prediction*

---

## Description

Change the threshold of a `Prediction` during the `predict` step. The incoming `Learner`'s `$predict_type` needs to be `"prob"`. Internally calls `PredictionClassif$set_threshold`.

## Format

`R6Class` inheriting from `PipeOp`.

## Construction

```
PipeOpThreshold$new(id = "threshold", param_vals = list())
```

- `id` :: `character(1)` Identifier of the resulting object, default `"threshold"`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Defaults to `numeric(0)`.

### Input and Output Channels

During training, the input and output are `NULL`. A `PredictionClassif` is required as input and returned as output during prediction.

### State

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

### Parameters

- `thresholds` :: `numeric`
  A numeric vector of thresholds for the different class levels. May have length 1 for binary classification predictions, must otherwise have length of the number of target classes; see `PredictionClassif`'s `$set_threshold()` method. Initialized to `0.5`, i.e. thresholding for binary classification at level `0.5`.

### Fields

Only fields inherited from `PipeOp`.

### Methods

Only methods inherited from `PipeOp`.

### See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`
`mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`
`mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscale`
`mlr_pipeops_textvectorizer`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

### Examples

```
library("mlr3")
t = tsk("german_credit")
```

```
gr = po(lrn("classif.rpart", predict_type = "prob")) %>>%
  po("threshold", param_vals = list(thresholds = 0.9))
gr$train(t)
gr$predict(t)
```

---

`mlr_pipeops_tunethreshold`

*Tune the Threshold of a Classification Prediction*

---

## Description

Tunes optimal probability thresholds over different `PredictionClassif`s.

`mlr3::Learner` `predict_type`: `"prob"` is required. Thresholds for each learner are optimized using the `Optimizer` supplied via the `param_set`. Defaults to `GenSA`. Returns a single `PredictionClassif`.

This PipeOp should be used in conjunction with `PipeOpLearnerCV` in order to optimize thresholds of cross-validated predictions. In order to optimize thresholds without cross-validation, use `PipeOpLearnerCV` in conjunction with `ResamplingInsample`.

## Format

`R6Class` object inheriting from `PipeOp`.

## Construction

```
* `PipeOpTuneThreshold$new(id = "tunethreshold", param_vals = list())` \cr
  (`character(1)`, `list`) -> `self` \cr
```

- `id` :: `character(1)`
  Identifier of resulting object. Default: "tunethreshold".
- `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 `PipeOp`.

## State

The `$state` is a named `list` with elements

- `thresholds` :: `numeric` learned thresholds

## Parameters

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

- measure :: Measure | character
  Measure to optimize for. Will be converted to a Measure in case it is character. Initialized to "classif.ce", i.e. misclassification error.
- optimizer :: Optimizer|character(1)
  Optimizer used to find optimal thresholds. If character, converts to Optimizer via opt. Initialized to OptimizerGenSA.
- log_level :: character(1) | integer(1)
  Set a temporary log-level for lgr::get_logger("bbotk"). Initialized to: "warn".

## Internals

Uses the optimizer provided as a param_val in order to find an optimal threshold. See the optimizer parameter for more info.

## Methods

Only methods inherited from PipeOp.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale
mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

task = tsk("iris")
```

```
pop = po("learner_cv", lrn("classif.rpart", predict_type = "prob")) %>>%
  po("tunethreshold")

task$data()
pop$train(task)

pop$state
```

---

`mlr_pipeops_unbranch` *Unbranch Different Paths*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

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_updatetarget` *Transform a Target without an Explicit Inversion*

## Description

EXPERIMENTAL, API SUBJECT TO CHANGE

Handles target transformation operations that do not need explicit inversion. In case the new target is required during predict, creates a vector of `NA`. Works similar to `PipeOpTargetTrafo` and `PipeOpTargetMutate`, but forgoes the inversion step. In case target after the `trafo` is a factor, levels are saved to `$state`.

During prediction: Sets all target values to `NA` before calling the `trafo` again. In case target after the `trafo` is a factor, levels saved in the `state` are set during prediction.

As a special case when `trafo` is `identity` and `new_target_name` matches an existing column name of the data of the input `Task`, this column is set as the new target. Depending on `drop_original_target` the original target is then either dropped or added to the features.

## Format

Abstract `R6Class` inheriting from `PipeOp`.

## Construction

```
PipeOpUpdateTarget$new(id, param_set = ps(),
  param_vals = list(), packages = character(0))
```

- `id` :: `character(1)`
  Identifier of resulting object. See `$id` slot of `PipeOp`.
- `param_vals` :: named `list`
  List of hyperparameter settings, overwriting the hyperparameter settings given in `param_set`. The subclass should have its own `param_vals` parameter and pass it on to `super$initialize()`. Default `list()`.

## Parameters

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

- `trafo` :: `function`
  Transformation function for the target. Should only be a function of the target, i.e., taking a single argument. Default is `identity`. Note, that the data passed on to the target is a `data.table` consisting of all target column.
- `new_target_name` :: `character(1)`
  Optionally give the transformed target a new name. By default the original name is used.

- `new_task_type` :: `character(1)`
  Optionally a new task type can be set. Legal types are listed in `mlr_reflections$task_types$type`.
  #' `drop_original_target` :: `logical(1)`
  Whether to drop the original target column. Default: `TRUE`.

## State

The `$state` is a list of class levels for each target after trafo. `list()` if none of the targets have levels.

## Methods

Only methods inherited from `PipeOp`.

## See Also

https://mlr-org.com/pipeops.html

Other mlr3pipelines backend related: `Graph`, `PipeOp`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_graphs`, `mlr_pipeops`

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`
`mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`
`mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscale`
`mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

## Examples

```
## Not run:
# Create a binary class task from iris
library(mlr3)
trafo_fun = function(x) {factor(ifelse(x$Species == "setosa", "setosa", "other"))}
po = PipeOpUpdateTarget$new(param_vals = list(trafo = trafo_fun, new_target_name = "setosa"))
po$train(list(tsk("iris")))
po$predict(list(tsk("iris")))

## End(Not run)
```

`mlr_pipeops_vtreat` *Interface to the vtreat Package*

## Description

Provides an interface to the vtreat package.

`PipeOpVtreat` naturally works for classification tasks and regression tasks. Internally, `PipeOpVtreat` follows the fit/prepare interface of vtreat, i.e., first creating a data treatment transform object via `vtreat::NumericOutcomeTreatment()`, `vtreat::BinomialOutcomeTreatment()`, or `vtreat::MultinomialOutcomeTr` followed by calling `vtreat::fit_prepare()` on the training data and `vtreat::prepare()` during predicton.

## Format

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

## Construction

```
PipeOpVreat$new(id = "vtreat", param_vals = list())
```

- `id` :: `character(1)`
  Identifier of resulting object, default `"vtreat"`.
- `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 features "prepared" by vtreat. If vtreat found "no usable vars", the input `Task` is returned unaltered.

## State

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

- `treatment_plan` :: object of class `vtreat_pipe_step` | `NULL`
  The treatment plan as constructed by vtreat based on the training data, i.e., an object of class `treatment_plan`. If vtreat found "no usable vars" and designing the treatment would have failed, this is `NULL`.

### Parameters

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

- recommended :: logical(1)
  Whether only the "recommended" prepared features should be returned, i.e., non constant variables with a significance value smaller than vtreat's threshold. Initialized to TRUE.
- cols_to_copy :: function | Selector
  Selector function, takes a Task as argument and returns a character() of features to copy. See Selector for example functions. Initialized to selector_none().
- minFraction :: numeric(1)
  Minimum frequency a categorical level must have to be converted to an indicator column.
- smFactor :: numeric(1)
  Smoothing factor for impact coding models.
- rareCount :: integer(1)
  Allow levels with this count or below to be pooled into a shared rare-level.
- rareSig :: numeric(1)
  Suppress levels from pooling at this significance value greater.
- collarProb :: numeric(1)
  What fraction of the data (pseudo-probability) to collar data at if doCollar = TRUE.
- doCollar :: logical(1)
  If TRUE collar numeric variables by cutting off after a tail-probability specified by collarProb during treatment design.
- codeRestriction :: character()
  What types of variables to produce.
- customCoders :: named list
  Map from code names to custom categorical variable encoding functions.
- splitFunction :: function
  Function taking arguments nSplits, nRows, dframe, and y; returning a user desired split.
- ncross :: integer(1)
  Integer larger than one, number of cross-validation rounds to design.
- forceSplit :: logical(1)
  If TRUE force cross-validated significance calculations on all variables.
- catScaling :: logical(1)
  If TRUE use stats::glm() linkspace, if FALSE use stats::lm() for scaling.
- verbose :: logical(1)
  If TRUE print progress.
- use_paralell :: logical(1)
  If TRUE use parallel methods.
- missingness_imputation :: function
  Function of signature f(values: numeric, weights: numeric), simple missing value imputer. Typically, an imputation via a PipeOp should be preferred, see PipeOpImpute.
- pruneSig :: numeric(1)
  Suppress variables with significance above this level. Only effects [regression tasksmlr3::TaskRegr and binary classification tasks.

- `scale` :: `logical(1)`
  If `TRUE` replace numeric variables with single variable model regressions ("move to outcome-scale"). These have mean zero and (for variables with significant less than 1) slope 1 when regressed (lm for regression problems/glm for classification problems) against outcome.
- `varRestriction` :: `list()`
  List of treated variable names to restrict to. Only effects [regression tasksmlr3::TaskRegr and binary classification tasks.
- `trackedValues` :: named `list()`
  Named list mapping variables to know values, allows warnings upon novel level appearances (see `vtreat::track_values()`). Only effects [regression tasksmlr3::TaskRegr and binary classification tasks.
- `y_dependent_treatments` :: `character()`
  Character what treatment types to build per-outcome level. Only effects multiclass classification tasks.
- `imputation_map` :: named `list`
  List of map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.
  Typically, an imputation via a `PipeOp` is to be preferred, see `PipeOpImpute`.

For more information, see `vtreat::regression_parameters()`, `vtreat::classification_parameters()`, or `vtreat::multinomial_parameters()`.

## Internals

Follows vtreat's fit/prepare interface. See `vtreat::NumericOutcomeTreatment()`, `vtreat::BinomialOutcomeTreatment`
`vtreat::MultinomialOutcomeTreatment()`, `vtreat::fit_prepare()` and `vtreat::prepare()`.

## Methods

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

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputecons`
`mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`
`mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`,

mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale
mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch,
mlr_pipeops_updatetarget, mlr_pipeops_yeojohnson

## Examples

```
library("mlr3")

set.seed(2020)

make_data <- function(nrows) {
    d <- data.frame(x = 5 * rnorm(nrows))
    d["y"] = sin(d[["x"]]) + 0.01 * d[["x"]] + 0.1 * rnorm(nrows)
    d[4:10, "x"] = NA  # introduce NAs
    d["xc"] = paste0("level_", 5 * round(d$y / 5, 1))
    d["x2"] = rnorm(nrows)
    d[d["xc"] == "level_-1", "xc"] = NA  # introduce a NA level
    return(d)
}

task = TaskRegr$new("vtreat_regr", backend = make_data(100), target = "y")

pop = PipeOpVtreat$new()
pop$train(list(task))
```

---

# mlr_pipeops_yeojohnson

*Yeo-Johnson Transformation of Numeric Features*

---

## 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons
mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs,

mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat

## Examples

```
library("mlr3")

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

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

pop$state
```

---

`Multiplicity` *Multiplicity*

---

## Description

A `Multiplicity` class S3 object.

The function of multiplicities is to indicate that `PipeOp`s should be executed multiple times with multiple values.

A `Multiplicity` is a container, like a `list()`, that contains multiple values. If the message that is passed along the edge of a `Graph` is a `Multiplicity`-object, then the `PipeOp` that receives this object will *usually* be called once for each contained value. The result of each of these calls is then, again, packed in a `Multiplicity` and sent along the outgoing edge(s) of that `PipeOp`. This means that a `Multiplicity` can cause multiple `PipeOp`s in a row to be run multiple times, where the run for each element of the `Multiplicity` is independent from the others.

Most `PipeOp`s only return a `Multiplicity` if their input was a `Multiplicity` (and after having run their code multiple times, once for each entry). However, there are a few special `PipeOp`s that are "aware" of `Multiplicity` objects. These may either *create* a `Multiplicity` even though not having a `Multiplicity` input (e.g. `PipeOpReplicate` or `PipeOpOVRSplit`) – causing the subsequent `PipeOp`s to be run multiple times – or *collect* a `Multiplicity`, being called only once even though their input is a `Multiplicity` (e.g. `PipeOpOVRUnite` or `PipeOpFeatureUnion` if constructed with the `collect_multiplicity` argument set to `TRUE`). The combination of these mechanisms makes it possible for parts of a `Graph` to be called variably many times if "sandwiched" between `Multiplicity` creating and collecting `PipeOp`s.

Whether a `PipeOp` creates or collects a `Multiplicity` is indicated by the `$input` or `$output` slot (which indicate names and types of in/out channels). If the `train` and `predict` types of an input or output are surrounded by square brackets (`"["`, `"]"`), then this channel handles a `Multiplicity` explicitly. Depending on the function of the `PipeOp`, it will usually collect (input channel) or create (output channel) a `Multiplicity`. `PipeOp`s without this indicator are `Multiplicity` agnostic and blindly execute their function multiple times when given a `Multiplicity`.

If a PipeOp is trained on a Multiplicity, the $state slot is set to a Multiplicity as well; this Multiplicity contains the "original" $state resulting from each individual call of the PipeOP with the input Multiplicity's content. If a PipeOp was trained with a Multiplicity, then the predict() argument must be a Multiplicity with the same number of elements.

## Usage

```
Multiplicity(...)
```

## Arguments

| | |
|---|---|
| ... | any<br>Can be anything. |

## Value

Multiplicity

## See Also

Other Special Graph Messages: NO_OP

Other Experimental Features: mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_replicate

Other Multiplicity PipeOps: PipeOpEnsemble, mlr_pipeops_classifavg, mlr_pipeops_featureunion, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_regravg, mlr_pipeops_replicate

---

# 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

Other Special Graph Messages: Multiplicity()

PipeOp *PipeOp Base Class*

## 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 `$input` and then passed on to the `private$.train()` or `private$.predict()` functions. There the data is processed and a result `list` is created. This `list` is again type-checked for declared output types of each channel. The length and types of the result `list` is as declared in `$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 = ps(), param_vals = list(), input, output, packages = character(0), tags = cha
```

- `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)`.
- `tags` ::`character`
  A set of tags associated with the `PipeOp`. Tags describe a PipeOp's purpose. Can be used to filter `as.data.table(mlr_pipeops)`. Default is `"abstract"`, indicating an abstract `PipeOp`.

## Internals

`PipeOp` is an abstract class with abstract functions `private$.train()` and `private$.predict()`. 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 `$output`. The `private$.train()` and `private$.predict()` function should not be called by the user; instead, a `$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.

`private$.train()` and `private$.predict()` should treat their inputs as read-only. If they are `R6` objects, they should be cloned before being manipulated in-place. Objects, or parts of objects, that are not changed, do not need to be cloned, and it is legal to return the same identical-by-reference objects to multiple outputs.

## 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 `$set_names()` method should be used.
- `packages` :: `character`
  Packages required for the `PipeOp`. Functions that are not in base R should still be called using `::` (or explicitly attached using `require()`) in `private$.train()` *and* `private$.predict()`, but packages declared here are checked before any (possibly expensive) processing has started within a `Graph`.
- `param_set` :: `ParamSet`
  Parameters and parameter constraints. Parameter values that influence the functioning of `$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 $train(), because private$.train() may theoretically be executed in a different R-session (e.g. for parallelization). $state should furthermore always be set to something with copy-semantics, since it is never cloned. This is a limitation not of PipeOp or mlr3pipelines, but of the way the system as a whole works, together with GraphLearner and mlr3.
- 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 $predict(). Column train is the (S3) class that an input object must conform to during training, column predict is the (S3) class that an input object must conform to during prediction. Types are checked by the PipeOp itself and do not need to be checked by private$.train() / private$.predict() code.
  A special name is "...", which creates a *vararg* input channel that accepts a variable number of inputs.
  If a row has both train and predict values enclosed by square brackets ("[", "]"), then this channel is Multiplicity-aware. If the PipeOp receives a Multiplicity value on these channels, this Multiplicity is given to the .train() and .predict() functions directly. Otherwise, the Multiplicity is transparently unpacked and the .train() and .predict() functions are called multiple times, once for each Multiplicity element. The type enclosed by square brackets indicates that only a Multiplicity containing values of this type are accepted. See Multiplicity for more information.
- 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 $predict functions. Column train is the (S3) class that an output object must conform to during training, column predict is the (S3) class that an output object must conform to during prediction. The PipeOp checks values returned by private$.train() and private$.predict() against these types specifications.
  If a row has both train and predict values enclosed by square brackets ("[", "]"), then this signals that the channel emits a Multiplicity of the indicated type. See Multiplicity for more information.
- innum :: numeric(1)
  Number of input channels. This equals nrow($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.
- tags ::character
  A set of tags associated with the PipeOp. Tags describe a PipeOp's purpose. Can be used to filter as.data.table(mlr_pipeops). PipeOp tags are inherited and child classes can introduce additional tags.
- hash :: character(1)
  Checksum calculated on the PipeOp, depending on the PipeOp's class and the slots $id and $param_set$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

- phash :: character(1)
  Checksum calculated on the PipeOp, depending on the PipeOp's class and the slots $id but ignoring $param_set$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 $predict() are saved to this slot, exactly as they are returned by these functions. This is mainly for debugging purposes and done, if requested, by the Graph backend itself; it should *not* be done explicitly by private$.train() or private$.predict().
- man :: character(1)
  Identifying string of the help page that shows with help().
- properties :: character()
  The properties of the pipeop. Currently supported values are:
  - "validation": the PipeOp can make use of the $internal_valid_task of an mlr3::Task. This is for example used for PipeOpLearners that wrap a Learner with this property, see mlr3::Learner. PipeOps that have this property, also have a $validate field, which controls whether to use the validation task, as well as a $internal_valid_scores field, which allows to access the internal validation scores after training.
  - "internal_tuning": the PipeOp is able to internally optimize hyperparameters. This works analogously to the internal tuning implementation for mlr3::Learner. PipeOps with that property also implement the standardized accessor $internal_tuned_values and have at least one parameter tagged with "internal_tuning". An example for such a PipeOp is a PipeOpLearner that wraps a Learner with the "internal_tuning" property.

  Programatic access to all available properties is possible via mlr_reflections$pipeops$properties.

## 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 $output train column. The workhorse function for training each PipeOp is the private .train(input)
  : (named list) -> list
  function. It's an Abstract function that must be implemented by concrete subclasses. private$.train() 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 $output train column. Names of the returned list are ignored.
  The private$.train() 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. The workhorse function for predicting in each using each PipeOp is
.predict(input)
(named list) -> list
Abstract function that must be implemented by concrete subclasses. private$.predict() is called by $predict() after typechecking and works analogously to private$.train(). Unlike private$.train(), private$.predict() should not modify the PipeOp in any way. Just as private$.train(), private$.predict() 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.
- help(help_type)
(character(1)) -> help file
Displays the help file of the concrete PipeOp instance. help_type is one of "text", "html", "pdf" and behaves as the help_type argument of R's help().

## Inheriting

To create your own PipeOp, you need to overload the private$.train() and private$.test() functions. It is most likely also necessary to overload the $initialize() function to do additional initialization. The $initialize() method should have at least the arguments id and param_vals, which should be passed on to super$initialize() unchanged. id should have a useful default value, and param_vals should have the default value list(), meaning no initialization of hyperparameters.

If the $initialize() method has more arguments, then it is necessary to also overload the private$.additional_phash_i function. This function should return either all objects, or a hash of all objects, that can change the function or behavior of the PipeOp and are independent of the class, the id, the $state, and the $param_set$values. The last point is particularly important: changing the $param_set$values should *not* change the return value of private$.additional_phash_input().

## See Also

https://mlr-org.com/pipeops.html

Other mlr3pipelines backend related: Graph, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_graphs, mlr_pipeops, mlr_pipeops_updatetarget

Other PipeOps: PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputecons mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin,

mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta
mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs,
mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign,
mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale
mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch,
mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

## 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$initialize(id, param_vals = param_vals,
        # declare "input" and "output" during construction here
        # training takes two 'numeric' and returns a 'numeric';
        # prediction takes 'NULL' and returns a 'character'.
        input = data.table::data.table(name = c("input1", "input2"),
          train = "numeric", predict = "NULL"),
        output = data.table::data.table(name = "output",
          train = "numeric", predict = "character")
      )
    }
  ),
  private = list(
    # PipeOp deriving classes must implement .train and
    # .predict; each taking an input list and returning
    # a list as output.
    .train = function(input) {
      sum = input[[1]] + input[[2]]
      self$state = sum
      list(sum)
    },
    .predict = 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 *Ensembling Base Class*

## Description

Parent class for PipeOps that aggregate predictions. Implements the private$.train() and private$.predict() methods necessary for a PipeOp and requires deriving classes to create the private$weighted_avg_predictions() 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, collect_multiplicity = FALSE, id, param_set = ps(), param_vals = list(), p
```

- 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.
- collect_multiplicity :: logical(1)
  If TRUE, the input is a Multiplicity collecting channel. This means, a Multiplicity input, instead of multiple normal inputs, is accepted and the members are aggregated. This requires innum to be 0. Default is FALSE.
- 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 $output type specifications. Should be "Prediction" (default) or one of its subclasses, e.g. "PredictionClassif", and correspond to the type accepted by private$.train() and private$.predict().

### 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 private$.predict() 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

https://mlr-org.com/pipeops.html

Other PipeOps: PipeOp, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

Other Multiplicity PipeOps: Multiplicity(), mlr_pipeops_classifavg, mlr_pipeops_featureunion, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_regravg, mlr_pipeops_replicate

Other Ensembles: mlr_learners_avg, mlr_pipeops_classifavg, mlr_pipeops_ovrunite, mlr_pipeops_regravg

---

# PipeOpImpute *Imputation Base Class*

---

## Description

Abstract base class for feature imputation.

## Format

Abstract R6Class object inheriting from PipeOp.

## Construction

```
PipeOpImpute$$new(id, param_set = ps(), param_vals = list(), whole_task_dependent = FALSE, packages = c
```

- 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$initialize(). Default list().
- whole_task_dependent :: logical(1)
  Whether the context_columns parameter should be added which lets the user limit the columns that are used for imputation inference. This should generally be FALSE if imputation depends only on individual features (e.g. mode imputation), and TRUE if imputation depends on other features as well (e.g. kNN-imputation).
- packages :: character
  Set of all required packages for the PipeOp's private$.train and private$.predict methods. See $packages slot. Default is character(0).
- task_type :: character(1)
  The class of Task that should be accepted as input and will be returned as output. This should generally be a character(1) identifying a type of Task, e.g. "Task", "TaskClassif" or "TaskRegr" (or another subclass introduced by other packages). Default is "Task".
- feature_types :: character
  Feature types affected by the PipeOp. See private$.select_cols() for more information.

### 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 private$.impute() function.

### State

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

- affected_cols :: character
  Names of features being selected by the affect_columns parameter.
- context_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 $feature_types slot. This is used during prediction to ensure that the Task resulting from the prediction operation has the same features, feature layout, and feature types as after training.
- model :: named list
  Model used for imputation. This is a list named by Task features, containing the result of the private$.train_imputer() or private$.train_nullmodel() 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. private$.select_cols() is for the *inheriting class* to determine which columns the operator should function on, e.g. based on feature type, while affect_columns is a way for the *user* to limit the columns that a PipeOpTaskPreproc should operate on. This method can optionally be overloaded when inheriting PipeOpImpute; If this method is not overloaded, it defaults to selecting the columns of type indicated by the feature_types construction argument.
- .train_imputer(feature, type, context)
  (atomic, character(1), data.table) -> any
  Abstract function that must be overloaded when inheriting. Called once for each feature selected by affect_columns to create the model entry to be used for private$.impute(). This function is only called for features with at least one non-missing value.
- .train_nullmodel(feature, type, context)
  (atomic, character(1), data.table) -> any
  Like .train_imputer(), but only called for each feature that only contains missing values. This is not an abstract function and, if not overloaded, gives a default response of 0 (integer, numeric), c(TRUE, FALSE) (logical), all available levels (factor/ordered), or the empty string (character).
- .impute(feature, type, model, context)
  (atomic, character(1), any, data.table) -> atomic
  Imputes the features. model is the model created by private$.train_imputer() Default behaviour is to assume model is an atomic vector from which values are sampled to impute

missing values of `feature`. `model` may have an attribute `probabilities` for non-uniform sampling.

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `PipeOpTaskPreprocSimple`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputeconstant`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscale`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

Other Imputation PipeOps: `mlr_pipeops_imputeconstant`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`

---

# PipeOpTargetTrafo *Target Transformation Base Class*

---

## Description

Base class for handling target transformation operations. Target transformations are different from feature transformation because they have to be "inverted" after prediction. The target is transformed during the training phase and information to invert this transformation is sent along to `PipeOpTargetInvert` which then inverts this transformation during the prediction phase. This inversion may need info about both the training and the prediction data.

Users can overload up to four `private$`-functions: `.get_state()` (optional), `.transform()` (mandatory), `.train_invert()` (optional), and `.invert()` (mandatory).

## Format

Abstract `R6Class` inheriting from `PipeOp`.

### Construction

```
PipeOpTargetTrafo$new(id, param_set = ps(), param_vals = list() packages = character(0), task_type_in =
```

- `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$initialize()`. Default `list()`.
- `task_type_in` :: `character(1)`
  The class of `Task` that should be accepted as input. This should generally be a `character(1)` identifying a type of `Task`, e.g. `"Task"`, `"TaskClassif"` or `"TaskRegr"` (or another subclass introduced by other packages). Default is `"Task"`.
- `task_type_out` :: `character(1)`
  The class of `Task` that is produced as output. This should generally be a `character(1)` identifying a type of `Task`, e.g. `"Task"`, `"TaskClassif"` or `"TaskRegr"` (or another subclass introduced by other packages). Default is the value of `task_type_in`.
- packages :: `character`
  Set of all required packages for the `PipeOp`'s methods. See `$packages` slot. Default is `character(0)`.
- tags :: `character` | `NULL`
  Tags of the resulting `PipeOp`. This is added to the tag `"target transform"`. Default `NULL`.

### Input and Output Channels

`PipeOpTargetTrafo` has one input channels named `"input"` taking a `Task` (or whatever class was specified by the `task_type` during construction) both during training and prediction.

`PipeOpTargetTrafo` has two output channels named `"fun"` and `"output"`. During training, `"fun"` returns `NULL` and during prediction, `"fun"` returns a function that can later be used to invert the transformation done during training according to the overloaded `.train_invert()` and `.invert()` functions. `"output"` returns the modified input `Task` (or `task_type`) according to the overloaded `transform()` function both during training and prediction.

### State

The `$state` is a named `list` and should be returned explicitly by the user in the overloaded `.get_state()` function.

### Internals

`PipeOpTargetTrafo` is an abstract class inheriting from `PipeOp`. It implements the `private$.train()` and `private$.predict()` functions. These functions perform checks and go on to call `.get_state()`, `.transform()`, `.train_invert()`. `.invert()` is packaged and sent along the `"fun"` output to be applied to a `Prediction` by `PipeOpTargetInvert`. A subclass of `PipeOpTargetTrafo` should implement these functions and be used in combination with `PipeOpTargetInvert`.

## Fields

Fields inherited from PipeOp.

## Methods

Methods inherited from PipeOp, as well as:

- `.get_state(task)`
  (Task) -> `list`
  Called by PipeOpTargetTrafo's implementation of `private$.train()`. Takes a single Task as input and returns a `list` to set the `$state`. `.get_state()` will be called a single time during *training* right before `.transform()` is called. The return value (i.e. the `$state`) should contain info needed in `.transform()` as well as in `.invert()`.
  The base implementation returns `list()` and should be overloaded if setting the state is desired.
- `.transform(task, phase)`
  (Task, `character(1)`) -> Task
  Called by PipeOpTargetTrafo's implementation of `private$.train()` and `private$.predict()`. Takes a single Task as input and modifies it. This should typically consist of calculating a new target and modifying the Task by using the convert_task function. `.transform()` will be called during training and prediction because the target (and if needed also type) of the input Task must be transformed both times. Note that unlike `$.train()`, the argument is *not* a list but a singular Task, and the return object is also *not* a list but a singular Task. The `phase` argument is `"train"` during training phase and `"predict"` during prediction phase and can be used to enable different behaviour during training and prediction. When `phase` is `"train"`, the `$state` slot (as previously set by `.get_state()`) may also be modified, alternatively or in addition to overloading `.get_state()`.
  The input should *not* be cloned and if possible should be changed in-place.
  This function is abstract and should be overloaded by inheriting classes.
- `.train_invert(task)`
  (Task) -> `any`
  Called by PipeOpTargetTrafo's implementation of `private$.predict()`. Takes a single Task as input and returns an arbitrary value that will be given as `predict_phase_state` to `.invert()`. This should not modify the input Task .
  The base implementation returns a list with a single element, the `$truth` column of the Task, and should be overloaded if a more training-phase-dependent state is desired.
- `.invert(prediction, predict_phase_state)`
  (Prediction, `any`) -> Prediction
  Takes a Prediction and a `predict_phase_state` object as input and inverts the prediction. This function is sent as `"fun"` to PipeOpTargetInvert.
  This function is abstract and should be overloaded by inheriting classes. Care should be taken that the `predict_type` of the Prediction being inverted is handled well.
- `.invert_help(predict_phase_state)`
  (`predict_phase_state` object) -> `function`
  Helper function that packages `.invert()` that can later be used for the inversion.

## See Also

https://mlr-org.com/pipeops.html

Other mlr3pipelines backend related: Graph, PipeOp, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_graphs, mlr_pipeops, mlr_pipeops_updatetarget

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTaskPreproc, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

---

# PipeOpTaskPreproc *Task Preprocessing Base Class*

---

## 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.

Prediction-behavior of preprocessing operations should always be independent for each row in the input-Task. This means that the prediction-operation of preprocessing-PipeOps should commute with `rbind()`: Running prediction on an `n`-row Task should result in the same result as `rbind()`-ing the prediction-result from `n` 1-row Tasks with the same content. In the large majority of cases, the number and order of rows should also not be changed during prediction.

Users must implement `private$.train_task()` and `private$.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 `private$.train_dt()` and `private$.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 `private$.train_task()`. The `private$.select_cols()` function can be overloaded for `private$.train_dt()` and `private$.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 private$.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$new(id, param_set = ps(), param_vals = list(), can_subset_cols = TRUE,
  packages = character(0), task_type = "Task", tags = NULL, feature_types = mlr_reflections$task_feature
```

- 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$initialize(). Default list().
- can_subset_cols :: logical(1)
  Whether the affect_columns parameter should be added which lets the user limit the columns that are modified by the PipeOpTaskPreproc. This should generally be FALSE if the operation adds or removes rows from the Task, and TRUE otherwise. Default is TRUE.
- packages :: character
  Set of all required packages for the PipeOp's private$.train() and private$.predict() methods. See $packages slot. Default is character(0).
- task_type :: character(1)
  The class of Task that should be accepted as input and will be returned as output. This should generally be a character(1) identifying a type of Task, e.g. "Task", "TaskClassif" or "TaskRegr" (or another subclass introduced by other packages). Default is "Task".
- tags :: character | NULL
  Tags of the resulting PipeOp. This is added to the tag "data transform". Default NULL.
- feature_types :: character
  Feature types affected by the PipeOp. See private$.select_cols() for more information. Defaults to all available feature types.

## 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 private$.train_task()/private$.predict_taks or private$.train_dt()/private$.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 $feature_types slot. This is used during prediction to ensure that the Task resulting from the prediction operation has the same features, feature layout, and feature types as after training.
- dt_columns :: character
  Names of features selected by the private$.select_cols() call during training. This is only present if the private$.train_dt() functionality is used, and not present if the private$.train_task() function is overloaded instead.
- feature_types :: character
  Feature types affected by the PipeOp. See private$.select_cols() for more information.

## 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 private$.train() and $.predict() functions. These functions perform checks and go on to call private$.train_task() and private$.predict_task(). A subclass of PipeOpTaskPreproc may implement these functions, or implement private$.train_dt() and private$.predict_dt() instead. This works by having the default implementations of private$.train_task() and private$.predict_task() call private$.train_dt() and private$.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 private$.train(). Takes a single Task as input and modifies it (ideally in-place without cloning) while storing information in the $state slot. Note that unlike $.train(), the argument is *not* a list but a singular Task, and the return object is also *not* a list but a singular Task. Also, contrary to private$.train(), 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 $state elements added by private$.train_task() and PipeOpTaskPreproc.
  By default this function calls the private$.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(). Takes a single Task as input and modifies it (ideally in-place without cloning) while using information in the $state slot. Works analogously to private$.train_task(). If private$.predict_task() should only be overloaded if private$.train_task() is overloaded (i.e. private$.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. If the input Task inherits from TaskSupervised, the target argument contains the $truth() information of the training Task; its type depends on the Task type being trained on.
  This method can be overloaded when inheriting from PipeOpTaskPreproc, together with private$.predict_dt() and optionally private$.select_cols(); alternatively, private$.train_task() and private$.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 $state. A transformed object must be returned that can be converted to a data.table using as.data.table. dt does not need to be copied deliberately, it is possible and encouraged to change it in-place.
  The levels argument is a named list of factor levels for factorial or character features.
  This method can be overloaded when inheriting PipeOpTaskPreproc, together with private$.train_dt() and optionally private$.select_cols(); alternatively, private$.train_task() and private$.predict_task() can be overloaded.
- .select_cols(task)
  (Task) -> character

Selects which columns the PipeOp operates on, if private$.train_dt() and private$.predict_dt() are overloaded. This function is not called if private$.train_task() and private$.predict_task() are overloaded. In contrast to the affect_columns parameter. private$.select_cols() is for the *inheriting class* to determine which columns the operator should function on, e.g. based on feature type, while affect_columns is a way for the *user* to limit the columns that a PipeOpTaskPreproc should operate on.
This method can optionally be overloaded when inheriting PipeOpTaskPreproc, together with private$.train_dt() and private$.predict_dt(); alternatively, private$.train_task() and private$.predict_task() can be overloaded.
If this method is not overloaded, it defaults to selecting of type indicated by the feature_types construction argument.

## See Also

https://mlr-org.com/pipeops.html

Other mlr3pipelines backend related: Graph, PipeOp, PipeOpTargetTrafo, PipeOpTaskPreprocSimple, mlr_graphs, mlr_pipeops, mlr_pipeops_updatetarget

Other PipeOps: PipeOp, PipeOpEnsemble, PipeOpImpute, PipeOpTargetTrafo, PipeOpTaskPreprocSimple, mlr_pipeops, 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_colroles, mlr_pipeops_copy, mlr_pipeops_datefeatures, mlr_pipeops_encode, mlr_pipeops_encodeimpact, mlr_pipeops_encodelmer, mlr_pipeops_featureunion, mlr_pipeops_filter, mlr_pipeops_fixfactors, mlr_pipeops_histbin, mlr_pipeops_ica, mlr_pipeops_imputeconstant, mlr_pipeops_imputehist, mlr_pipeops_imputelearner, mlr_pipeops_imputemean, mlr_pipeops_imputemedian, mlr_pipeops_imputemode, mlr_pipeops_imputeoor, mlr_pipeops_imputesample, mlr_pipeops_kernelpca, mlr_pipeops_learner, mlr_pipeops_missind, mlr_pipeops_modelmatrix, mlr_pipeops_multiplicityexply, mlr_pipeops_multiplicityimply, mlr_pipeops_mutate, mlr_pipeops_nmf, mlr_pipeops_nop, mlr_pipeops_ovrsplit, mlr_pipeops_ovrunite, mlr_pipeops_pca, mlr_pipeops_proxy, mlr_pipeops_quantilebin, mlr_pipeops_randomprojection, mlr_pipeops_randomresponse, mlr_pipeops_regravg, mlr_pipeops_removeconsta, mlr_pipeops_renamecolumns, mlr_pipeops_replicate, mlr_pipeops_scale, mlr_pipeops_scalemaxabs, mlr_pipeops_scalerange, mlr_pipeops_select, mlr_pipeops_smote, mlr_pipeops_spatialsign, mlr_pipeops_subsample, mlr_pipeops_targetinvert, mlr_pipeops_targetmutate, mlr_pipeops_targettrafoscale, mlr_pipeops_textvectorizer, mlr_pipeops_threshold, mlr_pipeops_tunethreshold, mlr_pipeops_unbranch, mlr_pipeops_updatetarget, mlr_pipeops_vtreat, mlr_pipeops_yeojohnson

---

# PipeOpTaskPreprocSimple

*Simple Task Preprocessing Base Class*

---

## Description

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

Alternatively, analogously to the PipeOpTaskPreproc approach of offering private$.train_dt()/private$.predict_dt( the private$.get_state_dt() and private$.transform_dt() functions may be implemented.

private$.get_state must not change its input value in-place and must return something that will be written into $state (which must not be NULL), private$.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 = ps(), param_vals = list(), can_subset_cols = TRUE, packages
```

(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$initialize(). Default list().
- can_subset_cols :: logical(1)
  Whether the affect_columns parameter should be added which lets the user limit the columns that are modified by the PipeOpTaskPreprocSimple. This should generally be FALSE if the operation adds or removes rows from the Task, and TRUE otherwise. Default is TRUE.
- packages :: character
  Set of all required packages for the PipeOp's private$.train() and private$.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 private$.transform() or private$.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 private$.train_task() and private$.predict_task() functions. A subclass of PipeOpTaskPreprocSimple may implement the functions private$.get_state() and private$.transform(), or alternatively the functions private$.get_state_dt() and private$.transform_dt() (as well as private$.select_cols(), in the latter case). This works by having the default implementations of private$.get_state() and private$.transform() call private$.get_state_dt() and private$.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 $state during training phase of PipeOpTaskPreprocSimple. The state can then influence the private$.transform() function. Note that private$.get_state() must *return* the state, and should not store it in $state. It is not strictly necessary to implement either private$.get_state() or private$.get_state_dt(); if they are not implemented, the state will be stored as list().
This method can optionally be overloaded when inheriting from PipeOpTaskPreprocSimple, together with private$.transform(); alternatively, private$.get_state_dt() (optional) and private$.transform_dt() (and possibly private$.select_cols(), from PipeOpTaskPreproc) can be overloaded.
- .transform(task)
(Task) -> Task
Predict on new data in task, possibly using the stored $state. task should not be cloned, instead it should be changed in-place. This method is called both during training and prediction phase, and should essentially behave the same independently of phase. (If this is incongruent with the functionality to be implemented, then it should inherit from PipeOpTaskPreproc, not from PipeOpTaskPreprocSimple.)
This method can be overloaded when inheriting from PipeOpTaskPreprocSimple, optionally with private$.get_state(); alternatively, private$.get_state_dt() (optional) and private$.transform_dt() (and possibly private$.select_cols(), from PipeOpTaskPreproc) can be overloaded.
- .get_state_dt(dt)
(data.table) -> named list
Create something that will be stored in $state during training phase of PipeOpTaskPreprocSimple. The state can then influence the private$.transform_dt() function. Note that private$.get_state_dt() must *return* the state, and should not store it in $state. If neither private$.get_state()

nor `private$.get_state_dt()` are overloaded, the state will be stored as `list()`.
This method can optionally be overloaded when inheriting from `PipeOpTaskPreprocSimple`, together with `private$.transform_dt()` (and optionally `private$.select_cols()`, from `PipeOpTaskPreproc`); Alternatively, `private$.get_state()` (optional) and `private$.transform()` can be overloaded.

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

## See Also

https://mlr-org.com/pipeops.html

Other PipeOps: `PipeOp`, `PipeOpEnsemble`, `PipeOpImpute`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `mlr_pipeops`, `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_colroles`, `mlr_pipeops_copy`, `mlr_pipeops_datefeatures`, `mlr_pipeops_encode`, `mlr_pipeops_encodeimpact`, `mlr_pipeops_encodelmer`, `mlr_pipeops_featureunion`, `mlr_pipeops_filter`, `mlr_pipeops_fixfactors`, `mlr_pipeops_histbin`, `mlr_pipeops_ica`, `mlr_pipeops_imputeconstant`, `mlr_pipeops_imputehist`, `mlr_pipeops_imputelearner`, `mlr_pipeops_imputemean`, `mlr_pipeops_imputemedian`, `mlr_pipeops_imputemode`, `mlr_pipeops_imputeoor`, `mlr_pipeops_imputesample`, `mlr_pipeops_kernelpca`, `mlr_pipeops_learner`, `mlr_pipeops_missind`, `mlr_pipeops_modelmatrix`, `mlr_pipeops_multiplicityexply`, `mlr_pipeops_multiplicityimply`, `mlr_pipeops_mutate`, `mlr_pipeops_nmf`, `mlr_pipeops_nop`, `mlr_pipeops_ovrsplit`, `mlr_pipeops_ovrunite`, `mlr_pipeops_pca`, `mlr_pipeops_proxy`, `mlr_pipeops_quantilebin`, `mlr_pipeops_randomprojection`, `mlr_pipeops_randomresponse`, `mlr_pipeops_regravg`, `mlr_pipeops_removeconsta`, `mlr_pipeops_renamecolumns`, `mlr_pipeops_replicate`, `mlr_pipeops_scale`, `mlr_pipeops_scalemaxabs`, `mlr_pipeops_scalerange`, `mlr_pipeops_select`, `mlr_pipeops_smote`, `mlr_pipeops_spatialsign`, `mlr_pipeops_subsample`, `mlr_pipeops_targetinvert`, `mlr_pipeops_targetmutate`, `mlr_pipeops_targettrafoscale`, `mlr_pipeops_textvectorizer`, `mlr_pipeops_threshold`, `mlr_pipeops_tunethreshold`, `mlr_pipeops_unbranch`, `mlr_pipeops_updatetarget`, `mlr_pipeops_vtreat`, `mlr_pipeops_yeojohnson`

Other mlr3pipelines backend related: `Graph`, `PipeOp`, `PipeOpTargetTrafo`, `PipeOpTaskPreproc`, `mlr_graphs`, `mlr_pipeops`, `mlr_pipeops_updatetarget`

---

# 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
- a PipeOpSelect from a Selector object
- a clone of a PipeOp from a given PipeOp (possibly with changed settings)

The object is initialized with given parameters and param_vals.

po() taks a single obj (PipeOp id, Learner, ...) and converts it to a PipeOp. pos() (with plural-s) takes either a character-vector, or a list of objects, and creates a list of PipeOps.

## Usage

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

pos(.objs, ...)
```

## Arguments

| | |
|---|---|
| .obj | [any]<br>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<br>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. |
| .objs | character \| list<br>Either a character of PipeOps to look up in mlr_pipeops, or a list of other objects to be converted to a PipeOp. If this is a named list, then the names are used as $id slot for the resulting PipeOps. |

## Value

A PipeOp (for po()), or a list of PipeOps (for pos()).

## 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))

pos(c("pca", original = "nop"))
```

---

ppl *Shorthand Graph Constructor*

---

## Description

Creates a Graph from mlr_graphs from given ID

ppl() taks a character(1) and returns a Graph. ppls() takes a character vector of any list and returns a list of possibly muliple Graphs.

## Usage

```
ppl(.key, ...)

ppls(.keys, ...)
```

## Arguments

| | |
|---|---|
| .key | [character(1)]<br>The key of the Graph in mlr_graphs. |
| ... | any<br>Additional parameters to give to constructed object. This may be an argument of the constructor of the underlying function. |
| .keys | [character]<br>The key of possibly multiple Graphs in mlr_graphs. If this is named, a named list is returned, but unlike pos() it will not set any $id slots. |

## Value

Graph (for ppl()) or list of Graphs (for ppls()).

## Examples

```
library("mlr3")

gr = ppl("bagging", graph = po(lrn("regr.rpart")),
  averager = po("regravg", collect_multiplicity = TRUE))
```

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.

Conversion functions should try to avoid cloning.

## 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)

selector_missing()

selector_cardinality_greater_than(min_cardinality)
```

## Arguments

| | |
|---|---|
| `types` | (`character`)<br>Type of feature to select |
| `pattern` | (`character(1)`)<br>grep pattern |
| `ignore.case` | (`logical(1)`)<br>ignore case |
| `perl` | (`logical(1)`)<br>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.

`min_cardinality`
(`integer`)
Minimum number of levels required to be selected.

## 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 `Selector`s: Only features selected by both `Selector`s are selected in the end.
- `selector_union()`: `selector_union` selects the union of two `Selector`s: Features selected by either `Selector` are selected in the end.
- `selector_setdiff()`: `selector_setdiff` selects the setdiff of two `Selector`s: Features selected by `selector_x` are selected, unless they are also selected by `selector_y`.
- `selector_missing()`: `selector_missing` selects features with missing values.
- `selector_cardinality_greater_than()`: `selector_cardinality_greater_than` selects categorical features with cardinality greater then a given threshold.

## 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)
```

---

`set_validate.GraphLearner` *Configure Validation for a GraphLearner*

---

## Description

Configure validation for a graph learner.

In a `GraphLearner`, validation can be configured on two levels:

1. On the `GraphLearner` level, which specifies **how** the validation set is constructed before entering the graph.
2. On the level of the individual `PipeOps` (such as `PipeOpLearner`), which specifies which pipeops actually make use of the validation data (set its `$validate` field to `"predefined"`) or not (set it to `NULL`). This can be specified via the argument `ids`.

## Usage

```
## S3 method for class 'GraphLearner'
set_validate(
  learner,
  validate,
  ids = NULL,
  args_all = list(),
  args = list(),
  ...
)
```

## Arguments

| | |
|---|---|
| `learner` | (`GraphLearner`)<br>The graph learner to configure. |
| `validate` | (`numeric(1)`, `"predefined"`, `"test"`, or `NULL`)<br>How to set the `$validate` field of the learner. If set to `NULL` all validation is disabled, both on the graph learner level, but also for all pipeops. |
| `ids` | (`NULL` or `character()`)<br>For which pipeops to enable validation. This parameter is ignored when `validate` is set to `NULL`. By default, validation is enabled for the final `PipeOp` in the `Graph`. |

args_all (list())
Rarely needed. A named list of parameter values that are passed to all subsequet set_validate() calls on the individual PipeOps.

args (named list())
Rarely needed. A named list of lists, specifying additional argments to be passed to set_validate() when calling it on the individual PipeOps.

... (any)
Currently unused.

## Examples

```
library(mlr3)

glrn = as_learner(po("pca") %>>% lrn("classif.debug"))
set_validate(glrn, 0.3)
glrn$validate
glrn$graph$pipeops$classif.debug$learner$validate

set_validate(glrn, NULL)
glrn$validate
glrn$graph$pipeops$classif.debug$learner$validate

set_validate(glrn, 0.2, ids = "classif.debug")
glrn$validate
glrn$graph$pipeops$classif.debug$learner$validate
```

---

# %>>% *PipeOp Composition Operator*

---

## Description

These operators creates a connection that "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()).
%>>% and %>>!% try 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.
%>>% always creates 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. %>>!%, on the other hand, tries to avoid cloning its first argument: If it is a Graph, then this Graph will be modified in-place.

When %>>!% fails, then it leaves g1 in an incompletely modified state. It is therefore usually recommended to use %>>%, since the very marginal gain of performance from using %>>!% often does not outweigh the risk of either modifying objects by-reference that should not be modified or getting graphs that are in an incompletely modified state. However, when creating long Graphs, chaining with %>>!% instead of %>>% can give noticeable performance benefits because %>>% makes a number of clone()-calls that is quadratic in chain length, %>>!% only linear.

concat_graphs(g1, g2, in_place = FALSE) is equivalent to g1 %>>% g2. concat_graphs(g1, g2, in_place = TRUE) is equivalent to g1 %>>!% g2.

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. If the first argument of %>>!% is not a Graph, then it is cloned just as when %>>% is used; %>>!% only avoids clone() if the first argument is a Graph.

Note that if g1 is NULL, g2 converted to a Graph will be returned. Analogously, if g2 is NULL, g1 converted to a Graph will be returned.

## Usage

```
g1 %>>% g2

concat_graphs(g1, g2, in_place = FALSE)

g1 %>>!% g2
```

## Arguments

| | |
|---|---|
| g1 | (Graph \| PipeOp \| Learner \| Filter \| list \| ...)<br>Graph / PipeOp / object-convertible-to-PipeOp to put in front of g2. |
| g2 | (Graph \| PipeOp \| Learner \| Filter \| list \| ...)<br>Graph / PipeOp / object-convertible-to-PipeOp to put after g1. |
| in_place | (logical(1))<br>Whether to try to avoid cloning g1. If g1 is not a Graph, then it is cloned regardless. |

## Value

Graph: the constructed Graph.

## See Also

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

## 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)$
  add_pipeop(o2)$
  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)$
  add_pipeop(o2)$
  add_pipeop(o3)$
  add_edge(o1$id, o3$id, dst_channel = 1)$
  add_edge(o2$id, o3$id, dst_channel = 2)

pipe1 %>>!% o3  # modify pipe1 in-place

pipe1  # contains o1, o2, and o3 now.

o1 %>>!% o2

o1  # not changed, becuase not a Graph.
```

# Index