Package ‘mlrCPO’

May 9, 2026

Title Composable Preprocessing Operators and Pipelines for Machine Learning

Description Toolset that enriches 'mlr' with a diverse set of preprocessing operators. Composable Preprocessing Operators (``CPO''s) are first-class R objects that can be applied to data.frames and 'mlr' ``Task''s to modify data, can be attached to 'mlr' ``Learner''s to add preprocessing to machine learning algorithms, and can be composed to form preprocessing pipelines.

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

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

License BSD_2_clause + file LICENSE

Encoding UTF-8

Depends R (>= 3.0.2), ParamHelpers (>= 1.10), mlr (>= 2.12)

Imports BBmisc (>= 1.11), stringi, checkmate (>= 1.8.3), methods, stats, utils, backports (>= 1.1.0)

Suggests care, party, rpart, mlbench, knitr, rmarkdown, testthat, mRMRe, digest, praznik, randomForestSRC, randomForest, ranger (>= 0.8.0), Rfast, FSelector, FSelectorRcpp, e1071, FNN, lintr, Hmisc, fastICA, rex

Config/testthat/edition 3

Config/testthat/parallel true

ByteCompile yes

Version 0.3.8

Collate 'CPOHelp.R' 'fauxCPOConstructor.R' 'auxiliary.R' 'ParamSetSugar.R' 'callInterface.R' 'FormatCheck.R' 'callCPO.R' 'properties.R' 'parameters.R' 'listCPO.R' 'makeCPO.R' 'CPO_applyFun.R' 'CPO_asNumeric.R' 'operators.R' 'NULLCPO.R' 'CPO_meta.R' 'CPO_cbind.R' 'CPO_collapseFact.R' 'CPO_dropConstants.R' 'CPO_dropMostlyConstants.R' 'CPO_encode.R' 'RandomForestSRC.R' 'CPO_filterFeatures.R' 'CPO_fixFactors.R' 'CPO_ica.R' 'CPO_impute.R' 'CPO_makeCols.R' 'CPO_missingIndicators.R' 'CPO_modelMatrix.R' 'CPO_pca.R'

'CPO_quantileBinNumerics.R' 'CPO_regrResiduals.R'
'CPO_responseFromSE.R' 'CPO_scale.R' 'CPO_scaleMaxAbs.R'
'CPO_scaleRange.R' 'CPO_select.R' 'CPO_smote.R'
'CPO_spatialSign.R' 'CPO_subsample.R' 'CPO_wrap.R'
'RetrafoState.R' 'attributes.R' 'auxhelp.R'
'composeProperties.R' 'doublecaret.R' 'inverter.R' 'learner.R'
'makeCPOHelp.R' 'print.R' 'zzz.R'

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Martin Binder [aut, cre],
Bernd Bischl [ctb],
Michel Lang [ctb],
Lars Kotthoff [ctb]

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

Repository CRAN

Date/Publication 2025-06-16 19:50:02 UTC

Contents

`mlrCPO-package` *Composable Preprocessing Operators*

Description

mlrCPO is a toolset that enriches `mlr` with a diverse set of preprocessing operators. Composable Preprocessing Operators (“CPO”s) are first-class R objects that can be applied to `data.frames` and `mlr Tasks` to modify data, they can be attached to `mlr Learners` to add preprocessing to machine learning algorithms, and they can be composed to form preprocessing pipelines.

mlrCPO focuses on preprocessing as part of automated machine learning pipelines. This means that it is designed with the expectation that the same preprocessing options are applied to incoming training data, and test data. A common mistake in machine learning is that a machine learning method is evaluated (e.g. using resampling) on a dataset *after* that dataset has been cleaned up and preprocessed in one go. The proper evaluation would need to consider that the preprocessing of training data may not be influenced by any information contained in the test data set. mlrCPO takes this duality into account by providing `CPO` objects that run on training data, and which then create `CPOTrained` objects that can be used on test data (or entirely new prediction data).

This focus on preprocessing is the reason for a strict separation between "Feature Operation" CPOs, "Target Operation" CPOs, and "Retrafoless" CPOs (see OperatingType). The first class only changes (predictor) features of a dataset, and does so in a way reproducible on test data. The second class only changes (outcome) target data of a dataset, and is then able to `invert` the prediction, made by a learner on new data, back to the space of the original target data. The "Retrafoless" CPO only operates during training and may only add or subtract data rows (e.g. for SMOTE-ing or subsampling), without transforming the space of either predictor or outcome variables.

CPO's design is supposed to help its user avoid bugs and errors. Therefore it often avoids doing things implicitly and relies on explicit commands e.g. for removing data or converting between datatypes. It has certain restrictions in place (e.g. CPOProperties, CPOTrainedCapability) that try to make it hard to do the wrong thing while not being in the way of the right thing.

Other packages with similar, partially overlapping functionality are recipes, dplyr, and caret.

`applyCPO` *Apply a CPO to Data*

Description

The given transformation will be applied to the data in the given `Task` or `data.frame`.

If the input data is a `data.frame`, the returned object will in most cases also be a `data.frame`, with exceptions if the applied `CPO` performs a conversion to a `Task`. If the input data is a Task, its type will only be changed to a different type of Task if the applied CPO performs such a conversion.

The `%>>%` operator can be used synonymously to apply CPO objects to data. In case of `CPORetrafo`, `predict` can be used synonymously.

Usage

```
applyCPO(cpo, task)
```

Arguments

`cpo`	[`CPO` | `CPORetrafo`] The CPO or CPORetrafo representing the operation to perform.
`task`	[`Task` | `data.frame`] The data to operate on.

Value

[Task | data.frame]. The transformed data, augmented with a inverter and possibly a retrafo tag.

Application of CPO

Application of a CPO is supposed to perform *preprocessing* on a given data set, to prepare it e.g. for model fitting with a Learner, or for other data handling tasks. When this preprocessing is performed, care is taken to make the transformation repeatable on later prediction or validation data. For this, the returned data set will have a CPORetrafo and CPOInverter object attached to it, which can be retrieved using retrafo and inverter. These can be used to perform the same transformation on new data, or to invert a prediction made with the transformed data.

An applied CPO can change the content of feature columns, target columns of Tasks, and may even change the number of rows of a given data set.

Application of CPORetrafo

Application of a CPORetrafo is supposed to perform a transformation that mirrors the transformation done before on a training data set. It should be used when trying to make predictions from new data, using a model that was trained with data preprocessed using a CPO. The predictions made may then need to be inverted. For this, the returned data set will have a CPOInverter object attached to it, which can be retrieved using inverter.

An applied CPORetrafo may change the content of feature columns and target columns of Tasks, but will never change the number or order of rows of a given data set.

See Also

Other operators: CPO, %>>%(), as.list.CPO, attachCPO(), composeCPO(), pipeCPO()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

---

as.list.CPO *Split a Pipeline into Its Constituents*

Description

Split a compound CPO or CPOTrained into a list of its constituent parts.

This is useful for inspection of pipelines, or for possible rearrangements or changes of pipelines. The resulting list can be changed and rebuilt using pipeCPO.

Usage

```
## S3 method for class 'CPOPrimitive'
as.list(x, ...)

## S3 method for class 'CPOTrained'
as.list(x, ...)
```

Arguments

x	[CPO | CPOTrained] The CPO or CPOTrained chain to split apart.
...	[any] Ignored.

Value

[list of CPO | list of CPOTrained]. The primitive constituents of x.

See Also

Other operators: CPO, %>>%(), applyCPO(), attachCPO(), composeCPO(), pipeCPO()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

---

attachCPO — *Attach a CPO to a Learner*

Description

A CPO object can be attached to a Learner object to create a pipeline combining preprocessing and model fitting. When the resulting CPOLearner is used to create a model using train, the attached CPO will be applied to the data before the internal model is trained. The resulting model will also contain the required CPOTrained elements, and apply the necessary CPORetrafo objects to new prediction data, and the CPOInverter objects to predictions made by the internal model.

The %>>% operator can be used synonymously to attach CPO objects to Learners.

Usage

```
attachCPO(cpo, learner)
```

Arguments

cpo	[CPO] The cpo.
learner	[Learner] The learner.

See Also

Other operators: CPO, %>>%(), applyCPO(), as.list.CPO, composeCPO(), pipeCPO()

Other CPO lifecycle related: CPO, CPOConstructor, CPOLearner, CPOTrained, NULLCPO, %>>%(), composeCPO(), getCPOClass(), getCPOConstructor(), getCPOTrainedCPO(), identicalCPO(), makeCPO()

Other CPOLearner related: CPOLearner, getLearnerBare(), getLearnerCPO()

---

clearRI *Clear Retrafo and Inverter Attributes*

Description

When applying CPOs to data, the operation entails saving the CPOTrained information that gets generated to an attribute of the resulting object. This is a useful solution to the problem that applying multiple CPOs should also lead to a retrafo object that performs the same multiple operations. However, sometimes this may lead to surprising and unwanted results when a CPO is applied and not meant to be part of a trafo-retrafo machine learning pipeline, e.g. for dropping columns that occur in training but not in prediction data. In that case, it is necessary to reset the `retrafo` and possibly `inverter` attributes of the data being used. This can be done either by using `retrafo(data) <- NULL`, or by using `clearRI`. `clearRI` clears both `retrafo` *and* `inverter` attributes.

Usage

```
clearRI(data)
```

Arguments

data	[data.frame | Task | WrappedModel] The result of a CPO applied to a data set.

Value

[data.frame | Task | WrappedModel] the `data` after stripping all `retrafo` and `inverter` attributes.

See Also

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Examples

```
# without clearRI
transformed = iris.task %>>% cpoPca()
transformed2 = transformed %>>% cpoScale()
retrafo(transformed2)  # [RETRAFO pca]=>[RETRAFO scale]

transformed = iris.task %>>% cpoPca()
transformed2 = clearRI(transformed) %>>% cpoScale()
retrafo(transformed2)  # [RETRAFO scale]
```

---

composeCPO *CPO Composition*

Description

Composes CPO or CPOTrained objects. The %>>% operator can be used synonymously to compose CPO objects.

Composition of operators is one of the main features they provide: this makes it possible for complex operations to be represented by single objects. Compound operators represent the operation of applying both its constituent operations in succession. Compound operators can themselves be composed to form arbitrarily long chains of operators.

Compound objects behave, in most ways, like primitive objects. Some exceptions are:

- Compound CPOs do not have an ID, so getCPOId and setCPOId will not work on them.
- Compound CPOs have no 'affect' property, so getCPOAffect will not work.

While CPOTrained operators can be composed just as CPO operators, this is only recommended in cases where the same primitive CPOTrained objects where retrieved using as.list.CPOTrained. This is because CPOTrained are closely related to the data that was used to create it, and therefore on their original position in the CPO pipeline during training.

Usage

```
composeCPO(cpo1, cpo2)
```

Arguments

- `cpo1` — [`CPO` | `CPOTrained`]
  The operation to perform first.
- `cpo2` — [`CPO` | `CPOTrained`]
  The operation to perform second, must have the same class as `cpo1`.

Value

[`CPO` | `CPOTrained`]. The operation representing the application of `cpo1` followed by the application of `cpo2`.

See Also

Other operators: `CPO`, `%>>%()`, `applyCPO()`, `as.list.CPO`, `attachCPO()`, `pipeCPO()`

Other CPO lifecycle related: `CPO`, `CPOConstructor`, `CPOLearner`, `CPOTrained`, `NULLCPO`, `%>>%()`, `attachCPO()`, `getCPOClass()`, `getCPOConstructor()`, `getCPOTrainedCPO()`, `identicalCPO()`, `makeCPO()`

`covrTraceCPOs` *Add 'covr' coverage to CPOs*

Description

Use this if you want to check code coverage of `CPO`s using `covr`. The functions inside `CPO`s is originally not accessible to `covr`, so `covrTraceCPOs` needs to be called in the `.onAttach` function. Note that putting it in `.onLoad` will **not** work.

Currently, for this to work, the mb706 fork of `covr` needs to be used. To install it, call

`devtools::install_github("mb706/covr")`

To use it on Travis CI, add the line `- mb706/covr` under the `r_github_packages:` category.

This function comes at no runtime penalty: If the `R_COVR` environment variable is not set to "true", then it only has an effect if `force` is `TRUE`.

Usage

```
covrTraceCPOs(env = parent.env(parent.frame()), force = FALSE)
```

Arguments

- `env` — [environment]
  The environment to search for `CPO`s. Default is `parent.env(parent.frame())`, which is the package namespace if called from `.onLoad`.
- `force` — [logical(1)]
  Trace `CPO` functions even when `R_COVR` is not "true". Default is `FALSE`.

Value

[invisible(NULL)].

CPO *Composable Preprocessing Operators*

Description

Composable Preprocessing Operators, or CPO, are the central entity provided by the mlrCPO package. CPOs can perform operations on a data.frame or a Task, for the latter even modifying target values and converting between different Task types.

CPOs can be "composed" using the %>>% operator, the composeCPO function, or the pipeCPO function, to create new ("compound") operators that perform multiple operations in a pipeline. While all CPOs have the class "CPO", primitive (i.e. not compound) CPOs have the additional class "CPOPrimitive", and compound CPOs have the class "CPOPipeline". It is possible to split a compound CPOs into its primitive constituents using as.list.CPO.

CPOs can be "attached" to a mlr-Learner objects to create CPOLearners, using the %>>% operator, or the attachCPO function. These CPOLearners fit the model specified by the Learner to the data after applying the attached CPO. Many CPOs can be attached to a Learner sequentially, or in form of a compound CPO.

CPOs can be "applied" to a data.frame or a Task using the %>>% operator, or the applyCPO function. Applying a CPO performs the operations specified by the (possibly compound) CPO, and returns the modified data. This data also contains a "retrafo" and and "inverter" tag, which can be accessed using the retrafo and inverter functions to get CPORetrafo and CPOInverter objects, respectively. These objects represent the "trained" CPOs that can be used when performing validation or predictions with new data.

Hyperparameters

CPOs can have hyperparameters that determine how they operate on data. These hyperparameters can be set during construction, as function parameters of the CPOConstructor, or they can potentially be modified later as exported hyperparameters. Which hyperparameters are exported is controlled using the export parameter of the CPOConstructor when the CPO was created. Hyperparameters can be listed using getParamSet, queried using getHyperPars and set using setHyperPars.

S3 properties

A CPO object should be treated as an opaque object and should only be queried / modified using the given set* and get* functions. A list of them is given below in the section "See Also"–"cpo-operations".

Special CPO

A special CPO is NULLCPO, which functions as the neutral element of the %>>% operator and represents the identity operation on data.

See Also

print.CPO for possibly verbose printing.

Other CPO lifecycle related: CPOConstructor, CPOLearner, CPOTrained, NULLCPO, %>>%(), attachCPO(), composeCPO(), getCPOClass(), getCPOConstructor(), getCPOTrainedCPO(), identicalCPO(), makeCPO()

Other operators: %>>%(), applyCPO(), as.list.CPO, attachCPO(), composeCPO(), pipeCPO()

Other getters and setters: getCPOAffect(), getCPOClass(), getCPOConstructor(), getCPOId(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), setCPOId()

Other CPO classifications: getCPOClass(), getCPOOperatingType(), getCPOTrainedCapability()

Examples

```
class(cpoPca())  # c("CPOPrimitive", "CPO")
class(cpoPca() %>>% cpoScale())  # c("CPOPipeline", "CPO")
print(cpoPca() %>>% cpoScale(), verbose = TRUE)

getHyperPars(cpoScale(center = FALSE))

head(getTaskData(iris.task %>>% cpoScale()))
```

---

cpoApplyFun *Apply a Function Element-Wise*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

The function must either vectorize over the given data, or will be applied to each data element on its own.

It must not change the type of the data, i.e. numeric data must remain numeric etc.

If the function can only handle a subset of the given columns, e.g. only a certain type, use `affect.*` arguments.

Usage

```
cpoApplyFun(
  fun,
  param = NULL,
  vectorize = TRUE,
  make.factors = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
```

```
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`fun` [function]
The function to apply. If `vectorize` is `TRUE`, the argument is a vector of the whole column, `fun` must vectorize over it and return a vector of the same length; otherwise, the function gets called once for every data item, and both the function argument and the return value must have length 1.

The function must take one or two arguments. If it takes two arguments, the second argument will be `param`.

`param` [any]
Optional argument to be given to `fun`. If `fun` only takes one argument, this is ignored. Default is `NULL`.

`vectorize` [logical(1)]
Whether to call `fun` once for each column, or once for each element. If `fun` vectorizes, it is recommended to have this set to `TRUE` for better performance. Default is `TRUE`.

`make.factors` [logical(1)]
Whether to turn resulting `logical` and `character` columns into `factor` columns (which are preferred by `mlr`). Default is `TRUE`.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern
: [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

CPOTrained State

The created state is empty.

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoApplyFunRegrTarget *Transform a Regression Target Variable*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Apply a given function to the target column of a regression Task.

Usage

```
cpoApplyFunRegrTarget(
  trafo,
  invert.response = NULL,
  invert.se = NULL,
  param = NULL,
  vectorize = TRUE,
  gauss.points = 23,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

trafo
: [function]
A function transforming the target column. If vectorize is TRUE, the argument is a vector of the whole column, trafo must vectorize over it and return a vector of the same length; otherwise, the function gets called once for every data item, and both the function argument and the return value must have length 1.

  The function must take one or two arguments. If it takes two arguments, the second argument will be param.

invert.response
: [function]
If a model is trained on data that was transformed by trafo, this function should invert a prediction made by this model back to the space of the original data. In most cases, this will be the inverse of trafo, so that invert.response(trafo(x)) == x.

  Similarly to trafo, this function takes / produces single elements or the whole column, depending on vectorize. The return value should be a numeric in both cases.

  This can also be NULL, in which case using this CPO for invert with predict.type = "response" is not possible.

  Default is NULL.

invert.se
: [function]
Similarly to invert.response, this is a function that inverts a "se" prediction made after training on trafo'd data. This function should take at least two arguments, mean and se, and return a numeric vector of length 2 if vectorize is FALSE, or a data.frame or matrix with two numeric columns if vectorize is TRUE. The function may also take a third argument, which will be set to param.

  invert.se may also be NULL, in which case "se" inversion is done by numeric integration using Gauss-Hermite quadrature.

  Default is NULL.

param
: [any]
Optional argument to be given to trafo and / or invert. If both of them only take one argument, this is ignored. Default is NULL.

vectorize
: [logical(1)]
Whether to call trafo, invert.response and invert.se once with the whole data column (or response *and* se column if predict.type == "se"), or once for each element. If the functions vectorize, it is recommended to have this set to TRUE for better performance. Default is TRUE.

gauss.points
: [numeric(1)]
Number of points at which to evaluate invert.response for Gauss-Hermite quadrature integration. Only used if invert.se is NULL. Default is 23.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

affect.pattern
: [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

Details

When both `mean` and `se` prediction is available, it may be possible to make more accurate mean inversion than for the `response predict.type`, using integrals or approximations like the *delta method*. In such cases it may be advisable to prepend this `CPO` with the `cpoResponseFromSE` `CPO`.

Note when `trafo` or `invert.response` take more than one argument, the second argument will be set to the value of `param`. This may lead to unexpected results when using functions with rarely used parameters, e.g. log. In these cases, it may be necessary to wrap the function: trafo = function(x) log(x).

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoAsNumeric *Convert All Features to Numerics*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Converts all feature columns to (integer) numeric columns by applying as.numeric to them.

Usage

```
cpoAsNumeric(
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`id` [`character(1)`]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [`character`]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [`character | NULL`]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [`numeric`]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [`character`]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [`character(1) | NULL`]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [`logical(1)`]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[`logical(1)`]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

cpoCache *Caches the Result of CPO Transformations*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Given a CPO to wrap, this caches an intermediate result (in fact, the retrafo object) whenever the CPO is applied to a Task or data.frame. This can reduce computation time when the same CPO is often applied to the same data, e.g. in a resampling or tuning evaluation.

The hyperparameters of the CPO are not exported, since in many cases changing the hyperparameters will also change the result and would defeat the point of caching. To switch between different settings of the same CPO, consider using cpoMultiplex.

The cache is kept in an environment; therefore, it does not communicate with other threads or processes when using parallelization at a level before the cache gets filled.

Caching needs the 'digest' package to be installed.

Usage

```
cpoCache(cpo = NULLCPO, cache.entries = 1024)
```

Arguments

cpo	[CPO] The CPO to wrap. The CPO may only have a single OperatingType. Default is NULLCPO.
cache.entries	[numeric(1)] Number of entries in the least recently used cache.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

---

cpoCbind — *"cbind" the Result of Multiple CPOs*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Build a `CPO` that represents the operations of its input parameters, performed in parallel and put together column wise.

For example, to construct a `Task` that contains the original data, as well as the data after scaling, one could do

`task %>>% cpoCbind(NULLCPO, cpoScale())`

The result of `cpoCbind` is itself a CPO which exports its constituents' hyperparameters. CPOs with the same type / ID get combined automatically. To get networks, e.g. of the form

```
      ,-C--E-.
     /    /   \
A---B----D-----F---G
```

one coul use the code

```
initcpo = A %>>% B
route1 = initcpo %>>% D
route2 = cpoCbind(route1, initcpo %>>% C) %>>% E
result = cpoCbind(route1, route2) %>>% F %>>% G
```

`cpoCbind` finds common paths among its arguments and combines them into one operation. This saves computation and makes it possible for one exported hyperparameter to influence multiple of `cpoCbind`'s inputs. However, if you want to use the same operation with different parameters on different parts of `cpoCbind` input, you must give these operations different IDs. If CPOs that could represent an identical CPO, with the same IDs (or both with IDs absent) but different parameter settings, `affect.*` settings or different parameter exportations occur, an error will be thrown.

Usage

```
cpoCbind(..., .cpos = list())
```

Arguments

`...`	[`CPO`] The `CPO`s to cbind. These must be Feature Operation CPOs. Named arguments will result in the respective columns being prefixed with the name. This is highly recommended if there is any chance of name collision otherwise. it is possible to use the same name multiple times (provided the resulting column names don't clash).
`.cpos`	[`list` of `CPO`] Alternatively, give the CPOs to cbind as a list. Default is `list()`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

Other special CPOs: cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoCollapseFact *Compine Rare Factors*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Combine rare factor levels into a single factor level.

Usage

```
cpoCollapseFact(
  max.collapsed.class.prevalence = 0.1,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`max.collapsed.class.prevalence`
: [numeric(1)]
  Maximum prevalence of newly created collapsed factor level. Default is `0.1`.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`CPOConstructor` *Constructor for CPO Objects*

Description

`CPO` objects are created by calling `CPOConstructor`s, which are R functions that have some parameters in common, use a convenient `print.CPOConstructor` generic, and always return a `CPO` object. The mlrCPO package provides many `CPOConstructor` functions, which can be listed using `listCPO`. It is also possible to create custom `CPOConstructor`s using `makeCPO`, `makeCPORetrafoless`, `link{makeCPOTargetOp}`, and `makeCPOExtendedTrafo`.

Arguments

`id` [`character(1)` | `NULL`]
ID to use for the CPO. if `NULL` is given, this defaults to a name describing the action performed by the CPO, which can be retrieved using `getCPOName`. The ID is used to identify the CPO in print messages, and is prefixed to the CPO's

hyperparameter names. This is can be used to avoid name clashes when composing a CPO with another CPO or `Learner` with hyperparameters with clashing names. Default is `NULL`.

`export`
: `[character]`
Which hyperparameters to export. This can be a character vector naming the hyperparameters to export (*excluding* the ID), or a `character(1)` with one of the special values:

"export.all"	export all parameters
"export.default"	exp. params that are exp. by def
"export.set"	exp. params set in construct call
"export.default.set"	intersection of "default" and "set"
"export.unset"	params *not* set in construct call
"export.default.unset"	isct. of "default" and "unset"
"export.all.plus"	not yet supported

Default is "export.default".

`affect.type`
: `[character | NULL]`
Type of columns to affect. May be a subset of "numeric", "factor", "ordered", "other", or can be or `NULL` to match all columns. Default is `NULL`.

`affect.index`
: `[numeric]`
Indices of feature columns to affect. The order of indices given is respected. Default is `integer(0)`.

`affect.names`
: `[character]`
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern`
: `[character(1) | NULL]`
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert`
: `[logical(1)]`
Whether to affect all features *not* matched by other `affect.*` parameters. Default is `FALSE`.

`affect.pattern.ignore.case`
: `[logical(1)]`
Ignore case when matching features with `affect.pattern`; see `grep`. Has no effect when `affect.pattern` is `NULL`. Default is `FALSE`.

`affect.pattern.perl`
: `[logical(1)]`
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Has no effect when `affect.pattern` is `NULL`, or when `affect.pattern.fixed` is `TRUE`. Default is `FALSE`.

`affect.pattern.fixed`
: `[logical(1)]`
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Has no effect when `affect.pattern` is `NULL`. Default is `FALSE`.

Value

`[CPO]` the constructed CPO.

CPO creation

CPOConstructors can be called like any R function, with any parameters given. Besides parameters that are common to most CPOConstructors (listed below), it is possible to set CPO-specific hyperparameters in the construction. Parameters that are being *exported* can also be modified later using the `CPO` object, see the documentation there.

`affect.*` parameters

When creating a `CPO`, it is possible to choose which columns of the given data the CPO operates on, and which columns it will ignore. This is done using the `affect.*` parameters. It is possible to choose columns by types, indices, names, or a regular expression matching names.

See Also

`print.CPOConstructor` for possibly verbose printing.

Other CPO lifecycle related: `CPO`, `CPOLearner`, `CPOTrained`, `NULLCPO`, `%>>%()`, `attachCPO()`, `composeCPO()`, `getCPOClass()`, `getCPOConstructor()`, `getCPOTrainedCPO()`, `identicalCPO()`, `makeCPO()`

Other CPOConstructor related: `getCPOClass()`, `getCPOConstructor()`, `getCPOName()`, `identicalCPO()`, `makeCPO()`, `print.CPOConstructor()`

Examples

```
class(cpoPca)  # c("CPOConstructor", "function")
print(cpoPca)  # default printer
print(cpoPca, verbose = TRUE)  # shows the trafo / retrafo functions

cpoPca()  # creating a CPO
class(cpoPca())  # c("CPOPrimitive", "CPO")
```

`cpoDropConstants` *Drop Constant or Near-Constant Features*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Drop all columns that are either constant, or close to constant for numerics, and columns that have only one value for factors or ordered columns.

Usage

```
cpoDropConstants(
  rel.tol = 1e-08,
  abs.tol = 1e-08,
  ignore.na = FALSE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`rel.tol`	`[numeric(1)]` Relative tolerance within which to consider a feature constant. Set to 0 to disregard relative tolerance. Default is `1e-8`.
`abs.tol`	`[numeric(1)]` Absolute tolerance within which to consider a feature constant. Set to 0 to disregard absolute tolerance. Default is `1e-8`.
`ignore.na`	`[logical(1)]` Whether to ignore `NA` and `NaN` values. If this is `TRUE`, values that are `NA` or `NaN` will not be counted as different from any other value. If this is `FALSE`, columns with `NA` or `NaN` in them will only count as constant if they are entirely made up of `NA`, or entirely made up of `NaN`. Default is `FALSE`.
`id`	`[character(1)]` id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
`export`	`[character]` Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".
`affect.type`	`[character	NULL]` Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoDropMostlyConstants

Drop Constant or Near-Constant Features

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Drop all columns that are mostly constant: Constant within tolerance with numerics, and and columns that have only one value for factors or ordered columns.

This CPO can also filter "mostly" constant Features: ones where at most a fraction of `ratio` samples differ from the mode value.

Usage

```
cpoDropMostlyConstants(
  ratio = 0,
  rel.tol = 1e-08,
  abs.tol = 1e-08,
  ignore.na = FALSE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

- `ratio` — [numeric(1)]
  Minimum ratio of values which must be different from the mode value in order to keep a feature in the task. Default is 0, which means only constant features with exactly one observed level are removed.
- `rel.tol` — [numeric(1)]
  Relative tolerance within which to consider a feature constant. Set to 0 to disregard relative tolerance. Default is `1e-8`.
- `abs.tol` — [numeric(1)]
  Absolute tolerance within which to consider a feature constant. Set to 0 to disregard absolute tolerance. Default is `1e-8`.
- `ignore.na` — [logical(1)]
  Whether to ignore `NA` and `NaN` values. If this is `TRUE`, values that are `NA` or `NaN` will not be counted as different from any other value. If this is `FALSE`, columns with `NA` or `NaN` in them will only count as constant if they are entirely made up of `NA`, or entirely made up of `NaN`. Default is `FALSE`.
- `id` — [character(1)]
  id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
- `export` — [character]
  Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".
- `affect.type` — [character | NULL]
  Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.
- `affect.index` — [numeric]
  Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.
- `affect.names` — [character]
  Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.
- `affect.pattern` — [character(1) | NULL]
  `grep` pattern to match feature names by. Default is `NULL` (no pattern matching)
- `affect.invert` — [logical(1)]
  Whether to affect all features *not* matched by other `affect.*` parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(),

cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoDummyEncode *CPO Dummy Encoder*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Usage

```
cpoDummyEncode(
  reference.cat = FALSE,
  infixdot = FALSE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

reference.cat	[logical] If "reference.cat" is TRUE, the first level of every factor column is taken as the reference category and the encoding is c(0, 0, 0, ...). If this is FALSE, the encoding is always one-hot-encoding. Default is FALSE.
infixdot	[logical] Whether to add an infix dot when creating names. This is nicer in some ways, but is not compatible with model.matrix.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
export	[character] Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default"

(export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUnce` `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

---

cpoFilterAnova *Filter Features: “anova.test”*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Filter “anova.test” is based on the Analysis of Variance (ANOVA) between feature and class. The value of the F-statistic is used as a measure of feature importance.

Usage

```
cpoFilterAnova(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
```

```
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`perc` [numeric(1)]
If set, select `perc`*100 top scoring features. Mutually exclusive with arguments `abs` and `threshold`.

`abs` [numeric(1)]
If set, select `abs` top scoring features. Mutually exclusive with arguments `perc` and `threshold`.

`threshold` [numeric(1)]
If set, select features whose score exceeds `threshold`. Mutually exclusive with arguments `perc` and `abs`.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other filter: cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(),

cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoFilterCarscore *Filter Features: "carscore"*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Filter "carscore" determines the "Correlation-Adjusted (marginal) coRelation scores" (short CAR scores). The CAR scores for a set of features are defined as the correlations between the target and the decorrelated features.

Usage

```
cpoFilterCarscore(
  diagonal = FALSE,
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

diagonal	[logical(1)] See the carscore help.
perc	[numeric(1)] If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.
abs	[numeric(1)] If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.
threshold	[numeric(1)] If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: cpoFilterAnova(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

`cpoFilterChiSquared` *Filter Features: "chi.squared"*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

The chi-square test is a statistical test of independence to determine whether two variables are independent. Filter "chi.squared" applies this test in the following way. For each feature the chi-square test statistic is computed checking if there is a dependency between the feature and the target variable. Low values of the test statistic indicate a poor relationship. High values, i.e., high dependency identifies a feature as more important.

Usage

```
cpoFilterChiSquared(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`perc`	`[numeric(1)]` If set, select `perc`*100 top scoring features. Mutually exclusive with arguments `abs` and `threshold`.
`abs`	`[numeric(1)]` If set, select `abs` top scoring features. Mutually exclusive with arguments `perc` and `threshold`.
`threshold`	`[numeric(1)]` If set, select features whose score exceeds `threshold`. Mutually exclusive with arguments `perc` and `abs`.
`id`	`[character(1)]` id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
`export`	`[character]` Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".
`affect.type`	`[character	NULL]` Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.
`affect.index`	`[numeric]` Indices of feature columns to affect. The order of indices given is respected. Tar-

	get column indices are not counted (since target columns are always included). Default is `integer(0)`.
`affect.names`	`[character]` Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.
`affect.pattern`	`[character(1)` | `NULL]` `grep` pattern to match feature names by. Default is `NULL` (no pattern matching)
`affect.invert`	`[logical(1)]` Whether to affect all features *not* matched by other `affect.*` parameters.
`affect.pattern.ignore.case`	`[logical(1)]` Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.
`affect.pattern.perl`	`[logical(1)]` Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.
`affect.pattern.fixed`	`[logical(1)]` Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGa cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoFilterFeatures *Filter Features by Thresholding Filter Values*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

First, calls generateFilterValuesData. Features are then selected via select and val.

Usage

```
cpoFilterFeatures(
  method = "randomForestSRC.rfsrc",
  fval = NULL,
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  filter.args = list(),
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

method	[character(1)] See listFilterMethods. Default is “randomForestSRC.rfsrc”.
fval	[FilterValues] Result of generateFilterValuesData. If you pass this, the filter values in the object are used for feature filtering. method and ... are ignored then. Default is NULL and not used.
perc	[numeric(1)] If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.
abs	[numeric(1)] If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.
threshold	[numeric(1)] If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.
filter.args	[list] Passed down to selected filter method. Default is list().
id	[character(1)] id to use as prefix for the CPO’s hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
export	[character] Either a character vector indicating the parameters to export as hyperparameters, or one of the special values “export.all” (export all parameters), “export.default” (export all parameters that are exported by default), “export.set” (export all parameters that were set during construction), “export.default.set” (export the intersection of the “default” and “set” parameters), “export.unset” (export all parameters that were *not* set during construction) or “export.default.unset” (export the intersection of the “default” and “unset” parameters). Default is “export.default”.
affect.type	[character | NULL] Type of columns to affect. A subset of “numeric”, “factor”, “ordered”, “other”, or NULL to not match by column type. Default is NULL.
affect.index	[numeric] Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).
affect.names	[character] Feature names of feature columns to affect. The order of names given is respected. Default is character(0).
affect.pattern	[character(1) | NULL] grep pattern to match feature names by. Default is NULL (no pattern matching)
affect.invert	[logical(1)] Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other filter: cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

Other filter: cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

---

cpoFilterGainRatio *Filter Features: "gain.ratio"*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Filter "gain.ratio" uses the entropy-based information gain ratio between each feature and target individually as an importance measure.

Usage

```
cpoFilterGainRatio(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc [numeric(1)]
If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.

- `abs` — [numeric(1)]
  If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.
- `threshold` — [numeric(1)]
  If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.
- `id` — [character(1)]
  id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
- `export` — [character]
  Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".
- `affect.type` — [character | NULL]
  Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.
- `affect.index` — [numeric]
  Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).
- `affect.names` — [character]
  Feature names of feature columns to affect. The order of names given is respected. Default is character(0).
- `affect.pattern` — [character(1) | NULL]
  grep pattern to match feature names by. Default is NULL (no pattern matching)
- `affect.invert` — [logical(1)]
  Whether to affect all features *not* matched by other affect.* parameters.
- `affect.pattern.ignore.case` — [logical(1)]
  Ignore case when matching features with affect.pattern; see grep. Default is FALSE.
- `affect.pattern.perl` — [logical(1)]
  Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.
- `affect.pattern.fixed` — [logical(1)]
  Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other filter: cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

cpoFilterInformationGain

Filter Features: "information.gain"

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Filter "information.gain" uses the entropy-based information gain between each feature and target individually as an importance measure.

Usage

```
cpoFilterInformationGain(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc	[numeric(1)] If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.
abs	[numeric(1)] If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.
threshold	[numeric(1)] If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUnce` `cpoFilterUnivariate()`, `cpoFilterVariance()`, `randomForestSRC_filters`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUnce` `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`cpoFilterKruskal` *Filter Features: “kruskal.test”*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Filter “kruskal.test” applies a Kruskal-Wallis rank sum test of the null hypothesis that the location parameters of the distribution of a feature are the same in each class and considers the test statistic as an variable importance measure: if the location parameters do not differ in at least one case, i.e., the null hypothesis cannot be rejected, there is little evidence that the corresponding feature is suitable for classification.

Usage

```
cpoFilterKruskal(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`perc`
: [numeric(1)]
If set, select `perc`*100 top scoring features. Mutually exclusive with arguments `abs` and `threshold`.

`abs`
: [numeric(1)]
If set, select `abs` top scoring features. Mutually exclusive with arguments `perc` and `threshold`.

`threshold`
: [numeric(1)]
If set, select features whose score exceeds `threshold`. Mutually exclusive with arguments `perc` and `abs`.

`id`
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export`
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type`
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index`
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Tar-

get column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
grep pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see grep. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see grep. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see grep. Default is `FALSE`.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoFilterLinearCorrelation

Filter Features: "linear.correlation"

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

The Pearson correlation between each feature and the target is used as an indicator of feature importance. Rows with NA values are not taken into consideration.

Usage

```
cpoFilterLinearCorrelation(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc
: [numeric(1)]
If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.

abs
: [numeric(1)]
If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.

threshold
: [numeric(1)]
If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern
: [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other filter: cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

cpoFilterMrmr *Filter Features: "mrmr"*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Minimum redundancy, maximum relevance filter "mrmr" computes the mutual information between the target and each individual feature minus the average mutual information of previously selected features and this feature using the **mRMRe** package.

Usage

```
cpoFilterMrmr(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc	[numeric(1)] If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.
abs	[numeric(1)] If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.
threshold	[numeric(1)] If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUnce`
`cpoFilterUnivariate()`, `cpoFilterVariance()`, `randomForestSRC_filters`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUnce`
`cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`cpoFilterOneR` *Filter Features: “oneR”*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Filter “oneR” makes use of a simple “One-Rule” (OneR) learner to determine feature importance. For this purpose the OneR learner generates one simple association rule for each feature in the data individually and computes the total error. The lower the error value the more important the correspoding feature.

Usage

```
cpoFilterOneR(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`perc`
: [numeric(1)]
If set, select `perc`*100 top scoring features. Mutually exclusive with arguments `abs` and `threshold`.

`abs`
: [numeric(1)]
If set, select `abs` top scoring features. Mutually exclusive with arguments `perc` and `threshold`.

`threshold`
: [numeric(1)]
If set, select features whose score exceeds `threshold`. Mutually exclusive with arguments `perc` and `abs`.

`id`
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export`
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type`
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index`
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Tar-

get column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see grep. Default is FALSE.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see grep. Default is FALSE.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, cpoFilterMrmr(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoFilterPermutationImportance

Filter Features: "permutation.importance"

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Filter "permutation.importance" computes a loss function between predictions made by a learner before and after a feature is permuted.

Usage

```
cpoFilterPermutationImportance(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  imp.learner,
  contrast = function(x, y) {
     x - y
 },
  measure = NULL,
  aggregation = function(x, ...) UseMethod("mean"),
  nmc = 50,
  replace = FALSE,
  id,
  export = "export.default",
  affect.type = NULL,
```

```
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`perc` [numeric(1)]
If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.

`abs` [numeric(1)]
If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.

`threshold` [numeric(1)]
If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.

`imp.learner` [Learner | character(1)]
Specifies the learner to use when computing the permutation importance.

`contrast` [function]
Contrast: takes two numeric vectors and returns one (default is the difference).

`measure` [Measure]
Measure to use. Defaults to the default measure of the task.

`aggregation` [function]
Aggregation: takes a numeric and returns a numeric(1) (default is the mean).

`nmc` [integer(1)]

`replace` [logical(1)]
Determines whether the feature being permuted is sampled with or without replacement.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern
: [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other filter: cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoFilterRankCorrelation

Filter Features: "rank.correlation"

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

The Spearman correlation between each feature and the target is used as an indicator of feature importance. Rows with NA values are not taken into consideration.

Usage

```
cpoFilterRankCorrelation(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
```

```
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`perc`
: [numeric(1)]
If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.

`abs`
: [numeric(1)]
If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.

`threshold`
: [numeric(1)]
If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.

`id`
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export`
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type`
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

`affect.index`
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

`affect.names`
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

`affect.pattern`
: [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

`affect.invert`
: [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

`affect.pattern.ignore.case`
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other filter: cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUnce cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(),

cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoFilterRelief *Filter Features: “relief”*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Filter “relief” is based on the feature selection algorithm “ReliefF” by Kononenko et al., which is a generalization of the orignal “Relief” algorithm originally proposed by Kira and Rendell. Feature weights are initialized with zeros. Then for each instance `sample.size` instances are sampled, `neighbours.count` nearest-hit and nearest-miss neighbours are computed and the weight vector for each feature is updated based on these values.

Usage

```
cpoFilterRelief(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc	[numeric(1)] If set, select `perc`*100 top scoring features. Mutually exclusive with arguments `abs` and `threshold`.
abs	[numeric(1)] If set, select `abs` top scoring features. Mutually exclusive with arguments `perc` and `threshold`.
threshold	[numeric(1)] If set, select features whose score exceeds `threshold`. Mutually exclusive with arguments `perc` and `abs`.

- `id` `[character(1)]`
  id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
- `export` `[character]`
  Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".
- `affect.type` `[character | NULL]`
  Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.
- `affect.index` `[numeric]`
  Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.
- `affect.names` `[character]`
  Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.
- `affect.pattern` `[character(1) | NULL]`
  `grep` pattern to match feature names by. Default is `NULL` (no pattern matching)
- `affect.invert` `[logical(1)]`
  Whether to affect all features *not* matched by other `affect.*` parameters.
- `affect.pattern.ignore.case`
  `[logical(1)]`
  Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.
- `affect.pattern.perl`
  `[logical(1)]`
  Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.
- `affect.pattern.fixed`
  `[logical(1)]`
  Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

References

Kira, Kenji and Rendell, Larry (1992). The Feature Selection Problem: Traditional Methods and a New Algorithm. AAAI-92 Proceedings.

Kononenko, Igor et al. Overcoming the myopia of inductive learning algorithms with RELIEFF (1997), Applied Intelligence, 7(1), p39-55.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUnce` `cpoFilterUnivariate()`, `cpoFilterVariance()`, `randomForestSRC_filters`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUnce` `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

cpoFilterRfCImportance
Filter Features: "cforest.importance"

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Permutation importance of random forests fitted in package **party**. The implementation follows the principle of mean decrese in accuracy used by the **randomForest** package (see description of "randomForest.importance") filter.

Usage

```
cpoFilterRfCImportance(
  mtry = 5,
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

mtry	[integer(1)] Number of features to draw during feature bagging
perc	[numeric(1)] If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.
abs	[numeric(1)] If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.
threshold	[numeric(1)] If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `randomForestSRC_filters`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`cpoFilterRfImportance` *Filter Features: "randomForest.importance"*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Filter "randomForest.importance" makes use of the `importance` from package **randomForest**. The importance measure to use is selected via the `method` parameter:

oob.accuracy Permutation of Out of Bag (OOB) data.

node.impurity Total decrease in node impurity.

Usage

```
cpoFilterRfImportance(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc	[numeric(1)] If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.
abs	[numeric(1)] If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.
threshold	[numeric(1)] If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
export	[character] Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".
affect.type	[character | NULL] Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.
affect.index	[numeric] Indices of feature columns to affect. The order of indices given is respected. Tar-

get column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoFilterRfSRCImportance

Filter Features: "randomForestSRC.rfsrc"

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Filter "randomForestSRC.rfsrc" computes the importance of random forests fitted in package **randomForestSRC**. The concrete method is selected via the method parameter. Possible values are permute (default), random, anti, permute.ensemble, random.ensemble, anti.ensemble. See the VIMP section in the docs for rfsrc for details.

Usage

```
cpoFilterRfSRCImportance(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc
: [numeric(1)]
If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.

abs
: [numeric(1)]
If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.

threshold
: [numeric(1)]
If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern
: [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other filter: cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

cpoFilterSymmetricalUncertainty
Filter Features: "symmetrical.uncertainty"

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Filter "symmetrical.uncertainty" uses the entropy-based symmetrical uncertainty between each feature and target individually as an importance measure.

Usage

```
cpoFilterSymmetricalUncertainty(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc	[numeric(1)] If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.
abs	[numeric(1)] If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.
threshold	[numeric(1)] If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `randomForestSRC_filters`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`cpoFilterUnivariate` *Filter Features: “univariate.model.score”*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

The “univariate.model.score” feature filter resamples an **mlr** learner specified via `perf.learner` for each feature individually with randomForest from package **rpart** being the default learner. Further parameter are the resamling strategey `perf.resampling` and the performance measure `perf.measure`.

Usage

```
cpoFilterUnivariate(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  perf.learner = NULL,
  perf.measure = NULL,
  perf.resampling = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc	[numeric(1)] If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.
abs	[numeric(1)] If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.
threshold	[numeric(1)] If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.
perf.learner	[Learner | NULL] Learner to resample. If this is NULL, regr.randomForest is used. Default is NULL.
perf.measure	[Measure | NULL] Measure to use for resampling. If this is NULL, the Task's default Measure is used. Default is NULL.
perf.resampling	[ResampleDesc or ResampleInstance] Resampling strategy to use. If this is NULL, 2/3 holdout resampling is used. Default is NULL.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
export	[character] Either a character vector indicating the parameters to export as hyperparameters,

or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [`character` | `NULL`]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [`numeric`]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [`character`]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [`character(1)` | `NULL`]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [`logical(1)`]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[`logical(1)`]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[`logical(1)`]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[`logical(1)`]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterVariance()`, `randomForestSRC_filters`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`cpoFilterVariance` *Filter Features: "variance"*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Simple filter based on the variance of the features indepentent of each other. Features with higher variance are considered more important than features with low importance.

Usage

```
cpoFilterVariance(
  perc = NULL,
  abs = NULL,
  threshold = NULL,
  id,
  export = "export.default",
```

```
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

perc
: [numeric(1)]
If set, select perc*100 top scoring features. Mutually exclusive with arguments abs and threshold.

abs
: [numeric(1)]
If set, select abs top scoring features. Mutually exclusive with arguments perc and threshold.

threshold
: [numeric(1)]
If set, select features whose score exceeds threshold. Mutually exclusive with arguments perc and abs.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern
: [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
[logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
[logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
[logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other filter: cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), randomForestSRC_filters

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoFixFactors *Clean Up Factorial Features*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Prevent common pitfalls when using factorial data, by making factorial data have the same levels in training and prediction, and by dropping factor levels that do not occur in training data.

Usage

```
cpoFixFactors(
  drop.unused.levels = TRUE,
  fix.factors.prediction = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`drop.unused.levels`
: Factor levels of data that have no instances in the data are dropped. If "fix.factors.prediction" is false, this can lead to training data having different factor levels than prediction data. Default is TRUE.

`fix.factors.prediction`
: Factor levels are kept the same in training and prediction. This is recommended. Default is TRUE.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

affect.pattern
: [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`cpoIca` *Construct a CPO for ICA Preprocessing*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Use the `fastICA` function implementing the "FastICA algorithm". See the documentation there.

Usage

```
cpoIca(
  n.comp = NULL,
  alg.typ = "parallel",
  fun = "logcosh",
  alpha = 1,
  method = "C",
```

```
  maxit = 200,
  tol = 1e-04,
  verbose = FALSE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`n.comp` [numeric(1) | NULL]
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".

`maxit` [numeric(1)]
Maximum number of iterations. Default is 200.

`tol` [numeric(1)]
Tolerance for convergence, default is `1e-4`.

`verbose` [logical(1)]
Default is `FALSE`.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (ex-

port the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is NULL (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is FALSE.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is FALSE.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is FALSE.

Value

[CPO].

CPOTrained State

The state contains a `$control` slot with the `$K`, `$W` and `$A` slots of the `fastICA` call, as well as a `$center` slot indicating the row-wise center of the training data that will be subtracted before rotation.

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`cpoImpactEncodeClassif` *Impact Encoding*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Impact coding 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.

Usage

```
cpoImpactEncodeClassif(
  smoothing = 1e-04,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
```

```
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`smoothing` [numeric(1)]
A finite positive value used for smoothing. Mostly relevant if a factor does not coincide with a target factor level (and would otherwise give an infinite logit value).

Default is `1e-4`.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
: [logical(1)]
  Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

CPOTrained State

The state's `$control` slot is a list of matrices for each factorial data column. Each of these matrices has rows for each of the data column's levels, and columns for each of the target factor levels, and gives the respective impact values.

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

cpoImpactEncodeRegr *Impact Encoding*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Impact coding 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.

Usage

```
cpoImpactEncodeRegr(
  smoothing = 1e-04,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`smoothing` `[numeric(1)]`
A finite positive value used for smoothing. Default is `1e-4`.

`id` `[character(1)]`
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` `[character]`
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` `[character | NULL]`
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

CPOTrained State

The state's `$control` slot is a list of vectors for each factorial data column. Each of these vectors has an entry for each of the the data column's levels, and gives the respective impact value.

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoImpute *Impute and Re-Impute Data*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

The function impute performs the imputation on a data set and returns, alongside with the imputed data set, an "ImputationDesc" object which can contain "learned" coefficients and helpful data. It can then be passed together with a new data set to reimpute.

The imputation techniques can be specified for certain features or for feature classes, see function arguments.

You can either provide an arbitrary object, use a built-in imputation method listed under imputations or create one yourself using makeImputeMethod.

cpoImpute will impute some columns. cpoImputeAll behaves just like cpoImpute, except that it will throw an error if there are any missings remaining in its output. cpoImputeAll should be used if one wants to prepend an imputer to a learner.

Usage

```
cpoImpute(
  target.cols = character(0),
  classes = list(),
  cols = list(),
  dummy.classes = character(0),
```

```
  dummy.cols = character(0),
  dummy.type = "factor",
  force.dummies = FALSE,
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)

cpoImputeAll(
  target.cols = character(0),
  classes = list(),
  cols = list(),
  dummy.classes = character(0),
  dummy.cols = character(0),
  dummy.type = "factor",
  force.dummies = FALSE,
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`target.cols` [character]
Name of the column(s) specifying the response. Default is `character(0)`.

`classes` [named list]
Named list containing imputation techniques for classes of columns. E.g. `list(numeric = imputeMedian())`.

`cols` [named list]
Named list containing names of imputation methods to impute missing values

in the data column referenced by the list element's name. Overrules imputation set via classes.

dummy.classes
: [character]
Classes of columns to create dummy columns for. Default is character(0).

dummy.cols
: [character]
Column names to create dummy columns (containing binary missing indicator) for. Default is character(0).

dummy.type
: [character(1)]
How dummy columns are encoded. Either as 0/1 with type "numeric" or as "factor". Default is "factor".

force.dummies
: [logical(1)]
Force dummy creation even if the respective data column does not contain any NAs. Note that (a) most learners will complain about constant columns created this way but (b) your feature set might be stochastic if you turn this off. Default is FALSE.

impute.new.levels
: [logical(1)]
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is TRUE.

recode.factor.levels
: [logical(1)]
Recode factor levels after reimputation, so they match the respective element of lvls (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is TRUE.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern	[character(1) | NULL] grep pattern to match feature names by. Default is NULL (no pattern matching)
affect.invert	[logical(1)] Whether to affect all features *not* matched by other affect.* parameters.
affect.pattern.ignore.case	[logical(1)] Ignore case when matching features with affect.pattern; see grep. Default is FALSE.
affect.pattern.perl	[logical(1)] Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.
affect.pattern.fixed	[logical(1)] Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Details

The description object contains these slots

target [character] See argument.

features [character] Feature names (column names of data).,

classes [character] Feature classes (storage type of data).

lvls [named list] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [named list] Mapping of column names to imputation functions.

dummies [named list] Mapping of column names to imputation functions.

impute.new.levels [logical(1)] See argument.

recode.factor.levels [logical(1)] See argument.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

Other imputation CPOs: `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`

`cpoImputeConstant` *Perform Imputation with Constant Value*

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

Usage

```
cpoImputeConstant(
  const,
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
```

```
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`const` [any]
Constant valued use for imputation.

`impute.new.levels` [logical(1)]
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is `TRUE`.

`recode.factor.levels` [logical(1)]
Recode factor levels after reimputation, so they match the respective element of `lvls` (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is `TRUE`.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Details

The description object contains these slots

target [`character`] See argument.

features [`character`] Feature names (column names of `data`).,

classes [`character`] Feature classes (storage type of `data`).

lvls [`named list`] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [`named list`] Mapping of column names to imputation functions.

dummies [`named list`] Mapping of column names to imputation functions.

impute.new.levels [`logical(1)`] See argument.

recode.factor.levels [`logical(1)`] See argument.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: `cpoImpute()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

---

cpoImputeHist *Perform Imputation with Random Values*

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

This imputation method imputes with random values drawn from a distribution that approximates the data distribution as a histogram.

Usage

```
cpoImputeHist(
  breaks = "Sturges",
  use.mids = TRUE,
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
```

```
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`breaks` [`numeric(1)` | "Sturges"]
Number of breaks to use in `hist`. Defaults to auto-detection via "Sturges".

`use.mids` [`logical(1)`]
If `x` is numeric and a histogram is used, impute with bin mids (default) or instead draw uniformly distributed samples within bin range.

`impute.new.levels` [`logical(1)`]
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is `TRUE`.

`recode.factor.levels` [`logical(1)`]
Recode factor levels after reimputation, so they match the respective element of `lvls` (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is `TRUE`.

`id` [`character(1)`]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [`character`]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [`character` | `NULL`]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [`numeric`]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

affect.names [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
[logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
[logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
[logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Details

The description object contains these slots

target [character] See argument.

features [character] Feature names (column names of data).,

classes [character] Feature classes (storage type of data).

lvls [named list] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [named list] Mapping of column names to imputation functions.

dummies [named list] Mapping of column names to imputation functions.

impute.new.levels [logical(1)] See argument.

recode.factor.levels [logical(1)] See argument.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

---

cpoImputeLearner — *Perform Imputation with an* `mlr` `Learner`

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

Usage

```
cpoImputeLearner(
  learner,
  features = NULL,
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
```

```
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`learner`
: [Learner | character(1)]
Supervised learner. Its predictions will be used for imputations. If you pass a string the learner will be created via makeLearner. Note that the target column is not available for this operation.

`features`
: [character]
Features to use in `learner` for prediction. Default is `NULL` which uses all available features except the target column of the original task.

`impute.new.levels`
: [logical(1)]
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is `TRUE`.

`recode.factor.levels`
: [logical(1)]
Recode factor levels after reimputation, so they match the respective element of `lvls` (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is `TRUE`.

`id`
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export`
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type`
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index`
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

affect.names [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

affect.pattern [character(1) | NULL]
grep pattern to match feature names by. Default is `NULL` (no pattern matching)

affect.invert [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

affect.pattern.ignore.case
[logical(1)]
Ignore case when matching features with `affect.pattern`; see grep. Default is `FALSE`.

affect.pattern.perl
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see grep. Default is `FALSE`.

affect.pattern.fixed
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see grep. Default is `FALSE`.

Details

The description object contains these slots

target [character] See argument.

features [character] Feature names (column names of `data`).,

classes [character] Feature classes (storage type of `data`).

lvls [named list] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [named list] Mapping of column names to imputation functions.

dummies [named list] Mapping of column names to imputation functions.

impute.new.levels [logical(1)] See argument.

recode.factor.levels [logical(1)] See argument.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

---

cpoImputeMax *Perform Imputation with Multiple of Minimum*

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

This method imputes by the maximum value of each column, multiplied by a constant.

Usage

```
cpoImputeMax(
  multiplier = 1,
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
```

```
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`multiplier` [numeric(1)]
Value that stored minimum or maximum is multiplied with when imputation is done.

`impute.new.levels` [logical(1)]
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is `TRUE`.

`recode.factor.levels` [logical(1)]
Recode factor levels after reimputation, so they match the respective element of `lvls` (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is `TRUE`.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

affect.pattern [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
[logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
[logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
[logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Details

The description object contains these slots

target [character] See argument.

features [character] Feature names (column names of data).,

classes [character] Feature classes (storage type of data).

lvls [named list] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [named list] Mapping of column names to imputation functions.

dummies [named list] Mapping of column names to imputation functions.

impute.new.levels [logical(1)] See argument.

recode.factor.levels [logical(1)] See argument.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

---

cpoImputeMean — *Perform Imputation with Mean Value*

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

Usage

```
cpoImputeMean(
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
```

```
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`impute.new.levels`
: [logical(1)]
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is `TRUE`.

`recode.factor.levels`
: [logical(1)]
Recode factor levels after reimputation, so they match the respective element of `lvls` (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is `TRUE`.

`id`
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export`
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type`
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index`
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names`
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern`
: [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert`
: [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Details

The description object contains these slots

target [character] See argument.

features [character] Feature names (column names of data).,

classes [character] Feature classes (storage type of data).

lvls [named list] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [named list] Mapping of column names to imputation functions.

dummies [named list] Mapping of column names to imputation functions.

impute.new.levels [logical(1)] See argument.

recode.factor.levels [logical(1)] See argument.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform()

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoImputeMedian *Perform Imputation with Median Value*

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

Usage

```
cpoImputeMedian(
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`impute.new.levels`
: `[logical(1)]`
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is `TRUE`.

`recode.factor.levels`
: `[logical(1)]`
Recode factor levels after reimputation, so they match the respective element of `lvls` (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is `TRUE`.

`id`
: `[character(1)]`
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export`
: `[character]`
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type`
: `[character | NULL]`
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index`
: `[numeric]`
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names`
: `[character]`
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern`
: `[character(1) | NULL]`
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert`
: `[logical(1)]`
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
: `[logical(1)]`
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
: `[logical(1)]`
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Details

The description object contains these slots

target [character] See argument.

features [character] Feature names (column names of data).,

classes [character] Feature classes (storage type of data).

lvls [named list] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [named list] Mapping of column names to imputation functions.

dummies [named list] Mapping of column names to imputation functions.

impute.new.levels [logical(1)] See argument.

recode.factor.levels [logical(1)] See argument.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform()

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoImputeMin *Perform Imputation with Multiple of Minimum*

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

This method imputes by the minimum value of each column, multiplied by a constant.

Usage

```
cpoImputeMin(
  multiplier = 1,
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

multiplier	[numeric(1)] Value that stored minimum or maximum is multiplied with when imputation is done.

impute.new.levels
: [logical(1)]
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is `TRUE`.

recode.factor.levels
: [logical(1)]
Recode factor levels after reimputation, so they match the respective element of `lvls` (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is `TRUE`.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

affect.pattern
: [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.fixed
: [logical(1)]

Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Details

The description object contains these slots

target [character] See argument.

features [character] Feature names (column names of data).,

classes [character] Feature classes (storage type of data).

lvls [named list] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [named list] Mapping of column names to imputation functions.

dummies [named list] Mapping of column names to imputation functions.

impute.new.levels [logical(1)] See argument.

recode.factor.levels [logical(1)] See argument.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform()

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoImputeMode *Perform Imputation with Mode Value*

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

Usage

```
cpoImputeMode(
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`impute.new.levels`
: [logical(1)]
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is TRUE.

`recode.factor.levels`
: [logical(1)]
Recode factor levels after reimputation, so they match the respective element

of lvls (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is TRUE.

id — [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export — [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type — [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index — [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names — [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern — [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert — [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case — [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl — [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed — [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Details

The description object contains these slots

target [character] See argument.

features [`character`] Feature names (column names of `data`).,

classes [`character`] Feature classes (storage type of `data`).

lvls [`named list`] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [`named list`] Mapping of column names to imputation functions.

dummies [`named list`] Mapping of column names to imputation functions.

impute.new.levels [`logical(1)`] See argument.

recode.factor.levels [`logical(1)`] See argument.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeNormal()`, `cpoImputeUniform()`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoImputeNormal *Perform Imputation with Normally Distributed Random Values*

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

Usage

```
cpoImputeNormal(
  mu = NA_real_,
  sd = NA_real_,
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`mu` [numeric(1)]
Mean of normal distribution. If missing it will be estimated from the data.

`sd` [numeric(1)]
Standard deviation of normal distribution. If missing it will be estimated from the data.

`impute.new.levels` [logical(1)]
If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is `TRUE`.

`recode.factor.levels` [logical(1)]
Recode factor levels after reimputation, so they match the respective element of `lvls` (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is `TRUE`.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern
: [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Details

The description object contains these slots

target [character] See argument.

features [character] Feature names (column names of data).,

classes [character] Feature classes (storage type of data).

lvls [named list] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [named list] Mapping of column names to imputation functions.

dummies [named list] Mapping of column names to imputation functions.

impute.new.levels [logical(1)] See argument.

recode.factor.levels [logical(1)] See argument.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeUniform()

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

`cpoImputeUniform` *Perform Imputation with Uniformly Random Values*

Description

Allows imputation of missing feature values through various techniques. Note that you have the possibility to re-impute a data set in the same way as the imputation was performed during training. This especially comes in handy during resampling when one wants to perform the same imputation on the test set as on the training set.

Usage

```
cpoImputeUniform(
  min = NA_real_,
  max = NA_real_,
  impute.new.levels = TRUE,
  recode.factor.levels = TRUE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`min`
: [numeric(1)]
  Lower bound for uniform distribution. If NA (default), it will be estimated from the data.

`max`
: [numeric(1)]
  Upper bound for uniform distribution. If NA (default), it will be estimated from the data.

`impute.new.levels`
: [logical(1)]
  If new, unencountered factor level occur during reimputation, should these be handled as NAs and then be imputed the same way? Default is `TRUE`.

`recode.factor.levels`
: [logical(1)]
  Recode factor levels after reimputation, so they match the respective element of `lvls` (in the description object) and therefore match the levels of the feature factor in the training data after imputation?. Default is `TRUE`.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Details

The description object contains these slots

target [character] See argument.

features [character] Feature names (column names of `data`).,

classes [`character`] Feature classes (storage type of `data`).

lvls [`named list`] Mapping of column names of factor features to their levels, including newly created ones during imputation.

impute [`named list`] Mapping of column names to imputation functions.

dummies [`named list`] Mapping of column names to imputation functions.

impute.new.levels [`logical(1)`] See argument.

recode.factor.levels [`logical(1)`] See argument.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other imputation CPOs: `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

CPOLearner *CPO Learner Object*

Description

CPO Learners are created when a CPO gets attached to an mlr-Learner object. The resulting learner performs the operation described by the attached CPO before fitting the model specified by the Learner. It is possible to attach compound CPOs, and it is possible to attach more CPOs to a learner that is already a CPOLearner. If the attached CPO exports hyperparameters, these become part of the newly created learner and can be queried and set using functions such as getParamSet, getHyperPars, and setHyperPars.

The model created when training a CPOLearner also contains the relevant CPORetrafo information to be applied to prediction data; this can be retrieved using retrafo. The CPOInverter functionality is handled equally transparently by the model.

A CPOLearner can possibly have different LearnerProperties than the base Learner to which it is attached. This depends on the CPO's properties, see CPOProperties.

It is possible to retrieve the CPOLearner's base learner using getLearnerBare, and to get the attached CPOs using getLearnerCPO.

See Also

Other CPO lifecycle related: CPO, CPOConstructor, CPOTrained, NULLCPO, %>>%(), attachCPO(), composeCPO(), getCPOClass(), getCPOConstructor(), getCPOTrainedCPO(), identicalCPO(), makeCPO()

Other CPOLearner related: attachCPO(), getLearnerBare(), getLearnerCPO()

Examples

```
lrn = makeLearner("classif.logreg")
cpolrn = cpoScale() %>>% lrn
print(cpolrn)

getLearnerBare(cpolrn)  # classif.logreg Learner
getLearnerCPO(cpolrn)  # cpoScale() CPO

getParamSet(cpolrn)  # includes cpoScale hyperparameters

model = train(cpolrn, pid.task)  # behaves like a learner
retrafo(model)  # the CPORetrafo that was trained

predict(model, pid.task)  # otherwise behaves like an mlr model
```

cpoLogTrafoRegr *Log-Transform a Regression Target Variable.*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Log-transforms the regression Task's target variable.

If predict.type is "response" for inversion, the model's prediction is exponentiated.

If predict.type = "se" prediction is performed, the model's prediction is taken as the parameters of a lognormal random variable; the inverted prediction is then mean = exp(mean + se^2 / 2), se = sqrt((exp(se^2) - 1) * exp(2 * mean + se^2)).

It is therefore recommended to use "se" prediction, possibly with the help of cpoResponseFromSE.

Usage

```
cpoLogTrafoRegr(id)
```

Arguments

id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoMakeCols *Create Columns from Expressions*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Create columns from expressions and the incoming data.

When cpoMakeCols or cpoAddCols are called as cpoMakeCols(<newcolname> = <expression>, ...), a new column with the name <newcolname containing the result of <expression> is created. The expressions need to be vectorising R expressions and may refer to any feature columns in the data (excluding the target) and any other values. The names should be valid data.frame column names and may not clash with the target column name.

cpoMakeCols replaces existing cols by the newly created ones, cpoAddCols adds them to the data already present.

Usage

```
cpoMakeCols(..., .make.factors = TRUE)

cpoAddCols(..., .make.factors = TRUE)
```

Arguments

...	[any] Expressions of the form colname = expr. See Examples.
.make.factors	[logical(1)] Whether to turn resulting logical and character columns into factor columns (which are preferred by mlr). Default is TRUE.

Value

[CPO].

CPOTrained State

The created state is empty.

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

Examples

```
res = pid.task %>>% cpoAddCols(gpi = glucose * pressure * insulin, pm = pregnant * mass)
head(getTaskData(res))
```

cpoMissingIndicators *Convert Data into Factors Indicating Missing Data*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Convert a data.frame into a data.frame with the same column names, but with columns of factors indicating whether data was missing or not.

This is most useful in combination with cpoCbind.

Usage

```
cpoMissingIndicators(
  force.dummies = FALSE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`force.dummies` [logical(1)]
Whether to create dummy columns even for data that is not missing. This can be useful if missing data is expected during test in columns where it did not occur during training.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern
: [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoModelMatrix — *Create a “Model Matrix” from the Data Given a Formula*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

This uses the “stats” function model.matrix to create (numerical) data from the given data, using the provided formula.

Usage

```
cpoModelMatrix(
  formula,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

formula	[formula] Formula to use. Higher order interactions can be created using constructs like `~. ^ 2`.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoOversample — *Over- or Undersample Binary Classification Tasks*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Oversamples the minor or undersamples the major class in a binary classification task to alleviate class imbalance. Uses mlr::oversample and mlr::undersample, see documentation there.

Usage

```
cpoOversample(rate = NULL, cl = NULL, id, export = "export.default")

cpoUndersample(rate = NULL, cl = NULL, id, export = "export.default")
```

Arguments

rate
: [numeric(1) | NULL]
Factor to up- or downsample a class. Must be between 0 and 1 for undersampling and greater or equal 1 for oversampling. If this is NULL, this is the ratio of major to minor class prevalence (for oversampling, or the inverse for undersampling). Must not be NULL if cl is not NULL and not the minor class for oversampling / the major class for undersampling. Default is NULL.

cl
: [character(1) | NULL]
Class to over- or undersample. For NULL, the minor class for oversampling or the major class for undersampling is chosen automatically.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoPca — *Construct a CPO for PCA Preprocessing*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Performs principal component analysis using prcomp.

Usage

```
cpoPca(
  center = TRUE,
  scale = FALSE,
  tol = NULL,
  rank = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

- `center` — [logical(1)]
  Whether to center columns before PCA. Default is TRUE.
- `scale` — [logical(1)]
  Whether to scale columns to unit variance before PCA. Default is FALSE.
- `tol` — [numeric(1) | NULL]
  Magnitude below which components are omitted. Default is NULL: all columns returned. Sensible settings are tol = 0, tol = sqrt(.Machine$double.eps).
- `rank` — [numeric(1) | NULL]
  Maximal number of components to return. Default is NULL, no limit.
- `id` — [character(1)]
  id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
- `export` — [character]
  Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".
- `affect.type` — [character | NULL]
  Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.
- `affect.index` — [numeric]
  Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).
- `affect.names` — [character]
  Feature names of feature columns to affect. The order of names given is respected. Default is character(0).
- `affect.pattern` — [character(1) | NULL]
  grep pattern to match feature names by. Default is NULL (no pattern matching)
- `affect.invert` — [logical(1)]
  Whether to affect all features *not* matched by other affect.* parameters.
- `affect.pattern.ignore.case` — [logical(1)]
  Ignore case when matching features with affect.pattern; see grep. Default is FALSE.
- `affect.pattern.perl` — [logical(1)]
  Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

CPOTrained State

The state's $control slot is a list with the $rotation matrix, the $scale vector and the $center vector as returned by prcomp.

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

cpoProbEncode *Probability Encoding*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Converts factor columns into columns giving the probability for each target class to have this target, given the column value.

Usage

```
cpoProbEncode(
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

affect.names [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
[logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
[logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
[logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

CPOTrained State

The state's $control slot is a list of matrices for each factorial data column. Each of these matrices has rows for each of the data column's levels, and columns for each of the target factor levels, and gives the empirical marginal conditional probabilities for each target value given the column value.

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoQuantileBinNumerics

Split Numeric Features into Quantile Bins

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Usage

```
cpoQuantileBinNumerics(
  numsplits = 2,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

numsplits	[numeric(1)] Number of bins to create. Default is 2.

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`cpoRegrResiduals` *Train a Model on a Task and Return the Residual Task*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Given a regression learner, this `CPO` fits the learner to a regression `Task` and replaces the regression target with the residuals–the differences of the target values and the model’s predictions–of the model.

For inversion, the predictions of the model for the prediction data are added to the predictions to be inverted.

If `predict.se` is `TRUE`, `property.type == "se"` inversion can also be performed. In that case, the `se` of the incoming prediction and the `se` of the internal model are assumed to be independently distributed, and the resulting `se` is the pythagorean sum of the `se`s.

Usage

```
cpoRegrResiduals(
  learner,
  predict.se = FALSE,
  crr.train.residuals = "plain",
  crr.resampling = cv5,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`learner`
: [`character(1)` | `Learner`]
A regression `Learner`, or a `character(1)` identifying a `Learner` to be constructed.

`predict.se`
: [`logical(1)`]
Whether to fit the model with "se" predict type. This enables the resulting `CPOInverter` to be used for `property.type == "se"` inversion. Default is `FALSE`.

`crr.train.residuals`
: [`character(1)`]
What residuals to use for training (i.e. initial transformation). One of "resample", "oob", "plain". If "resample" is given, the out-of-resampling-fold predictions are used when resampling according to the `resampling` parameter. If "oob" is used, the `Learner` must have the "oobpreds" property; the out-of-bag predictions are then used. If `train.residuals` is "plain", the simple regression residuals are used. "plain" may offer slightly worse performance than the alternatives, but few `mlr` `Learners` support "oobpreds", and "resample" can come at a considerable run time penalty. Default is "plain".

`crr.resampling`
: [`ResampleDesc` | `ResampleInstance`]
What resampling to use when `train.residuals` is "resample"; otherwise has no effect. The `$predict` slot of the resample description will be ignored and set to `test`. If a data point is predicted by multiple resampling folds, the average residual is used. If a data point is not predicted by any resampling fold, the "plain" residual is used for that one. Default is `cv5`.

`id`
: [`character(1)`]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

CPOTrained State

The `CPORetrafo` state's `$control` slot is the `WrappedModel` created when training the `learner` on the given data.

The `CPOInverter` state's `$control` slot is a `data.frame` of the "response" and (if `predict.se` is `TRUE`) "se" columns of the prediction done by the model on the data.

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoResponseFromSE — *Use the "se"* predict.type *for "response" Prediction*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Some Learners may give better "response" prediction if their "se" predict.type is used, especially when a cpoApplyFunRegrTargetis used on it. This CPO performs no transformation of the data, but instructs the underlying Learner to do "se" prediction when "response" prediction is requested (the default) and drops the se column.

Usage

```
cpoResponseFromSE(
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`id` [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

cpoSample *Sample Data from a Task*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Takes samples from a task to decrease (or possibly increase) its size. This can be used to reduce training time, or to implement bootstrapping.

Usage

```
cpoSample(
  rate = NULL,
  size = NULL,
  replace = FALSE,
  id,
  export = "export.default"
)
```

Arguments

rate	[numeric(1) | NULL] How many samples to take, relative to the task size. Default is NULL: Not using relative sampling rate. Exactly one of this or size must be non-NULL.
size	[integer(1) | NULL] How many samples to take. Default is NULL: Not using absolute size. Exactly one of this or size must be non-NULL.
replace	[logical(1)] Whether to sample with replacement. Default is FALSE.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
export	[character] Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frames`, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`, `makeCPOMultiplex()`

`cpoScale` *Construct a CPO for Scaling / Centering*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Usage

```
cpoScale(
  center = TRUE,
  scale = TRUE,
```

```
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

center	[logical(1)] Whether to center the data. Default is TRUE.
scale	[logical(1)] Whether to scale the data. Default is TRUE.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
export	[character] Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".
affect.type	[character | NULL] Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.
affect.index	[numeric] Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).
affect.names	[character] Feature names of feature columns to affect. The order of names given is respected. Default is character(0).
affect.pattern	[character(1) | NULL] grep pattern to match feature names by. Default is NULL (no pattern matching)
affect.invert	[logical(1)] Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoScaleMaxAbs *Max Abs Scaling CPO*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Scale 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.

Usage

```
cpoScaleMaxAbs(
  maxabs = 1,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`maxabs`	[`numeric(1)`] The maximum absolute value for each column after transformation. Default is 1.
`id`	[`character(1)`] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
`export`	[`character`] Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [character | NULL]
Type of columns to affect. A subset of “numeric”, “factor”, “ordered”, “other”, or NULL to not match by column type. Default is NULL.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

`affect.pattern` [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the “id” parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoScaleRange — *Range Scaling CPO*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Linearly transform data columns so they are between lower and upper. If lower is greater than upper, this will reverse the ordering of input data. NA, Inf are ignored.

Usage

```
cpoScaleRange(
  lower = 0,
  upper = 1,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`lower`
: [numeric(1)]
  Target value of smallest item of input data. Default is 0.

`upper`
: [numeric(1)]
  Target value of greatest item of input data. Default is 1.

`id`
: [character(1)]
  id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export`
: [character]
  Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type`
: [character | NULL]
  Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index`
: [numeric]
  Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names`
: [character]
  Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern`
: [character(1) | NULL]
  `grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert`
: [logical(1)]
  Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
: [logical(1)]
  Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
: [logical(1)]
  Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
: [logical(1)]
  Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoSelect — *Drop All Columns Except Certain Selected Ones from Data*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Select columns by type or name. The parameters "type" and "pattern" are additive; if both are given, all column that match either will be returned.

cpoSelectFreeProperties behaves just as cpoSelect, with the additional function that it is treated like a CPO that removes all data properties from the data. This disables the internal property check and can be useful when trying to compose CPOs that do not have compatible properties.

Usage

```
cpoSelect(
  type = character(0),
  index = integer(0),
  names = character(0),
  pattern = NULL,
  pattern.ignore.case = FALSE,
  pattern.perl = FALSE,
  pattern.fixed = FALSE,
  invert = FALSE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)

cpoSelectFreeProperties(
  type = character(0),
  index = integer(0),
  names = character(0),
  pattern = NULL,
  pattern.ignore.case = FALSE,
  pattern.perl = FALSE,
  pattern.fixed = FALSE,
  invert = FALSE,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`type` — [character]
One or more out of "numeric", "ordered", "factor", "other". The type of columns to keep. Default is `character(0)`.

`index` — [integer]
Indices of columns to keep. Note that the index counts columns without the target column(s). This and the next parameter make it possible to re-order columns. While all columns which match either "type", "pattern" or "index" remain in the resulting data, the ones selected by "index" are put at the front in the order specified. Default is `integer(0)`.

`names` — [character]
Names of columns to keep. Matching columns will be kept in order of their names occurring, but after the columns indicated in "index".

`pattern` — [character(1)]
A pattern to match against the column names. Same as in `grep`. Default is `NULL` for no matching.

`pattern.ignore.case` — [logical(1)]
Influences behaviour of "pattern": Whether to perform case insensitive matching. Same as in `grep`. Default is `FALSE`.

`pattern.perl` — [logical(1)]
Influences behaviour of "pattern": Should Perl-compatible regexps be used? Same as in `grep`. Default is `FALSE`.

`pattern.fixed` — [logical(1)]
Influences behaviour of "pattern": Whether to use match `pattern` as as is. Same as in `grep`. Default is `FALSE`.

`invert` — [logical(1)]
Invert column selection: Drop the named columns and return the rest, instead of keeping the selected columns only. Default is `FALSE`.

`id` — [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export` — [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` — [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [character]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [character(1) | NULL]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [logical(1)]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[logical(1)]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[logical(1)]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoSmote *Perform SMOTE Oversampling for Binary Classification*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Uses mlr's smote function to perform "Synthetic Minority Oversampling TEchnique" sample generation to handle class imbalance in binary tasks.

See the smote documentation for details.

Usage

```
cpoSmote(
  rate = NULL,
  nn = 5,
  standardize = TRUE,
  alt.logic = FALSE,
  id,
  export = "export.default"
)
```

Arguments

rate	[numeric(1) | NULL] Upsampling factor, between 1 and Inf. Default is NULL, which sets this to the ratio <majority prevalence> / <minority prevalence>
nn	[integer(1)] Number of nearest neighbours to consider. Defaults to 5.

standardize
: [integer(1)]
Standardize feature values. Default is TRUE.

alt.logic
: [integer(1)]
Use alternative logic for minority selection. Default is FALSE.

id
: [character(1)]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export
: [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSpatialSign(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

---

cpoSpatialSign *Scale Rows to Unit Length*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

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

Usage

```
cpoSpatialSign(
  length = 1,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

length	[numeric(1)] Length to scale rows to. Default is 1.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
export	[character] Either a character vector indicating the parameters to export as hyperparameters, or one of the special values “export.all” (export all parameters), “export.default”

(export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type` [`character` | `NULL`]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index` [`numeric`]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names` [`character`]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern` [`character(1)` | `NULL`]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert` [`logical(1)`]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
[`logical(1)`]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
[`logical(1)`]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
[`logical(1)`]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional “special” optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoTransformParams(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

CPOTrained — *Get the Retransformation or Inversion Function from a Resulting Object*

Description

When applying a CPO to a data.frame or Task, the data is not only changed, additionally a retransformation and an inversion object is created that can be applied to other data of the same kind. This is useful if new data (for prediction or validation) is to be handled in the same machine learning procedure.

For example, when performing PCA on training data using cpoPca, the rotation matrix is saved and can be used on new (prediction) data. As another example, consider a log-transformation of the target column in a regression problem. When predictions are made with new data, it may be useful to invert the transformation on the predicted values by exponentiating them.

The information created when a CPO is applied is saved in a CPORetrafo object, and a CPOInverter object, which are both saved as *attributes*. The retrafo and inverter function retrieve these objects. It is furthermore possible to set these attributes using the retrafo<- and inverter<- functions, using constructs like retrafo(data) <- retr.obj. The retrafo or inverter attributes can be reset individually by setting them to NULL: retrafo(data) <- NULL, or by using the clearRI function.

When chaining %>>% on a data object, the retrafo and inverter associated with the result is also chained automatically. Beware, however, that this just accesses the retrafo attribute internally. Therefore, if you plan to do apply multiple transformations with other operations in between, make sure to reset the retrafo function by setting it to NULL, or using the clearRI function. See examples.

Usage

```
retrafo(data)

inverter(data)

retrafo(data) <- value

inverter(data) <- value
```

Arguments

data	[data.frame | Task | WrappedModel] The result of a CPO applied to a data set.
value	[CPOTrained | NULL] The retrafo or inverter to set. This must either be a CPORetrafo for retrafo<- or a CPOInverter for inverter<-, or NULL to reset the retrafo or inverter attributes.

Value

[CPOTrained]. The retransformation function that can be applied to new data. This is a CPORetrafo object for retrafo or a CPOInverter object for inverter.

CPORetrafo and CPOInverter

CPORetrafo and CPOInverter objects are members of the CPOTrained class, which can be handled similarly to CPO objects: Their hyperparameters can be inspected using getParamSet and link[mlr]{getHyperPars}, print.CPOTrained is used for (possibly verbose) printing. To apply the retrafo or inverter transformation represented by the object to data, use the applyCPO or %>>% function.

CPOTrained objects can be chained using %>>% or pipeCPO, and broken into primitives using as.list.CPOTrained. However, since the CPOTrained objects represent transformations that relate closely to the data used to train it (and therefore to the position within a CPO pipeline), it is only advisable to chain or break apart CPOTrained pipes for inspection, or if you really know what you are doing.

(Primitive) CPORetrafo objects can be inspected using getCPOTrainedState, and it is possible to create new CPORetrafo objects from (possibly modified) retrafo state using makeCPOTrainedFromState.

Difference between CPORetrafo and CPOInverter

The fundamental difference between CPORetrafo and CPOInverter is that a CPORetrafo is created only when a CPO is applied to a data set, and is used to perform the same transformation on new (prediction) data. The CPOInverter is created whenever a CPO *or* CPORetrafo is applied to data (whether training or prediction data). It is in fact used to invert the transformation done to the target column of a Task. Since this operation may depend on the new prediction data, and not only on the training data fed to the CPO when the CPORetrafo was created, the CPOInverter object is more closely bound to the particular data set used to create it.

In some cases a target transformation is independent of the data used to create it (e.g. log-transform of a regression target column); in that case the `CPORetrafo` can be used with `invert`. This is the concept of `CPOTrainedCapability`, which can be queried using `getCPOTrainedCapability`.

Using CPORetrafo

`CPORetrafo` objects can be applied to new data sets using the `%>>%` operator, the `applyCPO` generic, or the `predict` generic, all of which perform the same action.

Using CPOInverter

To use a `CPOInverter`, use the `invert` function.

See Also

`clearRI` about the problem of needing to reset `retrafo` and `inverter` attributes sometimes.

Other CPO lifecycle related: `CPO`, `CPOConstructor`, `CPOLearner`, `NULLCPO`, `%>>%()`, `attachCPO()`, `composeCPO()`, `getCPOClass()`, `getCPOConstructor()`, `getCPOTrainedCPO()`, `identicalCPO()`, `makeCPO()`

Other retrafo related: `NULLCPO`, `%>>%()`, `applyCPO()`, `as.list.CPO`, `clearRI()`, `getCPOClass()`, `getCPOName()`, `getCPOOperatingType()`, `getCPOPredictType()`, `getCPOProperties()`, `getCPOTrainedCPO()`, `getCPOTrainedCapability()`, `getCPOTrainedState()`, `is.retrafo()`, `makeCPOTrainedFromState()`, `pipeCPO()`, `print.CPOConstructor()`

Other inverter related: `NULLCPO`, `%>>%()`, `applyCPO()`, `as.list.CPO`, `clearRI()`, `getCPOClass()`, `getCPOName()`, `getCPOOperatingType()`, `getCPOPredictType()`, `getCPOProperties()`, `getCPOTrainedCPO()`, `getCPOTrainedCapability()`, `getCPOTrainedState()`, `is.inverter()`, `makeCPOTrainedFromState()`, `pipeCPO()`, `print.CPOConstructor()`

Examples

```
traindat = subsetTask(pid.task, 1:400)
preddat = subsetTask(pid.task, 401:768)

trained = traindat %>>% cpoPca()
reFun = retrafo(trained)
predicted = preddat %>>% reFun
head(getTaskData(predicted))

# chaining works
trained = traindat %>>% cpoPca() %>>% cpoScale()
reFun = retrafo(trained)
predicted = preddat %>>% reFun
head(getTaskData(predicted))

# reset the retrafo when doing other steps!

trained.tmp = traindat %>>% cpoPca()
reFun1 = retrafo(trained.tmp)

imp = impute(trained.tmp)
```

```
trained.tmp = imp$task  # nonsensical example
retrafo(trained.tmp) = NULL  # NECESSARY HERE

trained = trained.tmp %>>% cpoScale()

reFun2 = retrafo(trained)
predicted = getTaskData(reimpute(preddat %>>% reFun1, imp$desc),
  target.extra = TRUE)$data %>>% reFun2
```

---

cpoTransformParams *Transform CPO Hyperparameters*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

Transforms hyperparameters, or establishes dependencies between them. The `CPO` given to `cpoTransformParams` gets wrapped inside a new `CPO` with different hyperparameters. The parameters for which a transformation is given are not exported (unless also given in `additional.parameters`).

Usage

```
cpoTransformParams(
  cpo = NULLCPO,
  transformations = list(),
  additional.parameters = makeParamSet(),
  par.vals = list()
)
```

Arguments

`cpo` [`CPO`]
The CPO to use. Currently this may only have a single OperatingType. Default is `NULLCPO`.

`transformations` [named `list` of `language`]
This list is contains `expression`s or `quote`s that are evaluated in the context of the *externally given* hyperparameters and then give the values of the internal hyperparameters. The name of each list element determines to what hyperparameter of `cpo` the result of the expression is written.

Expressions can not depend on the results of other expressions.

Hyperparameters of `cpo` named in this list are not exported by the Transform-Params CPO. It is, however, possible to create synonymous parameters in `additional.parameters`.

Default is `list()`.

additional.parameters
: [ParamSet]
Additional parameters to create, on which expressions in transformations may depend. They may contain the same names as transformations, but may not have names of hyperparameters of cpo that are *not* in transformations.

par.vals
: [list]
Optional default values of parameters in additional.parameters. These override the ParamSet's default values. Default is list(). These must only concern parameters in additional.parameters, not the ones in cpo.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

Other special CPOs: cpoCbind(), cpoWrap(), makeCPOCase(), makeCPOMultiplex()

cpoWrap	*CPO Wrapper*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

Applies the CPO that is given to the CPO hyperparameter.

cpoWrap only wraps Feature Operation CPOs, cpoWrapRetrafoless only wraps Retrafoless CPOs.

Target Operation CPOs currently cannot be wrapped, sorry.

Usage

```
cpoWrap(
  cpo,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)

cpoWrapRetrafoless(cpo, id, export = "export.default")
```

Arguments

cpo	[CPO] The CPO to wrap.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.
export	[character] Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type
: [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index
: [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names
: [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern
: [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert
: [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
: [logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
: [logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
: [logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a CPOConstructor

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the id parameter, and the affect.* parameters. The affect.* parameters enable the user to control which subset of a given dataset is affected. If no affect.* parameters are given, all data features are affected by default.

See Also

Other CPOs: cpoApplyFun(), cpoApplyFunRegrTarget(), cpoAsNumeric(), cpoCache(), cpoCbind(), cpoCollapseFact(), cpoDropConstants(), cpoDropMostlyConstants(), cpoDummyEncode(), cpoFilterAnova(), cpoFilterCarscore(), cpoFilterChiSquared(), cpoFilterFeatures(), cpoFilterGainRatio(), cpoFilterInformationGain(), cpoFilterKruskal(), cpoFilterLinearCorrelation(), cpoFilterMrmr(), cpoFilterOneR(), cpoFilterPermutationImportance(), cpoFilterRankCorrelation(), cpoFilterRelief(), cpoFilterRfCImportance(), cpoFilterRfImportance(), cpoFilterRfSRCImportance(), cpoFilterSymmetricalUncertainty(), cpoFilterUnivariate(), cpoFilterVariance(), cpoFixFactors(), cpoIca(), cpoImpactEncodeClassif(), cpoImpactEncodeRegr(), cpoImpute(), cpoImputeConstant(), cpoImputeHist(), cpoImputeLearner(), cpoImputeMax(), cpoImputeMean(), cpoImputeMedian(), cpoImputeMin(), cpoImputeMode(), cpoImputeNormal(), cpoImputeUniform(), cpoLogTrafoRegr(), cpoMakeCols(), cpoMissingIndicators(), cpoModelMatrix(), cpoOversample(), cpoPca(), cpoProbEncode(), cpoQuantileBinNumerics(), cpoRegrResiduals(), cpoResponseFromSE(), cpoSample(), cpoScale(), cpoScaleMaxAbs(), cpoScaleRange(), cpoSelect(), cpoSmote(), cpoSpatialSign(), cpoTransformParams(), makeCPOCase(), makeCPOMultiplex()

Other special CPOs: cpoCbind(), cpoTransformParams(), makeCPOCase(), makeCPOMultiplex()

---

discrete *defined to avoid problems with the static type checker*

Description

defined to avoid problems with the static type checker

Usage

```
discrete()
```

---

funct *defined to avoid problems with the static type checker*

Description

defined to avoid problems with the static type checker

Usage

```
funct()
```

getCPOAffect *Get the Selection Arguments for Affected CPOs*

Description

Get the affect.* arguments from when the CPO was constructed. These are in one-to-one correspondence to the affect.* parameters given to the CPOConstructor, see the parameter documentation there.

Usage

```
getCPOAffect(cpo, drop.defaults = TRUE)
```

Arguments

cpo	[CPO] The cpo.
drop.defaults	[logical(1)] Whether to only return the arguments that deviate from the default. Default is TRUE.

Value

[list]. A named list of the affect.* arguments given to the CPOConstructor. The names are stripped of the "affect."-prefix.

See Also

Other getters and setters: CPO, getCPOClass(), getCPOConstructor(), getCPOId(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), setCPOId()

getCPOClass *Get the CPO Class*

Description

Gets the relevant CPO class that distinguishes between steps in a CPO's lifecycle.

There is a fundamental distinction between CPO objects and CPOTrained objects, the latter of which can provide either retrafo or inverter functionality, or both. CPOTrained are subclassed into CPOInverter (only inverter functionality), or CPORetrafo (retrafo, possibly also inverter). To get more information about a CPORetrafo object's capabilities, use getCPOTrainedCapability.

Usage

```
getCPOClass(cpo)
```

Arguments

cpo [CPOConstructor | CPO | CPOTrained]
The CPO.

Value

[character(1)]. "CPOConstructor" if the given object is a CPOConstructor, "CPO" for a CPO, "CPOInverter" for a CPOInverter only, "CPORetrafo" for a CPORetrafo object (which may have inverter capabilities, see link{getCPOTrainedCapability}), "NULLCPO" for a NULLCPO.

See Also

Other getters and setters: CPO, getCPOAffect(), getCPOConstructor(), getCPOId(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), setCPOId()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other CPOConstructor related: CPOConstructor, getCPOConstructor(), getCPOName(), identicalCPO(), makeCPO(), print.CPOConstructor()

Other CPO classifications: CPO, getCPOOperatingType(), getCPOTrainedCapability()

Other CPO lifecycle related: CPO, CPOConstructor, CPOLearner, CPOTrained, NULLCPO, %>>%(), attachCPO(), composeCPO(), getCPOConstructor(), getCPOTrainedCPO(), identicalCPO(), makeCPO()

---

getCPOConstructor *Get the CPOConstructor Used to Create a CPO Object*

Description

Get the CPOConstructor used to create a CPO or CPOTrained object. Only primitive CPO or CPOTrained objects have an originating CPOConstructor.

Usage

```
getCPOConstructor(cpo)
```

Arguments

cpo [CPO | CPOTrained]
The CPO, Retrafo, or Inverter to get the original CPOConstructor from.

Value

[CPOConstructor]. The original CPOConstructor.

See Also

Other getters and setters: CPO, getCPOAffect(), getCPOClass(), getCPOId(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), setCPOId()

Other CPO lifecycle related: CPO, CPOConstructor, CPOLearner, CPOTrained, NULLCPO, %>>%(), attachCPO(), composeCPO(), getCPOClass(), getCPOTrainedCPO(), identicalCPO(), makeCPO()

Other CPOConstructor related: CPOConstructor, getCPOClass(), getCPOName(), identicalCPO(), makeCPO(), print.CPOConstructor()

---

getCPOId — *Get the ID of a CPO Object*

Description

Gets the *id* of a CPO. The id can be set during construction by a CPOConstructor using the id parameter, or with setCPOId.

The exported hyperparameters of a CPO all have the id as prefix. This makes it possible to compose CPOs that have clashing parameter names.

Usage

```
getCPOId(cpo)
```

Arguments

cpo	[CPO] The cpo.

Value

[character(1)] the CPO's id.

See Also

Other getters and setters: CPO, getCPOAffect(), getCPOClass(), getCPOConstructor(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), setCPOId()

Other CPO ID related: setCPOId()

getCPOName *Get the CPO Object's Name*

Description

Return the name associated with a CPO operation. This name is set when creating a CPOConstructor, e.g. using makeCPO, by the ".cpo.name" parameter. It is also the default id, as retrieved by getCPOId, of a CPO.

Usage

```
getCPOName(cpo)

## S3 method for class 'CPOTrained'
getCPOName(cpo)

## S3 method for class 'CPOConstructor'
getCPOName(cpo)
```

Arguments

cpo	[CPO] The cpo.

Value

[character(1)] the CPO's name.

See Also

Other getters and setters: CPO, getCPOAffect(), getCPOClass(), getCPOConstructor(), getCPOId(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), setCPOId()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other CPOConstructor related: CPOConstructor, getCPOClass(), getCPOConstructor(), identicalCPO(), makeCPO(), print.CPOConstructor()

getCPOOperatingType *Determine the Operating Type of the CPO*

Description

Gives the *operating type* of a CPO or Retrafo, i.e. the part of a given data set it operates on. This can be "target" for a CPO / Retrafo / Inverter that manipulates target columns, "feature" for a CPO / Retrafo that manipulates non-target columns, or "retrafoless" for a CPO that only handles training data (and hence can manipulate both feature and target columns, but produces no retrafo).

For a composite CPO / Retrafo of different operating types, all types are returned. `NULLCPO` has no operating type.

Usage

```
getCPOOperatingType(cpo)
```

Arguments

`cpo`	[`CPO` | `CPOTrained`] The CPO, Retrafo, or Inverter to inspect.

Value

[`character(1)`]. Zero or more of "target", "feature", "retrafoless".

Operating types

There are three types of `CPO` that differ in their effects on the data: "*Feature Operation*", "*Target Operation*", and "*Retrafoless*".

Feature Operation CPOs (**FOCPO**) only change the feature columns of a data set, and don't change the target column(s). They therefore cannot change the type of a `Task`, and will never change the number of rows of a data set. They are the easiest CPO to handle, as they do not require inversion of predictions made with processed data. Examples of Feature Operation CPOs is the scaling of individual features to have unit variance (`cpoScale`), or the projection on principal components (`cpoPca`).

Target Operation CPOs (**TOCPO**) only change the target column(s) of a data set, not the feature columns. They can thus also change the type of a `Task`, and the PredictTypes admitted by a `Learner`. They are thus a powerful instrument, but they are harder to handle, since predictions made with data sets processed with this kind of CPO need to be *inverted* using the `invert` function and possibly an `CPOInverter` object (see documentation there). (Note that attaching a Target Operation CPO to a `Learner` will hide this complexity from the user and is the recommended way of handling it.) Examples of Target Operation CPOs are the log-transformation of the target column of a regression task, the conversion of a binary classification task into a 0-1-regression task, or the substitution of the target values into the residuals after a `Learner` was applied to the task. Note that the last of these examples distinguishes itself by the fact that the inversion operation is dependent on the *prediction* data used. While for the first two examples, the `CPORetrafo` object can be used for inversionk, the last one requires the CPOInverter object. See CPOTrainedCapability for more on this.

Retrafoless CPOs (**ROCPO**) can change the feature *and* target columns of a task, but this comes at the cost of not allowing retransformations. When getting the CPORetrafo object using retrafo, one will always get an identity transformation. While other CPOs can be understood as *transforming* the space of features or target values, respectively, the Retrafoless CPO can only *add* or *subtract* points in the given space. Examples of this operation are subsampling and supersampling.

See Also

Other getters and setters: CPO, getCPOAffect(), getCPOClass(), getCPOConstructor(), getCPOId(), getCPOName(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), setCPOId()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other CPO classifications: CPO, getCPOClass(), getCPOTrainedCapability()

---

getCPOPredictType *Get the CPO* `predict.type`

Description

Get the possible predict.types a CPO is able to handle.

The concept of a `predict.type` originates from predict.WrappedModel, which allows the estimation of different aspects of a prediction. This is, currently:

"response" A best estimate of the actual target value

"prob" An estimate of probabilities of different target values

"se" An estimate of the target value, together with an estimate of the standard error of this first estimation

A Target Operation CPO is able to change the type of a Task, but it can also enhance the type of predictions that a Learner can make for it. Thus a CPO that converts a binary classification into a regression task can use a regression learner to not only predict the "response" class, but also the estimated probability ("prob") distribution over the two classes. For this, the CPO declares

1. what `predict.types` a Learner, when attached to it, can provide, and
2. what `predict.type` the Learner, in each case, must be capable of.

This information is provided in the form of a named `character`, where the names are the provided predict type capabilities, and the values are the predict type that the underlying `Learner` must provide for this.

The CPO converting classification to regression mentioned above would thus have the `predict.type` of:

`c(response = "response", prob = "response")`

Another example would be a CPO that converts a multiclass classification problem into an ordinary classification problem, but uses the "prob" prediction of the underlying learner to make both the "response" and "prob" predictions. It would have the `predict.type` of:

`c(response = "prob", prob = "prob")`

If this second CPO is attached to a `Learner` that does not have the "prob" property (see `LearnerProperties`), an error is given.

CPOs that are not Target Operating always have the `predict.type` of:

`c(response = "response", prob = "prob", se = "se")`

Usage

```
getCPOPredictType(cpo)

## S3 method for class 'CPOTrained'
getCPOPredictType(cpo)
```

Arguments

cpo	[`CPO`] The cpo.

Value

[`character`]. A named `character` that maps potential predict types that a CPO may provide to the required predict type of an underlying learner.

See Also

Other getters and setters: `CPO`, `getCPOAffect()`, `getCPOClass()`, `getCPOConstructor()`, `getCPOId()`, `getCPOName()`, `getCPOOperatingType()`, `getCPOProperties()`, `getCPOTrainedCPO()`, `getCPOTrainedCapability()`, `setCPOId()`

Other retrafo related: `CPOTrained`, `NULLCPO`, `%>>%()`, `applyCPO()`, `as.list.CPO`, `clearRI()`, `getCPOClass()`, `getCPOName()`, `getCPOOperatingType()`, `getCPOProperties()`, `getCPOTrainedCPO()`, `getCPOTrainedCapability()`, `getCPOTrainedState()`, `is.retrafo()`, `makeCPOTrainedFromState()`, `pipeCPO()`, `print.CPOConstructor()`

Other inverter related: `CPOTrained`, `NULLCPO`, `%>>%()`, `applyCPO()`, `as.list.CPO`, `clearRI()`, `getCPOClass()`, `getCPOName()`, `getCPOOperatingType()`, `getCPOProperties()`, `getCPOTrainedCPO()`, `getCPOTrainedCapability()`, `getCPOTrainedState()`, `is.inverter()`, `makeCPOTrainedFromState()`, `pipeCPO()`, `print.CPOConstructor()`

getCPOProperties *Get the Properties of the Given CPO Object*

Description

The properties of a CPO object determine the kind of data the CPO will be able to handle, and how it transforms data. Properties describe what kind of data a CPO can work with.

By default, this function returns a list of three values: `$handling`, `$adding`, and `$needed`.

The `$handling` determines what data the CPO handles. If a CPO is applied to a data set (using `%>>%` or `applyCPO`, or indirectly when a `CPOLearner` is trained) that has a property not listed in `$handling`, an error will be given.

`$adding` can be one or many of the same values as `$handling`. These properties get added to a `Learner` or CPO coming after / behind this CPO. When a CPO imputes missing values, for example, this is "missings". This is always a subset of `$handling`.

`$properties.needed` can be one or many of the same values as `$handling`. These properties are required from a Learner (or CPO) coming after / behind this CPO. E.g., when a CPO converts factors to numerics, this is "numerics" (and `$adding` would be "factors" in this case). `$adding` and `$needed` never have any value in common.

There are two more properties mostly for internal usage: `$adding.min` and `$needed.max`. These are for internal checking of trafo / retrafo function return values: If some hyperparameter settings lead to a CPO returning values not conforming to properties (e.g. not removing all 'missings', or creating 'missings' where there were none before), while in other cases the CPO *does* conform, it is desirable to treat the CPO like it behaves in the best case (and rely on the user to make good hyperparameter choices). The properties discussed so far thus represent the CPO on its 'best' behaviour. Internally, each CPO also has a list of properties that it minimally 'adds' to its successors or maximally 'needs' from it in the worst case. These are `$adding.min` and `$needed.max`. `$adding.min` is always a subset of `$adding`, `$needed.max` is always a superset of needed. Their compliance is checked by the CPO framework, so a CPO that doesn't conform to these crashes.

Usage

```
getCPOProperties(cpo, only.data = FALSE, get.internal = FALSE)

## S3 method for class 'CPOTrained'
getCPOProperties(cpo, only.data = FALSE, get.internal = FALSE)
```

Arguments

`cpo`	[CPO] The cpo.
`only.data`	[`logical(1)`] Only get the CPO *data properties* (not target or task type properties). Default is `FALSE`.
`get.internal`	[`logical(1)`] Also retrieve `$adding.min` and `$needed.max`. Default is `FALSE`.

Value

[list]. A list with slots $handling, $adding, and $needed; also $adding.min and $needed.max if get.internal is TRUE.

Possible properties

data properties "numerics", "factors", "ordered", "missings": Whether any data column contains the type in question, or has missings. When only.data is TRUE, only these are returned.

task type properties "cluster" "classif" "multilabel" "regr" "surv": The type of the task. data.frame data objects have the implicit property "cluster".

target properties "oneclass" "twoclass" "multiclass": Whether the target column of a classif task has one, two, or more classes.

See Also

Other getters and setters: CPO, getCPOAffect(), getCPOClass(), getCPOConstructor(), getCPOId(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOTrainedCPO(), getCPOTrainedCapability(), setCPOId()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

---

getCPOTrainedCapability

Get the CPOTrained's Capabilities

Description

While CPOInverter is only used for inversion, both CPORetrafo and CPOInverter objects could be used for inversion using invert in principle. However, some CPORetrafo objects forbid inversion (and one must use the CPOInverter object instead), some CPORetrafo objects are NO-OPS when called with invert, some can be used both for transformation and inversion.

The CPOTrainedCapability is a named integer(2) with two slots: "retrafo" and "invert". Both can be 1 (CPOTrained does something when used in retrafo / inversion), 0 (CPOTrained is a NO-OP when used in retrafo / inversion) or -1 (CPOTrained cannot be used in retrafo / inversion).

Usage

```
getCPOTrainedCapability(cpo)
```

Arguments

cpo [CPOTrained]
The CPOTrained object to query.

Value

[named integer(2)]. The first component is named "retrafo" and specifies whether the object can perform retrafo operations; the second component is named "invert" and specifies whether it can perform invert operations. 0 indicates no effect for the operation, 1 indicates an operation is performed, -1 indicates the object cannot be used for the purpose.

Inverter capability

The invert capability of a CPOTrained depends on the CPO which was used to create it. Whenever a CPO is applied to some data, the result has the link{retrafo} and inverter attributes set that can be retrieved using the respectively named functions to get the CPORetrafo and CPOInverter object.

Every CPO can be a "Feature Operation" CPO, a "Target Operation" CPO, or a "Retrafoless" CPO, or a composition of these (see OperatingType).

If a (possibly compound) CPO contains only Feature Operation CPOs and Retrafoless CPOs, then it does not perform any operation on the target column of a data set; hence there is no inversion to be performed, the resulting CPORetrafo is a NO-OP when used with invert. The inverter attribute created is in fact a NULLCPO), while the retrafo attribute contains a CPORetrafo with capabilities c(retrafo = 1, invert = 0).

If a (possibly compound) CPO also contains Target Operation CPOs, but they are independent of the prediction data features–e.g. a CPO that takes the logarithm of the target column in a regression task–then the CPORetrafo object has enough information to perform inversion and hence can also meaningfully be used with invert. In this case the capability of the CPORetrafo will be c(retrafo = 1, invert = 1). The CPOInverter object retrieved using the inverter function can be used for the same task, but the benefit of the CPORetrafo object is that it can be used for *all* prediction data applied to it, while the CPOInverter object needs to be retrieved for each prediction data set anew. The CPOInverter object furthermore cannot be used for retrafo and hence has, like all CPOInverter, capabilities c(retrafo = -1, invert = 1).

If a (possibly compound) CPO contains Target Operation CPOs that are not prediction data independent then the resulting CPORetrafo has capability c(retrafo = 1, invert = -1), since the inversion requires information about the particular data set that was transformed.

A CPOInverter object *always* has capabilities c(retrafo = -1, invert = 1), since it can always be used for invert and never used in the place of a CPORetrafo.

The only object with capabilities c(retrafo = 0, invert = 0) is NULLCPO. Other objects that don't have at least one capability equal to 1 cannot be created.

See Also

Other getters and setters: CPO, getCPOAffect(), getCPOClass(), getCPOConstructor(), getCPOId(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), setCPOId()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other CPO classifications: CPO, getCPOClass(), getCPOOperatingType()

---

getCPOTrainedCPO — *Get CPO Used to Train a Retrafo / Inverter*

Description

Get the CPO used to create a CPOTrained object. The retrieved CPO will usually have all its hyperparameters and `affect.*` settings set to the values used to create the particular CPOTrained object. The only case where this is *not true* is if cpo is a CPOTrained that was created using makeCPOTrainedFromState.

Usage

```
getCPOTrainedCPO(cpo)
```

Arguments

cpo	[CPOTrained] The Retrafo or Inverter to get the original CPO from.

Value

[CPO]. The original CPO.

See Also

Other getters and setters: CPO, getCPOAffect(), getCPOClass(), getCPOConstructor(), getCPOId(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCapability(), setCPOId()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other CPO lifecycle related: CPO, CPOConstructor, CPOLearner, CPOTrained, NULLCPO, %>>%(), attachCPO(), composeCPO(), getCPOClass(), getCPOConstructor(), identicalCPO(), makeCPO()

---

getCPOTrainedState *Get the Internal State of a CPORetrafo Object*

Description

A CPOTrained always has access to some kind of state that represents information gotten from the training data, as well as the parameters it was called with.

Only primitive CPOTrained objects can be inspected like this. If the supplied CPOTrained is not primitive, split it into its constituents using as.list.CPOTrained.

The structure of the internal state depends on the CPO backend used. For Functional CPO, the state is the environment of the retrafo function, turned into a list. For Object based CPO, the state is a list containing the parameters, as well as the control object generated by the trafo function.

The object can be slightly modified and used to create a new CPOTrained object using makeCPOTrainedFromState.

Usage

```
getCPOTrainedState(trained.object)
```

Arguments

trained.object	[CPOTrained] The object to get the state of.

Value

[list]. A named list, containing the complete internal state of the CPOTrained.

See Also

Other state functions: makeCPOTrainedFromState()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

getLearnerBare — *Get the Learner with the CPOs Removed*

Description

Get the bare `Learner` without the `CPO`s that were previously added.

It is still possible for the result to be a wrapped learner, e.g. a TuningWrapper wrapped learner. It is also possible that below the tuning wrapper, there are more CPOs. These can and will not be removed.

This function is complementary to `getLearnerCPO`.

Usage

```
getLearnerBare(learner)
```

Arguments

`learner`	[`Learner`] The learner to strip.

Value

[`Learner`]. The learner without attached CPOs.

See Also

Other CPOLearner related: `CPOLearner`, `attachCPO()`, `getLearnerCPO()`

getLearnerCPO — *Get the CPO Associated with a Learner*

Description

Returns the (outermost) chain of `CPO`s that are part of a `Learner`. This is useful to inspect the preprocessing done by a learner object.

If there are hidden CPOs (e.g. if a learner has CPOs, but is then wrapped by a `TuneWrapper`), this function can not retrieve these CPOs, but it will emit a warning if `warn.buried` is `TRUE`.

The retrieved CPOs will have the hyperparameter set according to the hyperparameter settings of the Learner.

This function is complementary to `getLearnerBare`.

Usage

```
getLearnerCPO(learner, warn.buried = TRUE)
```

Arguments

learner	[Learner] The learner to query
warn.buried	[logical(1)] Whether to warn about CPOs that could not be retrieved.

Value

[CPO]. The (possibly composite) CPO found attached to `learner`.

See Also

Other CPOLearner related: `CPOLearner`, `attachCPO()`, `getLearnerBare()`

`identicalCPO` *Check Whether Two CPO are Fundamentally the Same*

Description

Check whether two `CPO` perform the same operation. This compares the inner workings of a `CPO`, but not the hyperparameter, hyperparameter-export, or `affect.*` settings of the `CPO`.

Internally, this checks whether the `CPOConstructor` used to create the two `CPO`s is identical. When creating new `CPOConstructor`s with `makeCPO` and related functions, it may be necessary to overload this function, if the resulting `CPO`s should be differentiated in a different way.

This function is used in `cpoCbind` to check for equality of underlying `CPO`s.

Usage

```
identicalCPO(cpo1, cpo2)
```

Arguments

cpo1	[CPO] The `CPO` to compare.
cpo2	[CPO] The `CPO` to compare.

Value

[`logical(1)`]. `TRUE` if the `CPO`s are fundamentally the same.

See Also

Other CPO lifecycle related: CPO, CPOConstructor, CPOLearner, CPOTrained, NULLCPO, %>>%(), attachCPO(), composeCPO(), getCPOClass(), getCPOConstructor(), getCPOTrainedCPO(), makeCPO()

Other CPOConstructor related: CPOConstructor, getCPOClass(), getCPOConstructor(), getCPOName(), makeCPO(), print.CPOConstructor()

---

invert — *Invert Target Preprocessing*

Description

Invert the transformation, done on the target column(s) of a data set, after prediction.

Use either a CPORetrafo object with invert capability (see getCPOTrainedCapability, or a CPOInverter retrieved with inverter from a data object that was fed through a retrafo chain.

If a CPORetrafo object is used that contains no target-bound transformations (i.e. has "invert" capability 0), this is a no-op.

Usage

```
invert(inverter, prediction, predict.type = "response")
```

Arguments

inverter	[CPOInverter] The retrafo or inverter to apply
prediction	[Prediction | matrix | data.frame] The prediction to invert
predict.type	[character(1)] The equivalent to the predict.type property of a Learner] object, control what kind of prediction to perform. One of "response", "se", "prob". Default is "response". Care must be taken that the prediction was generated with a prediction type that fits this, i.e. it must be of type getCPOPredictType(inverter)[predict.type].

Value

[Prediction | data.frame]. A transformed Prediction if a prediction was given, or a data.frame. If the first object in the chain is a CPORetrafo object, the 'truth' column(s) of the prediction will be dropped.

is.inverter *Check CPOInverter*

Description

Check whether the given object is a CPOInverter object.

Usage

```
is.inverter(x)
```

Arguments

x	[any] The object to check.

Value

TRUE if x has class CPOInverter, FALSE otherwise.

See Also

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

is.nullcpo *Check for NULLCPO*

Description

Check whether the given object is a NULLCPO.

Usage

```
is.nullcpo(x)
```

Arguments

x	[any] The object to check

Value

[logical(1)]. TRUE if x is a NULLCPO, FALSE otherwise.

See Also

Other NULLCPO related: NULLCPO, nullToNullcpo(), nullcpoToNull()

`is.retrafo` *Check CPORetrafo*

Description

Check whether the given object is a CPORetrafo object.

Usage

```
is.retrafo(x)
```

Arguments

x	[any] The object to check.

Value

TRUE if x has class CPORetrafo, FALSE otherwise.

See Also

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

`listCPO` *List all Built-in CPOs*

Description

Return a data.frame with the columns "name", "cponame", "category", "subcategory", "description".

Categories and subcategories are:

category	**subcategory**	**description**
meta		CPO that acts on other CPOs
tools		
data	general	
	general data preproc	
	factor data preproc	

	numeric data preproc	
	feature conversion	
	cleanup	
featurefilter	general	fltr CPO with operation arg
	specialised	specific feat filter CPO
imputation	general	imp CPO with operation arg
	specialised	specific imputation CPO
tools	imputation	

Usage

```
listCPO()
```

`makeCPO` *Create a Custom CPO Constructor*

Description

`makeCPO` creates a *Feature Operation* `CPOConstructor`, i.e. a constructor for a `CPO` that will operate on feature columns. `makeCPOTargetOp` creates a *Target Operation* `CPOConstructor`, which creates `CPO`s that operate on the target column. `makeCPORetrafoless` creates a *Retrafoless* `CPOConstructor`, which creates `CPO`s that may operate on both feature and target columns, but have no retrafo operation. See OperatingType for further details on the distinction of these. `makeCPOExtendedTrafo` creates a *Feature Operation* `CPOConstructor` that has slightly more flexibility in its data transformation behaviour than `makeCPO` (but is otherwise identical). `makeCPOExtendedTargetOp` creates a *Target Operation* `CPOConstructor` that has slightly more flexibility in its data transformation behaviour than `makeCPOTargetOp` but is otherwise identical.

See example section for some simple custom CPO.

Usage

```
makeCPO(
  cpo.name,
  par.set = makeParamSet(),
  par.vals = NULL,
  dataformat = c("df.features", "split", "df.all", "task", "factor", "ordered",
    "numeric"),
  dataformat.factor.with.ordered = TRUE,
  export.params = TRUE,
  fix.factors = FALSE,
  properties.data = c("numerics", "factors", "ordered", "missings"),
  properties.adding = character(0),
  properties.needed = character(0),
 properties.target = c("cluster", "classif", "multilabel", "regr", "surv", "oneclass",
    "twoclass", "multiclass"),
  packages = character(0),
  cpo.train,
```

```
  cpo.retrafo
)

makeCPOExtendedTrafo(
  cpo.name,
  par.set = makeParamSet(),
  par.vals = NULL,
  dataformat = c("df.features", "split", "df.all", "task", "factor", "ordered",
    "numeric"),
  dataformat.factor.with.ordered = TRUE,
  export.params = TRUE,
  fix.factors = FALSE,
  properties.data = c("numerics", "factors", "ordered", "missings"),
  properties.adding = character(0),
  properties.needed = character(0),
 properties.target = c("cluster", "classif", "multilabel", "regr", "surv", "oneclass",
    "twoclass", "multiclass"),
  packages = character(0),
  cpo.trafo,
  cpo.retrafo
)

makeCPORetrafoless(
  cpo.name,
  par.set = makeParamSet(),
  par.vals = NULL,
  dataformat = c("df.all", "task"),
  dataformat.factor.with.ordered = TRUE,
  export.params = TRUE,
  fix.factors = FALSE,
  properties.data = c("numerics", "factors", "ordered", "missings"),
  properties.adding = character(0),
  properties.needed = character(0),
 properties.target = c("cluster", "classif", "multilabel", "regr", "surv", "oneclass",
    "twoclass", "multiclass"),
  packages = character(0),
  cpo.trafo
)

makeCPOTargetOp(
  cpo.name,
  par.set = makeParamSet(),
  par.vals = NULL,
  dataformat = c("df.features", "split", "df.all", "task", "factor", "ordered",
    "numeric"),
  dataformat.factor.with.ordered = TRUE,
  export.params = TRUE,
  fix.factors = FALSE,
```

```
  properties.data = c("numerics", "factors", "ordered", "missings"),
  properties.adding = character(0),
  properties.needed = character(0),
  properties.target = "cluster",
  task.type.out = NULL,
  predict.type.map = c(response = "response"),
  packages = character(0),
  constant.invert = FALSE,
  cpo.train,
  cpo.retrafo,
  cpo.train.invert,
  cpo.invert
)

makeCPOExtendedTargetOp(
  cpo.name,
  par.set = makeParamSet(),
  par.vals = NULL,
  dataformat = c("df.features", "split", "df.all", "task", "factor", "ordered",
    "numeric"),
  dataformat.factor.with.ordered = TRUE,
  export.params = TRUE,
  fix.factors = FALSE,
  properties.data = c("numerics", "factors", "ordered", "missings"),
  properties.adding = character(0),
  properties.needed = character(0),
  properties.target = "cluster",
  task.type.out = NULL,
  predict.type.map = c(response = "response"),
  packages = character(0),
  constant.invert = FALSE,
  cpo.trafo,
  cpo.retrafo,
  cpo.invert
)
```

Arguments

`cpo.name`	[`character(1)`] The name of the resulting `CPOConstructor` / `CPO`. This is used for identification in output, and as the default `id`.
`par.set`	[`ParamSet`] Optional parameter set, for configuration of CPOs during construction or by hyperparameters. Default is an empty `ParamSet`. It is recommended to use `pSS` to construct this, as it greatly reduces the verbosity of creating a `ParamSet` and makes it more readable.
`par.vals`	[`list` | `NULL`] Named list of default parameter values for the CPO. These are used *instead*

of the parameter default values in `par.set`, if not `NULL`. It is preferred to use `ParamSet` default values, and not `par.vals`. Default is `NULL`.

`dataformat`
[character(1)]
Indicate what format the data should be as seen by the `cpo.train` and `cpo.retrafo` function. The following table shows what values of `dataformat` lead to what is given to `cpo.train` and `cpo.retrafo` as `data` and `target` parameter value. (Note that for Feature Operating CPOs, `cpo.retrafo` has no `target` argument.) Possibilities are:

dataformat	**data**	**target**
"df.all"	`data.frame` with target cols	target colnames
"df.features"	`data.frame` without target	`data.frame` of target
"task"	full `Task`	target colnames
"split"	list of `data.frame`s by type	`data.frame` of target
[type]	`data.frame` of [type] feats only	`data.frame` of target

[type] can be any one of "factor", "numeric", "ordered"; if these are given, only a subset of the total data present is seen by the `CPO`.

Note that `makeCPORetrafoless` accepts only "task" and "df.all".

For `dataformat == "split"`, `cpo.train` and `cpo.retrafo` get a list with entries "factor", "numeric", "other", and, if `dataformat.factor.with.ordered` is `FALSE`, "ordered".

If the CPO is a Feature Operation CPO, then the return value of the `cpo.retrafo` function must be in the same format as the one requested. E.g. if `dataformat` is "split", the return value must be a named list with entries `$numeric`, `$factor`, and `$other`. The types of the returned data may be arbitrary: In the given example, the `$factor` slot of the returned list may contain numeric data. (Note however that if data is returned that has a type not already present in the data, `properties.needed` must specify this.)

For Feature Operating CPOs, if `dataformat` is either "df.all" or "task", the target column(s) in the returned value of the retrafo function must be identical with the target column(s) given as input.

If `dataformat` is "split", the `$numeric` slot of the value returned by the `cpo.retrafo` function may also be a `matrix`. If `dataformat` is "numeric", the returned object may also be a matrix.

Default is "df.features" for all functions except `makeCPORetrafoless`, for which it is "df.all".

`dataformat.factor.with.ordered`
[logical(1)]
Whether to treat `ordered` typed features as `factor` typed features. This affects how `dataformat` is handled, for which it only has an effect if `dataformat` is "split" or "factor". If `dataformat` is "ordered", this must be `FALSE`. It also affects how strictly data fed to a `CPORetrafo` object is checked for adherence to the data format of data given to the generating `CPO`. Default is `TRUE`.

`export.params`
[logical(1) | character]
Indicates which CPO parameters are exported by default. Exported parameters can be changed after construction using `setHyperPars`, but exporting too many

parameters may lead to messy parameter sets if many CPOs are combined using `composeCPO` or `%>>%`. The exported parameters can be set during construction, but `export.params` determines the *default* exported parameters. If this is a `logical(1)`, `TRUE` exports all parameters, `FALSE` to exports no parameters. It may also be a `character`, indicating the names of parameters to be exported. Default is `TRUE`.

`fix.factors` [`logical(1)`]
Whether to constrain factor levels of new data to the levels of training data, for each factorial or ordered column. If new data contains factors that were not present in training data, the values are set to `NA`. Default is `FALSE`.

`properties.data`
[`character`]
The kind if data that the CPO will be able to handle. This can be one or more of: “numerics”, “factors”, “ordered”, “missings”. There should be a bias towards including properties. If a property is absent, the preproc operator will reject the data. If an operation e.g. only works on numeric columns that have no missings (like PCA), it is recommended to give all properties, ignore the columns that are not numeric (using `dataformat = "numeric"`), and giving an error when there are missings in the numeric columns (since missings in factorial features are not a problem). Defaults to the maximal set.

`properties.adding`
[`character`]
Can be one or many of the same values as `properties.data` for Feature Operation CPOs, and one or many of the same values as `properties.target` for Target Operation CPOs. These properties *get added* to a `Learner` (or `CPO`) coming after / behind this CPO. When a CPO imputes missing values, for example, this should be “missings”. This must be a subset of “properties.data” or “properties.target”.
Note that this may *not* contain a `Task`-type property, even if the `CPO` is a Target Operation CPO that performs conversion.
Property names may be postfixed with “.sometimes”, to indicate that adherence should not be checked internally. This distinction is made by not putting them in the `$adding.min` slot of the `getCPOProperties` return value when `get.internal = TRUE`.
Default is `character(0)`.

`properties.needed`
[`character`]
Can be one or many of the same values as `properties.data` for Feature Operation CPOs, and one or many of the same values as `properties.target`. These properties are *required* from a `Learner` (or `CPO`) coming after / behind this CPO. E.g., when a CPO converts factors to numerics, this should be “numerics” (and `properties.adding` should be “factors”).
Note that this may *not* contain a `Task`-type property, even if the `CPO` is a Target Operation CPO that performs conversion.
Property names may be postfixed with “.sometimes”, to indicate that adherence should not be checked internally. This distinction is made by not putting them in the `$needed` slot of properties. They can still be found in the `$needed.max` slot of the `getCPOProperties` return value when `get.internal = TRUE`.

Default is `character(0)`.

`properties.target`
: [character]
For Feature Operation CPOs, this can be one or more of “cluster”, “classif”, “multilabel”, “regr”, “surv”, “oneclass”, “twoclass”, “multiclass”. Just as `properties.data`, it indicates what kind of data a CPO can work with. To handle data given as `data.frame`, the “cluster” property is needed. Default is the maximal set.

For Target Operation CPOs, this *must* contain exactly one of “cluster”, “classif”, “multilabel”, “regr”, “surv”. This indicates the type of `Task` the `CPO` can work on. If the input is a `data.frame`, it is treated as a “cluster” type `Task`. If the `properties.target` contains “classif”, the value must then also contain one or more of “oneclass”, “twoclass”, or “multiclass”. Default is “cluster”.

`packages`
: [character]
Package(s) that should be loaded when the CPO is constructed. This gives the user an error if a package required for the CPO is not available on his system, or can not be loaded. Default is `character(0)`.

`cpo.train`
: [function | NULL]
This is a function which must have the parameters `data` and `target`, as well as the parameters specified in `par.set`. (Alternatively, the function may have only some of these arguments and a `dotdotdot` argument). It is called whenever a `CPO` is applied to a data set to prepare for transformation of the training *and* prediction data. Note that this function is only used in Feature Operating CPOs created with `makeCPO`, and in Target Operating CPOs created with `makeCPOExtendedTargetOp`.

The behaviour of this function differs slightly in Feature Operation and Target Operation CPOs.

For **Feature Operation CPOs**, if `cpo.retrafo` is `NULL`, this is a constructor function which must return a “retrafo” function which will then modify (possibly new unseen) data. This retrafo function must have exactly one argument–the (new) data–and return the modified data. The format of the argument, and of the return value of the retrafo function, depends on the value of the `dataformat` parameter, see documentation there.

If `cpo.retrafo` is not `NULL`, this is a function which must return a control object. This control object returned by `cpo.train` will then be given as the `control` argument of the `cpo.retrafo` function, along with (possibly new unseen) data to manipulate.

For **Target Operation CPOs**, if `cpo.retrafo` is `NULL`, `cpo.train.invert` (or `cpo.invert` if `constant.invert` is `TRUE`) must likewise be `NULL`. In that case `cpo.train`’s return value is ignored and it must define, within its namespace, two functions `cpo.retrafo` and `cpo.train.invert` (or `cpo.invert` if `constant.invert` is `TRUE`) which will take the place of the respective functions. `cpo.retrafo` must take the parameters `data` and `target`, and return the modified target `target` (or `data`, depending on `dataformat`) data. `cpo.train.invert` must take a `data` and `control` argument and return either a modified control object, or a `cpo.invert` function. `cpo.invert` must have a `target` and `predict.type` argument and return the modified target data.

If `cpo.retrafo` is not `NULL`, `cpo.train.invert` (or `cpo.invert` if `constant.invert` is TRUE) must likewise be non-NULL. In that case, cpo.train must return a control object. This control object will then be given as the control argument of both cpo.retrafo and cpo.train.invert (or the control.invert argument of cpo.invert if constant.invert is TRUE).

This parameter may be NULL, resulting in a so-called *stateless* CPO. For Target Operation CPOs created with makeCPOTargetOp, constant.invert must be TRUE in this case. A stateless CPO does the same transformation for initial CPO application and subsequent prediction data transformation (e.g. taking the logarithm of numerical columns). Note that cpo.retrafo and cpo.invert should not have a control argument in a stateless CPO.

cpo.retrafo [function | NULL]

This is a function which must have the parameters data, target (Target Operation CPOs only) and control, as well as the parameters specified in par.set. (Alternatively, the function may have only some of these arguments and a dotdotdot argument). In Feature Operation CPOs created with makeCPO, if cpo.train is NULL, the control argument must be absent.

This function gets called during the "retransformation" step where prediction data is given to the CPORetrafo object before it is given to a fitted machine learning model for prediction. In makeCPO Featore Operation CPOs and makeCPOTargetOp Target Operation CPOs, this is *also* called during the first trafo step, where the CPO object is applied to training data.

In Feature Operation CPOs, this function receives the data to be transformed and must return the transformed data in the same format as it received them. The format of data is the same as the format in cpo.train and cpo.trafo, with the exception that if dataformat is "task" or "df.all", the behaviour here is as if "df.split" had been given.

In Target Operation CPOs created with makeCPOTargetOp, this function receives the data and target to be transformed and must return the transformed target. The input format of these parameters depends on dataformat. If dataformat is "task" or "df.all", the returned value must be the modified Task / data.frame with the feature columns not modified. Otherwise, the target values to be modified are in the target parameter, and the return value must be a data.frame of the modified target values only.

In Target Operation CPOs created with makeCPOExtendedTargetOp, this function is called during the retrafo step, and it must create a control.invert object in its environment to be used in the inversion step, as well as return the modified target data.The format of the data given to cpo.retrafo in Target Operation CPOs created with makeCPOExtendedTargetOp is the same as in other functions, with the exception that, if dataformat is "df.all" or "task", the full data.frame or Task will be given as the target parameter, while the data parameter will behave as if dataformat "df.split". Depending on what object the CPORetrafo object was applied to, the target argument *may be* NULL; in that case NULL must also be returned by the function.

If cpo.invert is NULL, cpo.retrafo should create a cpo.invert function in its environment instead of creating the control object; this function should then take the target and predict.type arguments. If constant.invert is TRUE, this function does not need to define the control.invert or cpo.invert variables, they are instead taken from cpo.trafo.

cpo.trafo
: [function]
This is a function which must have the parameters data and target, as well as the parameters specified in par.set. (Alternatively, the function may have only some of these arguments and a dotdotdot argument). It is called whenever a CPO is applied to a data set to transform the training data, and (except for Retrafoless CPOs) to collect a control object used by other transformation functions. Note that this function is not used in makeCPO.

This functions primary task is to transform the given data when the CPO gets applied to training data. For Target Operating CPOs (created with makeCPOExtendedTargetOp(!)), it must return the complete transformed target column(s), unless dataformat is "df.all" (in which case the complete, modified, data.frame must be returned) or "task" (in which case the complete, modified, Task must be returned). It must furthermore create the control objects for cpo.retrafo and cpo.invert, or create these functins themselves, and save them in its function environment (see below). For Retrafoless CPOs (created with makeCPORetrafoless) and Feature Operation CPOs (created with makeCPOExtendedTrafo(!)), it must return the data in the same format as received it in its data argument (depending on dataformat). If dataformat is a df.all or task, this means the target column(s) contained in the data.frame or Task returned must not be modified.

For CPOs that are not Retrafoless, a unit of information to be carried over to the retrafo step needs to be created inside the cpo.trafo function. This unit of information is a variable that must be defined inside the environment of the cpo.trafo function and will be retrieved by the CPO framework.

If cpo.retrafo is not NULL the unit is an object named "control" that will be passed on as the control argument to the cpo.retrafo function. If cpo.retrafo is NULL, the unit is a *function*, called "cpo.retrafo", that will be used *instead* of the cpo.retrafo function passed over to makeCPOExtendedTargetOp / makeCPOExtendedTrafo. It must behave the same as the function it replaces, but has only the data (and target, for Target Operation CPOs) argument.

For Target Operation CPOs created with makeCPOExtendedTargetOp, another unit of information to be used by cpo.invert must be used. The options here are similar to cpo.retrafo: Either a control object, named control.invert, is created, or the cpo.invert function itself is given (and cpo.invert in the makeCPOExtendedTargetOp call is set to NULL), with the target and predict.type arguments.

task.type.out
: [character(1) | NULL]
If Task conversion is to take place, this is the output task that the data should be converted to. Note that the CPO framework takes care of the conversion if dataformat is not "task", but the target column needs to have the proper format for that.

If this is NULL, Tasks will not be converted. Default is NULL.

predict.type.map
: [character | list]
This becomes the CPO's predict.type, explained in detail in PredictType.

In short, the predict.type.map is a character vector, or a list of character(1), with *names* according to the predict types predict can request in its predict.type argument when the created CPO was used as part of a CPOLearner to create the model under consideration. The *values* of predict.type.map are the predict.type that will be requested from the underlying Learner for prediction.

predict.type.map thus determines the format that the target parameter of cpo.invert can take: It is the format according to predict.type.map[predict.type], where predict.type is the respective cpo.invert parameter.

constant.invert
: [logical(1)]
Whether the cpo.invert step should not have information from the previous cpo.retrafo or cpo.train.invert step in Target Operation CPOs (makeCPOTargetOp or makeCPOExtendedTargetOp).

For makeCPOTargetOp, if this is TRUE, the cpo.train.invert argument must be NULL. If cpo.retrafo and cpo.invert are given, the same control object is given to both of them. Otherwise, if cpo.retrafo and cpo.invert are NULL, the cpo.train function must return NULL and define a cpo.retrafo and cpo.invert function in its namespace (see cpo.train documentation for more details). If constant.invert is FALSE, cpo.train may either return a control object that will then be given to cpo.train.invert, or define a cpo.retrafo and cpo.train.invert function in its namespace.

For makeCPOExtendedTargetOp, if this is TRUE, cpo.retrafo does not need to generate a control.invert object. The control.invert object created in cpo.trafo will then always be given to cpo.invert for all data sets.

Default is FALSE.

cpo.train.invert
: This is a function which must have the parameters data, and control, as well as the parameters specified in par.set. (Alternatively, the function may have only some of these arguments and a dotdotdot argument).

This function receives the feature columns given for prediction, and must return a control object that will be passed on to the cpo.invert function, *or* it must return a *function* that will be treated as the cpo.invert function if the cpo.invert argument is NULL. In the latter case, the returned function takes exactly two arguments (the prediction column to be inverted, and predict.type), and otherwise behaves identically to cpo.invert.

If constant.invert is TRUE, this must be NULL.

cpo.invert
: [function | NULL]
This is a function which must have the parameters target (a data.frame containing the columns of a prediction made), control.invert, and predict.type, as well as the parameters specified in par.set. (Alternatively, the function may have only some of these arguments and a dotdotdot argument).

The predict.type *requested* by the predict or invert call is given as a character(1) in the predict.type argument. Note that this is not necessarily the predict.type of the prediction made and given as target argument, depending on the value of predict.type.map (see there).

This function performs the inversion for a Target Operation CPO. It takes a control object, which summarizes information from the training and retrafo step, and the prediction as returned by a machine learning model, and undoes the operation done to the target column in the cpo.trafo function.

For example, if the trafo step consisted of taking the logarithm of a regression target, the `cpo.invert` function could return the exponentiated prediction values by taking the `exp` of the only column in the `target` `data.frame` and returning the result of that. This kind of operation does not need the `cpo.retrafo` step and should have `skip.retrafo` set to `TRUE`.

As a more elaborate example, a CPO could train a model on the training data and set the target values to the *residues* of that trained model. The `cpo.retrafo` function would then make predictions with that model on the new prediction data and save the result to the `control` object. The `cpo.invert` function would then add these predictions to the predictions given to it in the `target` argument to "invert" the antecedent subtraction of model predictions from target values when taking the residues.

Value

[`CPOConstructor`]. A Constructor for `CPO`s.

CPO Internals

The mlrCPO package offers a powerful framework for handling the tasks necessary for preprocessing, so that the user, when creating custom CPOs, can focus on the actual data transformations to perform. It is, however, useful to understand *what* it is that the framework does, and how the process can be influenced by the user during CPO definition or application. Aspects of preprocessing that the user needs to influence are:

Operating Type The core of preprocessing is the actual transformation being performed. In the most general sense, there are three points in a machine learning pipeline that preprocessing can influence.

1. Transformation of training data *before model fitting*, done in mlr using `train`. In the CPO framework (*when not using a* `CPOLearner` *which makes all of these steps transparent to the user*), this is done by a `CPO`.
2. transformation of new validation or prediction data that is given to the fitted model for *prediction*, done using `predict`. This is done by a `CPORetrafo` retrieved using `retrafo` from the result of step 1.
3. transformation of the predictions made to invert the transformation of the target values done in step 1, which is done using the `CPOInverter` retrieved using `inverter` from the result of step 2.

The framework poses restrictions on primitive (i.e. not compound using `composeCPO`) `CPO`s to simplify internal operation: A `CPO` may be one of three OperatingTypes (see there). The *Feature Operation* `CPO` does not transform target columns and hence only needs to be involved in steps 1 and 2. The *Target Operation* `CPO` only transforms target columns, and therefore mostly concerns itself with steps 1 and 3. A *Retrafoless* `CPO` may change both feature and target columns, but may not perform a retrafo *or* inverter operation (and is therefore only concerned with step 1). Note that this is effectively a restriction on what kind of transformation a Retrafoless CPO may perform: it must not be a transformation of the data or target *space*, it may only act or subtract points within this space.

The Operating Type of a `CPO` is ultimately dependent on the function that was used to create the `CPOConstructor`: `makeCPO` / `makeCPOExtendedTrafo`, `makeCPOTargetOp` / `makeCPOExtendedTargetOp`, or `makeCPORetrafoless`.

Data Transformation At the core of a CPO is the modification of data it performs. For Feature Operation CPOs, the transformation of each row, during training *and* prediction, should happen in the same way, and it may only depend on the entirety of the *training* data–i.e. the value of a data row in a prediction data set may not influence the transformation of a different prediction data row. Furthermore, if a data row occurs in both training and prediction data, its transformation result should ideally be the same.

This property is ensured by `makeCPO` by splitting the transformation into two functions: One function that collects all relevant information from the training data (called `cpo.train`), and one that transforms given data, using this collected information and (*potentially new, unseen*) data to be transformed (called `cpo.retrafo`). The `cpo.retrafo` function should handle all data as if it were prediction data and unrelated to the data given to `cpo.train`.

Internally, when a `CPO` gets applied to a data set using `applyCPO`, the `cpo.train` function is called, and the resulting control object is used for a subsequent `cpo.retrafo` call which transforms the data. Before the result is given back from the `applyCPO` call, the control object is used to create a `CPORetrafo` object, which is attached to the result as attribute. Target Operating CPOs additionally create and add a `CPOInverter` object.

When a `CPORetrafo` is then applied to new prediction data, the control object previously returned by `cpo.train` is given, combined with this *new* data, to another `cpo.retrafo` call that performs the new transformation.

`makeCPOExtendedTrafo` gives more flexibility by having calling only the `cpo.trafo` in the training step, which both creates a control object *and* modifies the data. This can increase performance if the underlying operation creates a control object and the transformed data in one step, as for example *PCA* does. Note that the requirement that the same row in training and prediction data should result in the same transformation result still stands. The `cpo.trafo` function returns the transformed data and creates a local variable with the control information, which the CPO framework will access.

Inversion If a `CPO` performs transformations of the *target* column, the predictions made by a following machine learning process should ideally have this transformation undone, so that if the process makes a prediction that coincides with a target value *after* the transformation, the whole pipeline should return a prediction that equals to the target value *before* this transformation.

This is done by the `cpo.invert` function given to `makeCPOTargetOp`. It has access to information from both the preceding training and prediction steps. During the training step, `cpo.train` createas a `control` object that is not only given to `cpo.retrafo`, but also to `cpo.train.invert`. This latter function is called before the prediction step, whenever new data is fed to the machine learning process. It takes the new data and the old `control` object and transforms it to a new `control.invert` object to include information about the prediction data. This object is then given to `cpo.invert`.

It is possible to have Target Operation CPOs that do not require information from the retrafo step. This is specified by setting `constant.invert` to `TRUE`. It has the advantage that the same `CPOInverter` can be used for inversion of predictions made with any new data. Otherwise, a new `CPOInverter` object must be obtained for each new data set after the retrafo step (using the `inverter` function on the retrafo result). Having `constant.invert` set to `TRUE` results in *hybrid* retrafo / inverter objects: The `CPORetrafo` object can then also be used for `inversions`. When defining a `constant.invert` Target Operating CPO, no `cpo.train.invert` function is given, and the same `control` object is given to both `cpo.retrafo` and `cpo.invert`.

makeCPOExtendedTargetOp gives more flexibility and allows more efficient implementation of Target Operating CPOs at cost of more complexity. With this method, a cpo.trafo function is given that is executed during the first training step; It must return the transformed target column, as well as a control and control.invert object. The cpo.retrafo function not only transforms the target, but must also create a new control.invert object (unless constant.invert is TRUE). The semantics of cpo.invert is identical with the basic makeCPOTargetOp.

cpo.train-cpo.retrafo **information transfer** One possibility to transfer information from cpo.train to cpo.retrafo is to have cpo.train return a control object (a list) that is then given to cpo.retrafo. The CPO is then called an *object based* CPO.

Another possibility is to not give the cpo.retrafo argument (set it to NULL in the makeCPO call) and have cpo.train instead return a *function* instead. This function is then used as the cpo.retrafo function, and should have access to all relevant information about the training data as a closure. This is called *functional* CPO. To save memory, the actual data (including target) given to cpo.train is removed from the environment of its return value in this case (i.e. the environment of the cpo.retrafo function). This means the cpo.retrafo function may not reference a "data" variable.

There are similar possibilities of functional information transfer for other types of CPOs: cpo.trafo in makeCPOExtendedTargetOp may create a cpo.retrafo function instead of a control object. cpo.train in makeCPOTargetOp has the option of creating a cpo.retrafo and cpo.train.invert (cpo.invert if constant.invert is TRUE) function (and returning NULL) instead of returning a control object. Similarly, cpo.train.invert may return a cpo.invert function instead of a control.invert object. In makeCPOExtendedTargetOp, cpo.trafo may create a cpo.retrafo or a cpo.invert function, each optionally instead of a control or control.invert object (one *or* both may be functional). cpo.retrafo similarly may create a cpo.invert function instead of giving a control.invert object. Functional information transfer may be more parsimonious and elegant than control object information transfer.

Hyperparameters The action performed by a CPO may be influenced using *hyperparameters*, during its construction as well as afterwards (then using setHyperPars). Hyperparameters must be specified as a ParamSet and given as argument par.set. Default values for each parameter may be specified in this ParamSet or optionally as another argument par.vals.

Hyperparameters given are made part of the CPOConstructor function and can thus be given during construction. Parameter default values function as the default values for the CPOConstructor function parameters (which are thus made optional function parameters of the CPOConstructor function). The CPO framework handles storage and changing of hyperparameter values. When the cpo.train and cpo.retrafo functions are called to transform data, the hyperparameter values are given to them as arguments, so cpo.train and cpo.retrafo functions must be able to accept these parameters, either directly, or with a ... argument.

Note that with *functional* CPOs, the cpo.retrafo function does not take hyperparameter arguments (and instead can usually refer to them by its environment).

Hyperparameters may be *exported* (or not), thus making them available for setHyperPars. Not exporting a parameter has advantage that it does not clutter the ParamSet of a big CPO or CPOLearner pipeline with many hyperparameters. Which hyperparameters are exported is chosen during the constructing call of a CPOConstructor, but the default exported hyperparameters can be chosen with the export.params parameter.

Properties Similarly to Learners, CPOs may specify what kind of data they are and are not able to handle. This is done by specifying `.properties.*` arguments. The names of possible properties are the same as possible `LearnerProperties`, but since `CPO`s mostly concern themselves with data, only the properties indicating column and task types are relevant.

For each `CPO` one must specify

1. which kind of data does the `CPO` handle,
2. which kind of data must the `CPO` or `Learner` be able to handle that comes *after* the given `CPO`, and
3. which kind of data handling capability does the given `CPO` *add* to a following `CPO` or `Learner` if coming before it in a pipeline.

The specification of (1) is done with `properties.data` and `properties.target`, (2) is specified using `properties.needed`, and (3) is specified using `properties.adding`. Internally, `properties.data` and `properties.target` are concatenated and treated as one vector, they are specified separately in `makeCPO` etc. for convenience reasons. See `CPOProperties` for details.

The CPO framework checks the `cpo.retrafo` etc. functions for adherence to these properties, so it e.g. throws an error if a `cpo.retrafo` function adds missing values to some data but didn't declare "missings" in `properties.needed`. It may be desirable to have this internal checking happen to a laxer standard than the property checking when composing CPOs (e.g. when a CPO adds missings only with certain hyperparameters, one may still want to compose this CPO to another one that can't handle missings). Therefore it is possible to postfix listed properties with ".sometimes". The internal CPO checking will ignore these when listed in `properties.adding` (it uses the 'minimal' set of adding properties, `adding.min`), and it will not declare them externally when listed in `properties.needed` (but keeps them internally in the 'maximal' set of needed properties, `needed.max`). The `adding.min` and `needed.max` can be retrieved using `getCPOProperties` with `get.internal = TRUE`.

Data Format Different CPOs may want to change different aspects of the data, e.g. they may only care about numeric columns, they may or may not care about the target column values, sometimes they might need the actual task used as input. The CPO framework offers to present the data in a specified formats to the `cpo.train`, `cpo.retrafo` and other functions, to reduce the need for boilerplate data subsetting on the user's part. The format is requested using the `dataformat` and `dataformat.factor.with.ordered` parameter. A `cpo.retrafo` function is expected to return data in the same format as it requested, so if it requested a `Task`, it must return one, while if it only requested the feature `data.frame`, a `data.frame` must be returned.

Task Conversion Target Operation CPOs can be used for conversion between `Task`s. For this, the `type.out` value must be given. Task conversion works with all values of `dataformat` and is handled by the CPO framework. The `cpo.trafo` function must take care to return the target data in a proper format (see above). Note that for conversion, not only does the `Task` type need to be changed during `cpo.trafo`, but also the *prediction* format (see above) needs to change.

Fix Factors Some preprocessing for factorial columns needs the factor levels to be the same during training and prediction. This is usually not guarranteed by mlr, so the framework offers to do this if the `fix.factors` flag is set.

ID To prevent parameter name clashes when `CPO`s are concatenated, the parameters are prefixed with the `CPO`s *id*. The ID can be set during `CPO` construction, but will default to the `CPO`s *name* if not given. The name is set using the `cpo.name` parameter.

Packages Whenever a `CPO` needs certain packages to be installed to work, it can specify these in the `packages` parameter. The framework will check for the availability of the packages and throw an error if not found *during construction*. This means that loading a CPO from a savefile will omit this check, but in most cases it is a sufficient measure to make the user aware of missing packages in time.

Target Column Format Different Task types have the target in a different formats. They are listed here for reference. Target data is in this format when given to the `target` argument of some functions, and must be returned in this format by `cpo.trafo` in Target Operation CPOs. Target values are always in the format of a `data.frame`, even when only one column.

Task type	**target format**
"classif"	one column of `factor`
"cluster"	`data.frame` with zero columns.
"multilabel"	several columns of `logical`
"regr"	one column of `numeric`
"surv"	two columns of `numeric`

When inverting, the format of the `target` argument, as well as the return value of, the `cpo.invert` function depends on the Task type as well as the `predict.type`. The requested return value `predict.type` is given to the `cpo.invert` function as a parameter, the `predict.type` of the `target` parameter depends on this and the `predict.type.map` (see PredictType). The format of the prediction, depending on the task type and `predict.type`, is:

Task type	`predict.type`	**target format**
"classif"	"response"	`factor`
"classif"	"prob"	`matrix` with nclass cols
"cluster"	"response"	`integer` cluster index
"cluster"	"prob"	`matrix` with nclustr cols
"multilabel"	"response"	`logical` `matrix`
"multilabel"	"prob"	`matrix` with nclass cols
"regr"	"response"	`numeric`
"regr"	"se"	2-col `matrix`
"surv"	"response"	`numeric`
"surv"	"prob"	[NOT YET SUPPORTED]

All `matrix` formats are `numeric`, unless otherwise stated.

Headless function definitions

In the place of all `cpo.*` arguments, it is possible to make a *headless* function definition, consisting only of the function body. This function body must always begin with a '{'. For example, instead of `cpo.retrafo = function(data, control) data[-1]`, it is possible to use `cpo.retrafo = function(data, control) { data[-1] }`. The necessary function head is then added automatically by the CPO framework. This will always contain the necessary parameters (e.g. "`data`", "`target`", hyperparameters as defined in `par.set`) in the names as required. This can declutter the definition of a `CPOConstructor` and is recommended if the CPO consists of few lines.

Note that if this is used when writing an R package, inside a function, this may lead to the automatic R correctness checker to print warnings.

See Also

Other CPOConstructor related: `CPOConstructor`, `getCPOClass()`, `getCPOConstructor()`, `getCPOName()`, `identicalCPO()`, `print.CPOConstructor()`

Other CPO lifecycle related: `CPO`, `CPOConstructor`, `CPOLearner`, `CPOTrained`, `NULLCPO`, `%>>%()`, `attachCPO()`, `composeCPO()`, `getCPOClass()`, `getCPOConstructor()`, `getCPOTrainedCPO()`, `identicalCPO()`

Examples

```
# an example constant feature remover CPO
constFeatRem = makeCPO("constFeatRem",
 dataformat = "df.features",
 cpo.train = function(data, target) {
   names(Filter(function(x) {  # names of columns to keep
       length(unique(x)) > 1
     }, data))
   }, cpo.retrafo = function(data, control) {
   data[control]
 })
# alternatively:
constFeatRem = makeCPO("constFeatRem",
  dataformat = "df.features",
  cpo.train = function(data, target) {
    cols.keep = names(Filter(function(x) {
        length(unique(x)) > 1
      }, data))
    # the following function will do both the trafo and retrafo
    result = function(data) {
      data[cols.keep]
    }
    result
  }, cpo.retrafo = NULL)
```

`makeCPOCase` *Build Data-Dependent CPOs*

Description

This is a `CPOConstructor` to be used to create a `CPO`. It is called like any R function and returns the created `CPO`.

The meta CPO which determines what CPO to apply to a data depending on a provided function. Many parameters coincide with the parameters of `makeCPO`, it is suggested to read the relevant parameter description there.

`makeCPOCase` creates a `CPOConstructor`, while cpoCase can be used as `CPOConstructor` itself.

Usage

```
makeCPOCase(
  par.set = makeParamSet(),
  par.vals = list(),
  export.cpos = list(),
  dataformat = c("df.features", "split", "df.all", "task", "factor", "ordered",
    "numeric"),
  dataformat.factor.with.ordered = TRUE,
  properties.data = NULL,
  properties.adding = NULL,
  properties.needed = NULL,
  properties.target = NULL,
  cpo.build
)

cpoCase(
  par.set = makeParamSet(),
  par.vals = list(),
  export.cpos = list(),
  dataformat = c("df.features", "split", "df.all", "task", "factor", "ordered",
    "numeric"),
  dataformat.factor.with.ordered = TRUE,
  properties.data = NULL,
  properties.adding = NULL,
  properties.needed = NULL,
  properties.target = NULL,
  cpo.build,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

`par.set`
: [`ParamSet`]
Parameters (additionally to the exported CPOs) of the CPO. Default is the empty ParamSet.

`par.vals`
: [`list`]
Named list of default parameter values for the CPO. These are used additionally to the parameter default values of `par.set`. It is often more elegant to use these default values, and not `par.vals`. Default is `list()`. Default is `list()`.

export.cpos
: [list of CPO]
List of CPO objects that have their hyperparameters exported. If this is a named list, the names must be unique and represent the parameter name by which they are given to the cpo.build function. They are also the IDs that will be given to the CPOs upon construction. If the list is not named, the IDs (or default names, in case of CPOConstructors), are used instead, and need to be unique.
All CPOs in the list must either be all Feature Operation CPOs, all Target Operation CPOs performing the same conversion, or all Retrafoless CPOs.
The cpo.build function needs to have an argument for each of the names in the list. The CPO objects are pre-configured by the framework to have the hyperparameter settings as set by the ones exported by cpoCase. Default is list().

dataformat
: [character(1)]
Indicate what format the data should be as seen by "cpo.build". See the parameter in makeCPO for details.
Note that if the CPOs in export.cpos are Retrafoless CPOs, this must be either "task" or "df.all". Default is "df.features".

dataformat.factor.with.ordered
: [logical(1)]
Whether to treat ordered typed features as factor typed features. See the parameter in makeCPO. Default is TRUE.

properties.data
: [character]
See the parameter in makeCPO.
The properties of the resulting CPO are calculated from the constituent CPOs automatically in the most lenient way. If this parameter is not NULL, the calculated the given properties are used instead of the calculated properties.
Default is NULL.

properties.adding
: [character]
See the parameter in makeCPO.
The properties of the resulting CPO are calculated from the constituent CPOs automatically in the most lenient way. If this parameter is not NULL, the calculated the given properties are used instead of the calculated properties.
Default is NULL.

properties.needed
: [character]
See the parameter in makeCPO.
The properties of the resulting CPO are calculated from the constituent CPOs automatically in the most lenient way. If this parameter is not NULL, the calculated the given properties are used instead of the calculated properties.
Default is NULL.

properties.target
: [character]
See the parameter in makeCPO.
The properties of the resulting CPO are calculated from the constituent CPOs automatically in the most lenient way. If this parameter is not NULL, the calculated the given properties are used instead of the calculated properties.

Default is `NULL`.

`cpo.build`
: [`function`]
This function works similar to `cpo.trafo` in `makeCPO`: It has the arguments `data`, `target`, one argument for each hyperparameter declared in `par.set`. However, it also has one parameter for each entry in `export.cpos`, named by each item in that list. The cpoCase framework supplies the pre-configured `CPO`s (pre-configured as the exported hyperparameters of `cpoCase` demand) to the `cpo.build` code via these parameters. The return value of `cpo.build` must be a `CPO`, which will then be used on the data.
Just as `cpo.trafo` in `makeCPO`, this can also be a 'headless' function; it then must be written as an expression, starting with a `{`.

`id`
: [`character(1)`]
id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

`export`
: [`character`]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

`affect.type`
: [`character` | `NULL`]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or `NULL` to not match by column type. Default is `NULL`.

`affect.index`
: [`numeric`]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is `integer(0)`.

`affect.names`
: [`character`]
Feature names of feature columns to affect. The order of names given is respected. Default is `character(0)`.

`affect.pattern`
: [`character(1)` | `NULL`]
`grep` pattern to match feature names by. Default is `NULL` (no pattern matching)

`affect.invert`
: [`logical(1)`]
Whether to affect all features *not* matched by other `affect.*` parameters.

`affect.pattern.ignore.case`
: [`logical(1)`]
Ignore case when matching features with `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.perl`
: [`logical(1)`]
Use Perl-style regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

`affect.pattern.fixed`
: [logical(1)]
Use fixed matching instead of regular expressions for `affect.pattern`; see `grep`. Default is `FALSE`.

Value

[`CPO`].

General CPO info

This function creates a CPO object, which can be applied to `Task`s, `data.frame`s, `link[mlr]{Learner}`s and other CPO objects using the `%>>%` operator.

The parameters of this object can be changed after creation using the function `setHyperPars`. The other hyper-parameter manipulating functins, `getHyperPars` and `getParamSet` similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOMultiplex()`

Other special CPOs: `cpoCbind()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOMultiplex()`

makeCPOMultiplex *CPO Multiplexer*

Description

This is a CPOConstructor to be used to create a CPO. It is called like any R function and returns the created CPO.

makeCPOMultiplex creates a CPOConstructor, cpoMultiplex *is* a CPOConstructor.

Usage

```
makeCPOMultiplex(cpos, selected.cpo = NULL)

cpoMultiplex(
  cpos,
  selected.cpo = NULL,
  id,
  export = "export.default",
  affect.type = NULL,
  affect.index = integer(0),
  affect.names = character(0),
  affect.pattern = NULL,
  affect.invert = FALSE,
  affect.pattern.ignore.case = FALSE,
  affect.pattern.perl = FALSE,
  affect.pattern.fixed = FALSE
)
```

Arguments

cpos	[list of (CPO | CPOConstructor)] The CPOs to multiplex. If this is a named list, the names must be unique and represent the index by which selected.cpo selects CPOs. They are also the IDs that will be given to the CPOs upon construction. If the list is not named, the IDs (or default names, in case of CPOConstructors), are used instead, and need to be unique. All CPOs in the list must either be all Feature Operation CPOs, all Target Operation CPOs performing the same conversion, or all Retrafoless CPOs.
selected.cpo	[character(1)] Selected CPO. Will default to the first item of cpos if NULL. Default is NULL.
id	[character(1)] id to use as prefix for the CPO's hyperparameters. this must be used to avoid name clashes when composing two CPOs of the same type, or with learners or other CPOS with hyperparameters with clashing names.

export [character]
Either a character vector indicating the parameters to export as hyperparameters, or one of the special values "export.all" (export all parameters), "export.default" (export all parameters that are exported by default), "export.set" (export all parameters that were set during construction), "export.default.set" (export the intersection of the "default" and "set" parameters), "export.unset" (export all parameters that were *not* set during construction) or "export.default.unset" (export the intersection of the "default" and "unset" parameters). Default is "export.default".

affect.type [character | NULL]
Type of columns to affect. A subset of "numeric", "factor", "ordered", "other", or NULL to not match by column type. Default is NULL.

affect.index [numeric]
Indices of feature columns to affect. The order of indices given is respected. Target column indices are not counted (since target columns are always included). Default is integer(0).

affect.names [character]
Feature names of feature columns to affect. The order of names given is respected. Default is character(0).

affect.pattern [character(1) | NULL]
grep pattern to match feature names by. Default is NULL (no pattern matching)

affect.invert [logical(1)]
Whether to affect all features *not* matched by other affect.* parameters.

affect.pattern.ignore.case
[logical(1)]
Ignore case when matching features with affect.pattern; see grep. Default is FALSE.

affect.pattern.perl
[logical(1)]
Use Perl-style regular expressions for affect.pattern; see grep. Default is FALSE.

affect.pattern.fixed
[logical(1)]
Use fixed matching instead of regular expressions for affect.pattern; see grep. Default is FALSE.

Value

[CPO].

General CPO info

This function creates a CPO object, which can be applied to Tasks, data.frames, link[mlr]{Learner}s and other CPO objects using the %>>% operator.

The parameters of this object can be changed after creation using the function setHyperPars. The other hyper-parameter manipulating functins, getHyperPars and getParamSet similarly work as one expects.

If the "id" parameter is given, the hyperparameters will have this id as aprefix; this will, however, not change the parameters of the creator function.

Calling a `CPOConstructor`

CPO constructor functions are called with optional values of parameters, and additional "special" optional values. The special optional values are the `id` parameter, and the `affect.*` parameters. The `affect.*` parameters enable the user to control which subset of a given dataset is affected. If no `affect.*` parameters are given, all data features are affected by default.

See Also

Other CPOs: `cpoApplyFun()`, `cpoApplyFunRegrTarget()`, `cpoAsNumeric()`, `cpoCache()`, `cpoCbind()`, `cpoCollapseFact()`, `cpoDropConstants()`, `cpoDropMostlyConstants()`, `cpoDummyEncode()`, `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`, `cpoFixFactors()`, `cpoIca()`, `cpoImpactEncodeClassif()`, `cpoImpactEncodeRegr()`, `cpoImpute()`, `cpoImputeConstant()`, `cpoImputeHist()`, `cpoImputeLearner()`, `cpoImputeMax()`, `cpoImputeMean()`, `cpoImputeMedian()`, `cpoImputeMin()`, `cpoImputeMode()`, `cpoImputeNormal()`, `cpoImputeUniform()`, `cpoLogTrafoRegr()`, `cpoMakeCols()`, `cpoMissingIndicators()`, `cpoModelMatrix()`, `cpoOversample()`, `cpoPca()`, `cpoProbEncode()`, `cpoQuantileBinNumerics()`, `cpoRegrResiduals()`, `cpoResponseFromSE()`, `cpoSample()`, `cpoScale()`, `cpoScaleMaxAbs()`, `cpoScaleRange()`, `cpoSelect()`, `cpoSmote()`, `cpoSpatialSign()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`

Other special CPOs: `cpoCbind()`, `cpoTransformParams()`, `cpoWrap()`, `makeCPOCase()`

`makeCPOTrainedFromState`

Create a CPOTrained with Given Internal State

Description

This creates a new `CPOTrained` object which will behave according to the given state. The state should usually be obtained using `getCPOTrainedState` and then slightly modified. No checks for correctness of the state will (or can) be done, it is the user's responsibility to ensure that the correct `CPOConstructor` is used, and that the state is only modified in a way the CPO can handle.

Usage

```
makeCPOTrainedFromState(constructor, state, get.inverter = FALSE)
```

Arguments

constructor	[CPOConstructor] A cpo constructor.
state	[list] A state gotten from another CPORetrafo or CPOInverter object using getCPOTrainedState.
get.inverter	[logical(1)] Whether to get a CPOInverter. Usually a CPORetrafo is created. This must be TRUE if the state was created from a CPOInverter, FALSE otherwise. Default is FALSE.

Value

[CPOTrained]. A CPORetrafo or CPOInverter (as if retrieved using retrafo or inverter after a primitive CPO was applied to some data) with the given state.

See Also

Other state functions: getCPOTrainedState()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), pipeCPO(), print.CPOConstructor()

NULLCPO *CPO Composition Neutral Element*

Description

NULLCPO is the neutral element of CPO and CPOTrained composition when using %>>% or composeCPO. It is furthermore no effect when attached to a Learner using attachCPO (or %>>%), or when applied to data using applyCPO, invert, or predict (or, again, %>>%).

NULLCPO works as a stand-in for certain operations that have an "empty" return value: It is returned when retrafo and inverter are applied to an object that has no retrafo or inverter associated with it, and by pipeCPO when applied to an empty list.

NULLCPO can be checked using is.nullcpo, and converted from or to NULL using nullToNullcpo and nullcpoToNull. Otherwise it behaves very similarly to other CPO or CPOTrained objects.

Usage

```
NULLCPO
```

Format

An object of class NULLCPO (inherits from CPOPrimitive, CPORetrafo, CPOInverter, CPOTrained, CPO) of length 0.

See Also

Other retrafo related: CPOTrained, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other CPO lifecycle related: CPO, CPOConstructor, CPOLearner, CPOTrained, %>>%(), attachCPO(), composeCPO(), getCPOClass(), getCPOConstructor(), getCPOTrainedCPO(), identicalCPO(), makeCPO()

Other NULLCPO related: is.nullcpo(), nullToNullcpo(), nullcpoToNull()

---

nullcpoToNull *NULLCPO to NULL*

Description

Convert NULLCPO to NULL, leave other values intact.

Usage

```
nullcpoToNull(cpo)
```

Arguments

cpo	[CPO] The cpo.

Value

[CPO | NULL]. NULL if cpo is NULLCPO, cpo otherwise.

See Also

Other NULLCPO related: NULLCPO, is.nullcpo(), nullToNullcpo()

nullToNullcpo	*NULL to NULLCPO*

Description

Convert NULL to NULLCPO, leave other values intact.

Usage

```
nullToNullcpo(cpo)
```

Arguments

cpo	[CPO | NULL] The CPO.

Value

[CPO]. NULLCPO if cpo is NULL, cpo otherwise.

See Also

Other NULLCPO related: NULLCPO, is.nullcpo(), nullcpoToNull()

pipeCPO	*Turn a* list *of CPOs into a Single Chained One*

Description

Chain a list of preprocessing operators, or retrafo objects, turning list(a, b, c) into a %>>% b %>>% c.

This is the inverse of as.list.CPO / as.list.CPOTrained when applied to CPO or CPOTrained.

Usage

```
pipeCPO(pplist)
```

Arguments

pplist	[list of CPO | list of CPOTrained] A list of CPO or CPOTrained objects.

Value

[CPO \| CPOTrained]. The compound CPO(Trained) obtained when chaining the elements of the input list.

See Also

Other operators: CPO, %>>%(), applyCPO(), as.list.CPO, attachCPO(), composeCPO()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), print.CPOConstructor()

---

print.CPOConstructor *Print CPO Objects*

Description

Prints a simple representation of a CPOConstructor, CPO or CPOTrained. If verbose is TRUE, more information about the given objects will be given. For CPOConstructor, that is the trafo and retrafo functions, for CPO, the individual constituents of a compound CPO will be printed.

Verbose printing can also be done using the ! operator. !cpo is equivalent to print(cpo, verbose = TRUE).

Usage

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

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

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

## S3 method for class 'CPOConstructor'
!x

## S3 method for class 'CPO'
!x

## S3 method for class 'CPOTrained'
!x
```

Arguments

x	[CPOConstructor | CPO | CPOTrained] The CPOConstructor to print.
verbose	[logical(1)] Whether to print further information. Default is FALSE.
...	[any] Further arguments.

Value

[invisible(NULL)].

See Also

Other CPOConstructor related: CPOConstructor, getCPOClass(), getCPOConstructor(), getCPOName(), identicalCPO(), makeCPO()

Other retrafo related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO()

Other inverter related: CPOTrained, NULLCPO, %>>%(), applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO()

---

pSS *Turn the argument list into a* ParamSet

Description

pSS, short for "ParamSet Sugar", is a shorthand API for makeParamSet which enables entry of ParamSets in short form. It behaves similarly to makeParamSet, but instead of having to construct each parameter individually, the parameters can be given in shorthand form with a convenient syntax, making use of R's nonstandard evaluation.

This makes definition of ParamSets shorter and more readable.

The difference between pSS and pSSLrn is only in the default value of .pss.learner.params being FALSE for the former and TRUE for the latter.

Usage

```
pSS(..., .pss.learner.params = FALSE, .pss.env = parent.frame())

pSSLrn(..., .pss.learner.params = TRUE, .pss.env = parent.frame())
```

Arguments

... Parameters, see Details below.

.pss.learner.params
[logical]
Whether to create LearnerParam instead of Param objects. Default is TRUE for pSSLrn and FALSE for pSS.

.pss.env [environment]
Which environment to use when evaluating expressions. Defaults to the calling function's frame.

Details

The arguments are of the form

name = default: type range [^ dimension] [settings].

name is any valid R identifier name.

= default Determines the 'default' setting in makeXXXParam. Note that this is different from an R function parameter default value, in that it serves only as information to the user and does not set the parameter to this value if it is not given. To define 'no default', use NA or leave the "= default" part out. Leaving it out can cause problems when R's static type checker verifies a package, so this is *only* recommended for interactive sessions and top-level applications. (To actually set a parameter default to NA, use (NA) in parentheses)

type is one of "integer", "numeric", "logical", "discrete", "funct", "character", "untyped". Each of these types leads to a Param or LearnerParam of the given type to be created. Note that "character" is not available if 'Learner'-parameters are created.

range is optional and only used for *integer*, *numeric*, and *discrete* parameters. For "discrete", it is either [valuelist] with valuelist evaluating to a list, or of the form [value1, value2, ...], creating a discrete parameter of character or numeric values according to value1, value2 etc. If type is one of "integer" or "numeric", range is of the form [lowBound, upBound], where lowBound and upBound must either be numerical (or integer) values indicating the lower and upper bound, or may be missing (indicating the absence of a bound). To indicate an exclusive bound, prefix the values with a tilde ("~"). For a "numeric" variable, to indicate an unbounded value which may not be infinite, you can use ~Inf or ~-Inf, or use tilde-dot ("~.").

^ dimension is optionally determining the dimension of a 'vector' parameter. If it is absent, the result is a normal Param or LearnerParam, if it is present, the result is a Vector(Learner)Param. Note that a one-dimensional Vector(Learner)Param is distinct from a normal (Learner)Param.

settings may be a collection of further settings to supply to makeXXXParam and is optional. To specify one or more settings, put in double square brackets ([[,]]), and comma-separate settings if more than one is present.

Examples

```
pSSLrn(a = NA: integer [~0, ]^2 [[requires = expression(b != 0)]],
       b = -10: numeric [~., 0],
       c: discrete [x, y, 1],
       d: logical,
       e: integer)
```

```
# is equivalent to

makeParamSet(
    makeIntegerVectorLearnerParam("a", len = 2, lower = 1,  # note exclusive bound
         upper = Inf, requires = expression(b != 0)),
    makeNumericLearnerParam("b", lower = -Inf, upper = 0,
         allow.inf = FALSE, default = -10),  # note infinite value is prohibited.
    makeDiscreteLearnerParam("c", values = list(x = "x", y = "y", `1` = 1)),
    makeLogicalLearnerParam("d"),
    makeIntegerLearnerParam("e"))
```

---

randomForestSRC_filters *Filter "randomForestSRC_importance" computes the importance of random forests fitted in package* **randomForestSRC**. *The concrete method is selected via the 'method' parameter. Possible values are 'permute' (default), 'random', 'anti', 'permute.ensemble', 'random.ensemble', 'anti.ensemble'. See the VIMP section in the docs for [randomForestSRC::rfsrc] for details.*

Description

Filter "randomForestSRC_importance" computes the importance of random forests fitted in package **randomForestSRC**. The concrete method is selected via the 'method' parameter. Possible values are 'permute' (default), 'random', 'anti', 'permute.ensemble', 'random.ensemble', 'anti.ensemble'. See the VIMP section in the docs for [randomForestSRC::rfsrc] for details.

See Also

Other filter: `cpoFilterAnova()`, `cpoFilterCarscore()`, `cpoFilterChiSquared()`, `cpoFilterFeatures()`, `cpoFilterGainRatio()`, `cpoFilterInformationGain()`, `cpoFilterKruskal()`, `cpoFilterLinearCorrelation()`, `cpoFilterMrmr()`, `cpoFilterOneR()`, `cpoFilterPermutationImportance()`, `cpoFilterRankCorrelation()`, `cpoFilterRelief()`, `cpoFilterRfCImportance()`, `cpoFilterRfImportance()`, `cpoFilterRfSRCImportance()`, `cpoFilterSymmetricalUncertainty()`, `cpoFilterUnivariate()`, `cpoFilterVariance()`

---

setCPOId *Set the ID of a CPO Object*

Description

Sets the *id* of a `CPO`. Setting the id is also possible during construction by a `CPOConstructor` using the `id` parameter.

The exported hyperparameters of a CPO will all have the id as prefix. This makes it possible to compose CPOs that have clashing parameter names.

Usage

```
setCPOId(cpo, id)
```

Arguments

cpo	[CPO] The cpo.
id	[character(1) | NULL] The ID. If this is NULL, the ID is set to the default for the CPO at hand, which is the CPO "name", see getCPOName.

Value

[CPO] the CPO with modified id.

See Also

Other getters and setters: CPO, getCPOAffect(), getCPOClass(), getCPOConstructor(), getCPOId(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability()

Other CPO ID related: getCPOId()

---

untyped *defined to avoid problems with the static type checker*

Description

defined to avoid problems with the static type checker

Usage

```
untyped()
```

%>>% *CPO Composition / Attachment / Application Operator*

Description

This operator "pipes" data from the source into the sink object.

If both objects are a `CPO` object, or both are a `CPOTrained` object, they will be composed. A new object, representing the operation of performing both object's operations in succession, will be created, which can be handled like a new CPO or CPOTrained object. See `composeCPO`.

If the source object is a `data.frame` or a `link[mlr]{Task}`, the transformation operation will be applied to this data, and the same resulting data will be returned. See `applyCPO`.

If the sink object is a `Learner`, the CPO will be attached to this learner. The same operation will be performed during the `train` and `predict` phase; the behaviour during the predict phase may furthermore be depend on the training data. See `attachCPO`.

Note that you can not link a `data.frame` or `Task` directly to a `Learner`, since this operation is not algebraically associative with the composition of CPOs. Use `train` for this.

The `%<<%` operator is synonymous with `%>>%` with source and sink argument swapped.

The `%>|%` and `%|<%` operators perform piping followed by application of `retrafo`. The `%>|%` evaluates the expression to its right before the expression to its left, so it may be used in the most natural way without parentheses:

`data %>|% cpo1 %>>% cpo2`

is the same as

`retrafo(data %>>% cpo1 %>>% cpo2)`.

The `%<>>%` and `%<«%` operators perform the piping operation and assign the result to the left hand variable. This way it is possible to apply a `CPO`, or to attach a `CPO` to a `Learner`, and just keep the resulting object. The assignment operators evaluate their right hand side before their left hand side, so it is possible to build long chains that end up writing to the leftmost variable. Therefore the expression

`data %<>>% cpo1 %<>>% cpo2 %>>% cpo3`

is the same as

```
cpo1 = cpo1 %>>% cpo2 %>>% cpo3
data = data %>>% cpo1
```

Usage

```
cpo1 %>>% cpo2

cpo2 %<<% cpo1

cpo1 %<>>% cpo2

cpo2 %<<<% cpo1
```

```
cpo1 %>|% cpo2

cpo2 %|<% cpo1
```

Arguments

cpo1 — [data.frame | Task | CPO | CPOTrained]
The source object.

cpo2 — [CPO | CPOTrained | Learner]
The sink object.

Value

[data.frame | Task | CPO | CPOTrained].

See Also

Other operators: CPO, applyCPO(), as.list.CPO, attachCPO(), composeCPO(), pipeCPO()

Other retrafo related: CPOTrained, NULLCPO, applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.retrafo(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other inverter related: CPOTrained, NULLCPO, applyCPO(), as.list.CPO, clearRI(), getCPOClass(), getCPOName(), getCPOOperatingType(), getCPOPredictType(), getCPOProperties(), getCPOTrainedCPO(), getCPOTrainedCapability(), getCPOTrainedState(), is.inverter(), makeCPOTrainedFromState(), pipeCPO(), print.CPOConstructor()

Other CPO lifecycle related: CPO, CPOConstructor, CPOLearner, CPOTrained, NULLCPO, attachCPO(), composeCPO(), getCPOClass(), getCPOConstructor(), getCPOTrainedCPO(), identicalCPO(), makeCPO()

Examples

```
# PCA-rotate pid.task
rotated.pid.task = pid.task %>>% cpoScale() %>>% cpoPca()

# Centering / Scaling *after* PCA
newPCA = cpoPca() %>>% cpoScale()

# Attach the above to learner
pcaLogreg = newPCA %>>% makeLearner("classif.logreg")

# append cpoAsNumeric to newPCA
newPCA %<>>% cpoAsNumeric()
print(newPCA)

# prepend cpoAsNumeric to pcaLogreg
pcaLogreg %<<<% cpoAsNumeric()
```

Index