Package 'mlr3filters'

July 18, 2020

Title Filter Based Feature Selection for 'mlr3'

Version 0.3.0

Description Extends 'mlr3' with filter methods for feature selection. Besides standalone filter methods built-in methods of any machine-learning algorithm are supported. Partial scoring of multivariate filter methods is supported.

License LGPL-3

URL https://mlr3filters.mlr-org.com, https://github.com/mlr-org/mlr3filters

BugReports https://github.com/mlr-org/mlr3filters/issues

Depends R (>= 3.1.0)

Imports backports, checkmate, data.table, mlr3 (>= 0.1.8), mlr3misc, paradox, R6

Suggests care, caret, FSelectorRcpp, lgr, mlr3learners, mlr3measures, praznik, rpart, testthat

Encoding UTF-8

NeedsCompilation no

RoxygenNote 7.1.1

Collate 'Filter.R' 'mlr_filters.R' 'FilterAUC.R' 'FilterAnova.R' 'FilterCMIM.R' 'FilterCarScore.R' 'FilterCorrelation.R' 'FilterDISR.R' 'FilterFindCorrelation.R' 'FilterImportance.R' 'FilterInformationGain.R' 'FilterJMI.R' 'FilterJMIM.R' 'FilterKruskalTest.R' 'FilterMIM.R' 'FilterMRMR.R' 'FilterNJMIM.R' 'FilterPerformance.R' 'FilterPermutation.R' 'FilterVariance.R' 'flt.R' 'reexports.R' 'zzz.R'

Author Patrick Schratz [aut, cre] (<https://orcid.org/0000-0003-0748-6624>), Michel Lang [aut] (<https://orcid.org/0000-0001-9754-0393>), Bernd Bischl [aut] (<https://orcid.org/0000-0001-6002-6980>), Martin Binder [aut]

Maintainer Patrick Schratz <patrick.schratz@gmail.com>

Repository CRAN

Date/Publication 2020-07-18 09:30:02 UTC

R topics documented:

`mlr3filters-package` *mlr3filters: Filter Based Feature Selection for 'mlr3'*

Description

Extends 'mlr3' with filter methods for feature selection. Besides standalone filter methods built-in methods of any machine-learning algorithm are supported. Partial scoring of multivariate filter methods is supported.

Author(s)

Maintainer: Patrick Schratz <patrick.schratz@gmail.com> (ORCID)

Authors:

- Michel Lang <michellang@gmail.com> (ORCID)
- Bernd Bischl <bernd_bischl@gmx.net> (ORCID)
- Martin Binder <mlr.developer@mb706.com>

See Also

Useful links:

- https://mlr3filters.mlr-org.com
- https://github.com/mlr-org/mlr3filters
- Report bugs at https://github.com/mlr-org/mlr3filters/issues

---

`Filter` *Filter Base Class*

Description

Base class for filters. Predefined filters are stored in the dictionary mlr_filters. A Filter calculates a score for each feature of a task. Important features get a large value and unimportant features get a small value. Note that filter scores may also be negative.

Details

Some features support partial scoring of the feature set: If `nfeat` is not `NULL`, only the best `nfeat` features are guaranteed to get a score. Additional features may be ignored for computational reasons, and then get a score value of `NA`.

Public fields

`id` (`character(1)`)
Identifier of the object. Used in tables, plot and text output.

`task_type` (`character(1)`)
Task type, e.g. `"classif"` or `"regr"`.
For a complete list of possible task types (depending on the loaded packages), see `mlr_reflections$task_types$type`

`task_properties` (`character()`)
mlr3::Tasktask properties.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`feature_types` (`character()`)
Feature types of the filter.

`packages` (`character()`)
Packages which this filter is relying on.

`man` (`character(1)`)
String in the format [pkg]::[topic] pointing to a manual page for this object. Defaults to `NA`, but can be set by child classes.

`scores` Stores the calculated filter score values as named numeric vector. The vector is sorted in decreasing order with possible `NA` values last. Tied values (this includes `NA` values) appear in a random, non-deterministic order.

Methods

Public methods:

- `Filter$new()`
- `Filter$format()`
- `Filter$print()`
- `Filter$help()`
- `Filter$calculate()`
- `Filter$clone()`

Method `new()`**:** Create a Filter object.

Usage:

```
Filter$new(
  id,
  task_type,
  task_properties = character(),
  param_set = ParamSet$new(),
  feature_types = character(),
  packages = character(),
  man = NA_character_
)
```

Arguments:

`id (character(1))`
: Identifier for the filter.

`task_type (character())`
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`task_properties (character())`
: Required task properties, see mlr3::Task. Must be a subset of `mlr_reflections$task_properties`.

`param_set (paradox::ParamSet)`
: Set of hyperparameters.

`feature_types (character())`
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

`packages (character())`
: Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`man (character(1))`
: String in the format [pkg]::[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

Method `format()`**:** Format helper for Filter class

Usage:

```
Filter$format()
```

Method `print()`**:** Printer for Filter class

Usage:

```
Filter$print()
```

Method `help()`: Opens the corresponding help page referenced by field $man.

Usage:

```
Filter$help()
```

Method `calculate()`: Calculates the filter score values for the provided mlr3::Task and stores them in field `scores`. `nfeat` determines the minimum number of features to score (see details), and defaults to the number of features in `task`. Loads required packages and then calls `private$.calculate()` of the respective subclass. If the task has no rows, each feature gets the score `NA`.

Usage:

```
Filter$calculate(task, nfeat = NULL)
```

Arguments:

`task` (mlr3::Task)
mlr3::Task to calculate the filter scores for.

`nfeat` (`integer()`)
THe minimum number of features to calculate filter scores for.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Filter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Filter: `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

`flt` *Syntactic Sugar for Filter Construction*

Description

These functions complements mlr_filters with a function in the spirit of mlr3::mlr_sugar.

Usage

```
flt(.key, ...)

flts(.keys, ...)
```

Arguments

.key	(character(1)) Key passed to the respective dictionary to retrieve the object.
...	(named list()) Named arguments passed to the constructor, to be set as parameters in the paradox::ParamSet, or to be set as public field. See mlr3misc::dictionary_sugar_get() for more details.
.keys	(character()) Keys passed to the respective dictionary to retrieve multiple objects.

Value

Filter.

Examples

```
flt("correlation", method = "kendall")
flts(c("mrmr", "jmim"))
```

---

mlr_filters *Dictionary of Filters*

Description

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

This dictionary can get populated with additional filters by add-on packages.

For a more convenient way to retrieve and construct filters, see flt().

Usage

```
mlr_filters
```

Format

R6Class object

Usage

See Dictionary.

See Also

Other Filter: Filter, mlr_filters_anova, mlr_filters_auc, mlr_filters_carscore, mlr_filters_cmim, mlr_filters_correlation, mlr_filters_disr, mlr_filters_find_correlation, mlr_filters_importance, mlr_filters_information_gain, mlr_filters_jmim, mlr_filters_jmi, mlr_filters_kruskal_test, mlr_filters_mim, mlr_filters_mrmr, mlr_filters_njmim, mlr_filters_performance, mlr_filters_permutation, mlr_filters_variance

Examples

```
mlr_filters$keys()
as.data.table(mlr_filters)
mlr_filters$get("mim")
flt("anova")
```

`mlr_filters_anova` *ANOVA F-Test Filter*

Description

ANOVA F-Test filter calling `stats::aov()`. Note that this is equivalent to a t-test for binary classification.

The filter value is `-log10(p)` where `p` is the p-value. This transformation is necessary to ensure numerical stability for very small p-values.

Super class

`mlr3filters::Filter` -> `FilterAnova`

Methods

Public methods:

- `FilterAnova$new()`
- `FilterAnova$clone()`

Method `new()`**:** Create a FilterAnova object.

Usage:

```
FilterAnova$new(
  id = "anova",
  task_type = "classif",
  task_properties = character(),
  param_set = ParamSet$new(),
  feature_types = c("integer", "numeric"),
  packages = "stats"
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the filter.

`task_type` (`character()`)
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`task_properties` (`character()`)
: Required task properties, see mlr3::Task. Must be a subset of `mlr_reflections$task_properties`.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`feature_types (character())`
Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

`packages (character())`
Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterAnova$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
filter = flt("anova")
filter$calculate(task)
head(as.data.table(filter), 3)

# transform to p-value
10^(-filter$scores)
```

---

mlr_filters_auc *AUC Filter*

Description

Area under the (ROC) Curve filter, analogously to `mlr3measures::auc()` from **mlr3measures**. Missing values of the features are removed before calculating the AUC. If the AUC is undefined for the input, it is set to 0.5 (random classifier). The absolute value of the difference between the AUC and 0.5 is used as final filter value.

Super class

`mlr3filters::Filter` -> `FilterAUC`

Methods

Public methods:

- `FilterAUC$new()`
- `FilterAUC$clone()`

Method `new()`**:** Create a FilterAUC object.

Usage:

```
FilterAUC$new(
  id = "auc",
  task_type = "classif",
  task_properties = "twoclass",
  param_set = ParamSet$new(),
  packages = "mlr3measures",
  feature_types = c("integer", "numeric")
)
```

Arguments:

`id (character(1))`
: Identifier for the filter.

`task_type (character())`
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`task_properties (character())`
: Required task properties, see mlr3::Task. Must be a subset of `mlr_reflections$task_properties`.

`param_set (paradox::ParamSet)`
: Set of hyperparameters.

`packages (character())`
: Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types (character())`
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterAUC$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("pima")
filter = flt("auc")
filter$calculate(task)
head(as.data.table(filter), 3)
```

`mlr_filters_carscore` *Conditional Mutual Information Based Feature Selection Filter*

Description

Calculates the Correlation-Adjusted (marginal) coRelation scores (short CAR scores) implemented in `care::carscore()` in package **care**. The CAR scores for a set of features are defined as the correlations between the target and the decorrelated features. The filter returns the absolute value of the calculated scores.

Argument `verbose` defaults to `FALSE`.

Super class

`mlr3filters::Filter` -> `FilterCarScore`

Methods

Public methods:

- `FilterCarScore$new()`
- `FilterCarScore$clone()`

Method `new()`**:** Create a FilterCarScore object.

Usage:

```
FilterCarScore$new(
  id = "carscore",
  task_type = "regr",
 param_set = ParamSet$new(list(ParamDbl$new("lambda", lower = 0, upper = 1, default =
  NO_DEF), ParamLgl$new("diagonal", default = FALSE), ParamLgl$new("verbose", default =
    TRUE))),
  packages = "care",
  feature_types = "numeric"
)
```

Arguments:

`id` (`character(1)`)
Identifier for the filter.

`task_type` (`character()`)
Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`packages` (`character()`)
Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types` (`character()`)
Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterCarScore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("mtcars")
filter = flt("carscore")
filter$calculate(task)
head(as.data.table(filter), 3)

## changing filter settings
filter = flt("carscore")
filter$param_set$values = list("diagonal" = TRUE)
filter$calculate(task)
head(as.data.table(filter), 3)
```

---

mlr_filters_cmim *Minimal Conditional Mutual Information Filter*

Description

Minimal conditional mutual information maximisation filter calling `praznik::CMIM()` from package **praznik**.

This filter supports partial scoring (see Filter).

Super class

`mlr3filters::Filter` -> `FilterCMIM`

Methods

Public methods:

- `FilterCMIM$new()`
- `FilterCMIM$clone()`

Method `new()`**:** Create a FilterCMIM object.

Usage:

```
FilterCMIM$new(
  id = "cmim",
  task_type = c("classif", "regr"),
 param_set = ParamSet$new(list(ParamInt$new("threads", lower = 0L, default = 0L))),
  feature_types = c("integer", "numeric", "factor", "ordered"),
  packages = "praznik"
)
```

Arguments:

`id` (`character(1)`)
Identifier for the filter.

`task_type` (`character()`)
Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`feature_types` (`character()`)
Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

`packages` (`character()`)
Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterCMIM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
filter = flt("cmim")
filter$calculate(task, nfeat = 2)
as.data.table(filter)
```

`mlr_filters_correlation` *Correlation Filter*

Description

Simple correlation filter calling `stats::cor()`. The filter score is the absolute value of the correlation.

Super class

`mlr3filters::Filter` -> `FilterCorrelation`

Methods

Public methods:

- `FilterCorrelation$new()`
- `FilterCorrelation$clone()`

Method `new()`**:** Create a FilterCorrelation object.

Usage:

```
FilterCorrelation$new(
  id = "correlation",
  task_type = "regr",
 param_set = ParamSet$new(list(ParamFct$new("use", default = "everything", levels =
    c("everything", "all.obs", "complete.obs", "na.or.complete",
   "pairwise.complete.obs")), ParamFct$new("method", default = "pearson", levels =
    c("pearson", "kendall", "spearman")))),
  packages = "stats",
  feature_types = c("integer", "numeric")
)
```

Arguments:

`id` (`character(1)`)
Identifier for the filter.

`task_type` (`character()`)
Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`packages (character())`
: Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types (character())`
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterCorrelation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
## Pearson (default)
task = mlr3::tsk("mtcars")
filter = flt("correlation")
filter$calculate(task)
as.data.table(filter)

## Spearman
filter = FilterCorrelation$new()
filter$param_set$values = list("method" = "spearman")
filter$calculate(task)
as.data.table(filter)
```

---

mlr_filters_disr *Double Input Symmetrical Relevance Filter*

Description

Double input symmetrical relevance filter calling `praznik::DISR()` from package **praznik**.

This filter supports partial scoring (see Filter).

Super class

`mlr3filters::Filter` -> `FilterDISR`

Methods

Public methods:

- `FilterDISR$new()`
- `FilterDISR$clone()`

Method `new()`**:** Create a FilterDISR object.

Usage:

```
FilterDISR$new(
  id = "disr",
  task_type = "classif",
 param_set = ParamSet$new(list(ParamInt$new("threads", lower = 0L, default = 0L))),
  packages = "praznik",
  feature_types = c("integer", "numeric", "factor", "ordered")
)
```

Arguments:

`id (character(1))`
Identifier for the filter.

`task_type (character())`
Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`packages (character())`
Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types (character())`
Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterDISR$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
filter = flt("disr")
filter$calculate(task, nfeat = 2)
as.data.table(filter)
```

`mlr_filters_find_correlation` *Correlation Filter*

Description

Simple filter emulating `caret::findCorrelation(exact = FALSE)`.

This gives each feature a score between 0 and 1 that is *one minus* the cutoff value for which it is excluded when using `caret::findCorrelation()`. The negative is used because `caret::findCorrelation()` excludes everything *above* a cutoff, while filters exclude everything below a cutoff. Here the filter scores are shifted by +1 to get positive values for to align with the way other filters work.

Subsequently `caret::findCorrelation(cutoff = 0.9)` lists the same features that are excluded with `FilterFindCorrelation` at score 0.1 (= 1 - 0.9).

Super class

`mlr3filters::Filter` -> `FilterFindCorrelation`

Methods

Public methods:

- `FilterFindCorrelation$new()`
- `FilterFindCorrelation$clone()`

Method `new()`**:** Create a FilterFindCorrelation object.

Usage:

```
FilterFindCorrelation$new(
  id = "find_correlation",
  task_type = c("classif", "regr"),
 param_set = ParamSet$new(list(ParamFct$new("use", default = "everything", levels =
    c("everything", "all.obs", "complete.obs", "na.or.complete",
   "pairwise.complete.obs")), ParamFct$new("method", default = "pearson", levels =
    c("pearson", "kendall", "spearman")))),
  packages = "stats",
  feature_types = c("integer", "numeric")
)
```

Arguments:

`id (character(1))`
Identifier for the filter.

`task_type` (`character()`)
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`packages` (`character()`)
: Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types` (`character()`)
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterFindCorrelation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
## Pearson (default)
task = mlr3::tsk("mtcars")
filter = flt("find_correlation")
filter$calculate(task)
as.data.table(filter)

## Spearman
filter = flt("find_correlation", method = "spearman")
filter$calculate(task)
as.data.table(filter)
```

---

mlr_filters_importance

Filter for Embedded Feature Selection via Variable Importance

---

Description

Variable Importance filter using embedded feature selection of machine learning algorithms. Takes a mlr3::Learner which is capable of extracting the variable importance (property "importance"), fits the model and extracts the importance values to use as filter scores.

Super class

`mlr3filters::Filter` -> `FilterImportance`

Public fields

`learner` (mlr3::Learner)
Learner to extract the importance values from.

Methods

Public methods:

- `FilterImportance$new()`
- `FilterImportance$clone()`

Method `new()`**:** Create a FilterImportance object.

Usage:

```
FilterImportance$new(
  id = "importance",
  task_type = learner$task_type,
  feature_types = learner$feature_types,
  learner = mlr3::lrn("classif.rpart"),
  packages = learner$packages,
  param_set = learner$param_set
)
```

Arguments:

`id` (`character(1)`)
Identifier for the filter.

`task_type` (`character()`)
Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`feature_types` (`character()`)
Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

`learner` (mlr3::Learner)
Learner to extract the importance values from.

`packages` (`character()`)
Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterImportance$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: Filter, mlr_filters_anova, mlr_filters_auc, mlr_filters_carscore, mlr_filters_cmim, mlr_filters_correlation, mlr_filters_disr, mlr_filters_find_correlation, mlr_filters_information_gain, mlr_filters_jmim, mlr_filters_jmi, mlr_filters_kruskal_test, mlr_filters_mim, mlr_filters_mrmr, mlr_filters_njmim, mlr_filters_performance, mlr_filters_permutation, mlr_filters_variance, mlr_filters

Examples

```
task = mlr3::tsk("iris")
learner = mlr3::lrn("classif.rpart")
filter = flt("importance", learner = learner)
filter$calculate(task)
as.data.table(filter)
```

---

mlr_filters_information_gain

Information Gain Filter

Description

Information gain filter calling FSelectorRcpp::information_gain() in package **FSelectorRcpp**. Set parameter "type" to "gainratio" to calculate the gain ratio, or set to "symuncert" to calculate the symmetrical uncertainty (see FSelectorRcpp::information_gain()). Default is "infogain".

Argument equal defaults to FALSE for classification tasks, and to TRUE for regression tasks.

Super class

mlr3filters::Filter -> FilterInformationGain

Methods

Public methods:

- FilterInformationGain$new()
- FilterInformationGain$clone()

Method new(): Create a FilterInformationGain object.

Usage:

```
FilterInformationGain$new(
  id = "information_gain",
  task_type = c("classif", "regr"),
  param_set = ParamSet$new(list(ParamFct$new("type", levels = c("infogain",
   "gainratio", "symuncert"), default = "infogain"), ParamLgl$new("equal", default =
   FALSE), ParamLgl$new("discIntegers", default = TRUE), ParamInt$new("threads", lower =
    0L, default = 1L))),
```

```
  packages = "FSelectorRcpp",
  feature_types = c("integer", "numeric", "factor", "ordered")
)
```

Arguments:

`id (character(1))`
: Identifier for the filter.

`task_type (character())`
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set (paradox::ParamSet)`
: Set of hyperparameters.

`packages (character())`
: Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types (character())`
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterInformationGain$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
## InfoGain (default)
task = mlr3::tsk("pima")
filter = flt("information_gain")
filter$calculate(task)
head(filter$scores, 3)
as.data.table(filter)

## GainRatio

filterGR = flt("information_gain")
filterGR$param_set$values = list("type" = "gainratio")
filterGR$calculate(task)
head(as.data.table(filterGR), 3)
```

`mlr_filters_jmi` *Joint Mutual Information Filter*

Description

Joint mutual information filter calling `praznik::JMI()` in package **praznik**.

This filter supports partial scoring (see Filter).

Super class

`mlr3filters::Filter` -> `FilterJMI`

Methods

Public methods:

- `FilterJMI$new()`
- `FilterJMI$clone()`

Method `new()`**:** Create a FilterJMI object.

Usage:

```
FilterJMI$new(
  id = "jmi",
  task_type = "classif",
 param_set = ParamSet$new(list(ParamInt$new("threads", lower = 0L, default = 0L))),
  packages = "praznik",
  feature_types = c("integer", "numeric", "factor", "ordered")
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the filter.

`task_type` (`character()`)
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`packages` (`character()`)
: Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types` (`character()`)
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterJMI$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
filter = flt("jmi")
filter$calculate(task, nfeat = 2)
as.data.table(filter)
```

---

mlr_filters_jmim *Minimal Joint Mutual Information Maximisation Filter*

Description

Minimal joint mutual information maximisation filter calling `praznik::JMIM()` in package **praznik**.

This filter supports partial scoring (see Filter).

Super class

`mlr3filters::Filter` -> `FilterJMIM`

Methods

Public methods:

- `FilterJMIM$new()`
- `FilterJMIM$clone()`

Method `new()`**:** Create a FilterJMIM object.

Usage:

```
FilterJMIM$new(
  id = "jmim",
  task_type = "classif",
 param_set = ParamSet$new(list(ParamInt$new("threads", lower = 0L, default = 0L))),
  packages = "praznik",
  feature_types = c("integer", "numeric", "factor", "ordered")
)
```

Arguments:

`id (character(1))`
Identifier for the filter.

`task_type` (`character()`)
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`packages` (`character()`)
: Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types` (`character()`)
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterJMIM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
filter = flt("jmim")
filter$calculate(task, nfeat = 2)
as.data.table(filter)
```

---

mlr_filters_kruskal_test

Kruskal-Wallis Test Filter

Description

Kruskal-Wallis rank sum test filter calling `stats::kruskal.test()`.

The filter value is `-log10(p)` where `p` is the p-value. This transformation is necessary to ensure numerical stability for very small p-values.

Super class

`mlr3filters::Filter` -> `FilterKruskalTest`

Methods

Public methods:

- `FilterKruskalTest$new()`
- `FilterKruskalTest$clone()`

Method `new()`**:** Create a FilterKruskalTest object.

Usage:

```
FilterKruskalTest$new(
  id = "kruskal_test",
  task_type = "classif",
 param_set = ParamSet$new(list(ParamFct$new("na.action", default = "na.omit", levels =
    c("na.omit", "na.fail", "na.exclude", "na.pass")))),
  packages = "stats",
  feature_types = c("integer", "numeric")
)
```

Arguments:

`id` (`character(1)`)
Identifier for the filter.

`task_type` (`character()`)
Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`packages` (`character()`)
Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types` (`character()`)
Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterKruskalTest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
filter = flt("kruskal_test")
filter$calculate(task)
as.data.table(filter)

# transform to p-value
10^(-filter$scores)
```

`mlr_filters_mim` *Conditional Mutual Information Based Feature Selection Filter*

Description

Conditional mutual information based feature selection filter calling `praznik::MIM()` in package **praznik**.

This filter supports partial scoring (see Filter).

Super class

`mlr3filters::Filter` -> `FilterMIM`

Methods

Public methods:

- `FilterMIM$new()`
- `FilterMIM$clone()`

Method `new()`**:** Create a FilterMIM object.

Usage:

```
FilterMIM$new(
  id = "mim",
  task_type = "classif",
 param_set = ParamSet$new(list(ParamInt$new("threads", lower = 0L, default = 0L))),
  packages = "praznik",
  feature_types = c("integer", "numeric", "factor", "ordered")
)
```

Arguments:

`id` (`character(1)`)
Identifier for the filter.

`task_type` (`character()`)
Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`packages (character())`
Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types (character())`
Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterMIM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
filter = flt("mim")
filter$calculate(task, nfeat = 2)
as.data.table(filter)
```

---

mlr_filters_mrmr *Minimum redundancy maximal relevancy filter*

Description

Minimum redundancy maximal relevancy filter calling `praznik::MRMR()` in package **praznik**.

This filter supports partial scoring (see Filter).

Super class

`mlr3filters::Filter` -> `FilterMRMR`

Methods

Public methods:

- `FilterMRMR$new()`
- `FilterMRMR$clone()`

Method `new()`**:** Create a FilterMRMR object.

Usage:

```
FilterMRMR$new(
  id = "mrmr",
  task_type = "classif",
 param_set = ParamSet$new(list(ParamInt$new("threads", lower = 0L, default = 0L))),
  packages = "praznik",
  feature_types = c("integer", "numeric", "factor", "ordered")
)
```

Arguments:

`id (character(1))`
Identifier for the filter.

`task_type (character())`
Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`packages (character())`
Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types (character())`
Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterMRMR$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
filter = flt("mrmr")
filter$calculate(task, nfeat = 2)
as.data.table(filter)
```

`mlr_filters_njmim` *Minimal Normalised Joint Mutual Information Maximisation Filter*

Description

Minimal normalised joint mutual information maximisation filter calling `praznik::NJMIM()` from package **praznik**.

This filter supports partial scoring (see Filter).

Super class

`mlr3filters::Filter` -> `FilterNJMIM`

Methods

Public methods:

- `FilterNJMIM$new()`
- `FilterNJMIM$clone()`

Method `new()`**:** Create a FilterNJMIM object.

Usage:

```
FilterNJMIM$new(
  id = "njmim",
  task_type = "classif",
 param_set = ParamSet$new(list(ParamInt$new("threads", lower = 0L, default = 0L))),
  packages = "praznik",
  feature_types = c("integer", "numeric", "factor", "ordered")
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the filter.

`task_type` (`character()`)
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`packages` (`character()`)
: Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types (character())`
Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterNJMIM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_performance`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
filter = flt("njmim")
filter$calculate(task, nfeat = 2)
as.data.table(filter)
```

---

mlr_filters_performance

Predictive Performance Filter

Description

Filter which uses the predictive performance of a mlr3::Learner as filter score. Performs a `mlr3::resample()` for each feature separately. The filter score is the aggregated performance of the mlr3::Measure, or the negated aggregated performance if the measure has to be minimized.

Super class

`mlr3filters::Filter` -> `FilterPerformance`

Public fields

`learner` (mlr3::Learner)

`resampling` (mlr3::Resampling)

`measure` (mlr3::Measure)

Methods

Public methods:

- `FilterPerformance$new()`
- `FilterPerformance$clone()`

Method `new()`**:** Create a FilterDISR object.

Usage:

```
FilterPerformance$new(
  id = "performance",
  task_type = learner$task_type,
  param_set = learner$param_set,
  feature_types = learner$feature_types,
  learner = mlr3::lrn("classif.rpart"),
  resampling = mlr3::rsmp("holdout"),
  measure = mlr3::msr("classif.ce")
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the filter.

`task_type` (`character()`)
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`feature_types` (`character()`)
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

`learner` (mlr3::Learner)
: mlr3::Learner to use for model fitting.

`resampling` (mlr3::Resampling)
: mlr3::Resampling to be used within resampling.

`measure` (mlr3::Measure)
: mlr3::Measure to be used for evaluating the performance.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterPerformance$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_permutation`, `mlr_filters_variance`, `mlr_filters`

Examples

```
task = mlr3::tsk("iris")
learner = mlr3::lrn("classif.rpart")
filter = flt("performance", learner = learner)
filter$calculate(task)
as.data.table(filter)
```

`mlr_filters_permutation`

Permutation Filter

Description

The permutation filter randomly permutes the values of a single feature in a mlr3::Task to break the association with the response. The permutated feature, together with the unmodified features, is used to perform a `mlr3::resample()`. The permutation filter score is the difference between the aggregated performance of the mlr3::Measure and the performance estimated on the unmodified mlr3::Task.

Parameters

`standardize logical(1)`
: Standardize feature importance by maximum score.

`nmc integer(1)`
: Number of Monte-Carlo iterations to use in computing the feature importance.

Super class

`mlr3filters::Filter` -> `FilterPermutation`

Public fields

`learner` (mlr3::Learner)

`resampling` (mlr3::Resampling)

`measure` (mlr3::Measure)

Methods

Public methods:

- `FilterPermutation$new()`
- `FilterPermutation$clone()`

Method `new()`**:** Create a FilterDISR object.

Usage:

```
FilterPermutation$new(
  id = "permutation",
  task_type = learner$task_type,
  param_set = ParamSet$new(list(ParamLgl$new("standardize", default = FALSE),
    ParamInt$new("nmc", default = 50L))),
  feature_types = learner$feature_types,
  learner = mlr3::lrn("classif.rpart"),
  resampling = mlr3::rsmp("holdout"),
  measure = mlr3::msr("classif.ce")
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the filter.

`task_type` (`character()`)
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`feature_types` (`character()`)
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

`learner` (mlr3::Learner)
: mlr3::Learner to use for model fitting.

`resampling` (mlr3::Resampling)
: mlr3::Resampling to be used within resampling.

`measure` (mlr3::Measure)
: mlr3::Measure to be used for evaluating the performance.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterPermutation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: `Filter`, `mlr_filters_anova`, `mlr_filters_auc`, `mlr_filters_carscore`, `mlr_filters_cmim`, `mlr_filters_correlation`, `mlr_filters_disr`, `mlr_filters_find_correlation`, `mlr_filters_importance`, `mlr_filters_information_gain`, `mlr_filters_jmim`, `mlr_filters_jmi`, `mlr_filters_kruskal_test`, `mlr_filters_mim`, `mlr_filters_mrmr`, `mlr_filters_njmim`, `mlr_filters_performance`, `mlr_filters_variance`, `mlr_filters`

`mlr_filters_variance` *Variance Filter*

Description

Variance filter calling `stats::var()`.

Argument `na.rm` defaults to `TRUE` here.

Super class

`mlr3filters::Filter` -> `FilterVariance`

Methods

Public methods:

- `FilterVariance$new()`
- `FilterVariance$clone()`

Method `new()`**:** Create a FilterVariance object.

Usage:

```
FilterVariance$new(
  id = "variance",
  task_type = c("classif", "regr"),
  param_set = ParamSet$new(list(ParamLgl$new("na.rm", default = TRUE))),
  packages = "stats",
  feature_types = c("integer", "numeric")
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the filter.

`task_type` (`character()`)
: Types of the task the filter can operator on. E.g., `"classif"` or `"regr"`.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`packages` (`character()`)
: Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.

`feature_types` (`character()`)
: Feature types the filter operates on. Must be a subset of `mlr_reflections$task_feature_types`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilterVariance$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Other Filter: Filter, mlr_filters_anova, mlr_filters_auc, mlr_filters_carscore, mlr_filters_cmim, mlr_filters_correlation, mlr_filters_disr, mlr_filters_find_correlation, mlr_filters_importance, mlr_filters_information_gain, mlr_filters_jmim, mlr_filters_jmi, mlr_filters_kruskal_test, mlr_filters_mim, mlr_filters_mrmr, mlr_filters_njmim, mlr_filters_performance, mlr_filters_permutation, mlr_filters

Examples

```
task = mlr3::tsk("mtcars")
filter = flt("variance")
filter$calculate(task)
head(filter$scores, 3)
as.data.table(filter)
```

Index