Package ‘mlr3filters’

November 10, 2020

Title Filter Based Feature Selection for 'mlr3'
Version 0.4.0
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
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 (>= 3.0.0)
Config/testthat/edition 3
Config/testthat/parallel true
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'
  'FilterRelief.R' 'FilterVariance.R' 'flt.R' 'helper.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]
mlr3filters-package

Description

Filter

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 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(1))
  mlr3::Task task properties.

param_set (paradox::ParamSet)
  Set of hyperparameters.

feature_types (character(1))
  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. The more important the feature, the higher the score. 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 de-
  tails), 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_distr, 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_relief, mlr_filters_variance, mlr_filters
**Description**

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

**Usage**

```r
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 `param::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**

```r
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


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 \(-\log_{10}(p)\) 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()

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

**Usage:**
FilterAnova$clone(deep = FALSE)

**Arguments:**
depth 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_reliability,
mlr_filters_variance, mlr_filters

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

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

---

<table>
<thead>
<tr>
<th>mlr_filters_auc</th>
<th>AUC Filter</th>
</tr>
</thead>
</table>

### 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 $\rightarrow$ FilterAUC
Methods

Public methods:

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

Method new(): Create a FilterAUC object.

Usage:
FilterAUC$new()

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


Examples

```r
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()

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

Usage:
FilterCarScore$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

See Also

Dictionary of Filters: 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).
Details

As the scores calculated by the praznik package are not monotone due to the greedy forward fashion, the returned scores simply reflect the selection order: 1, (k−1)/k, ..., 1/k where k is the number of selected features.

Super class

mlr3filters::Filter -> FilterCMIM

Methods

Public methods:

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

Method new(): Create a FilterCMIM object.

Usage:
FilterCMIM$new()

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


Examples

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

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

Usage:

`FilterCorrelation$clone(deep = FALSE)`

Arguments:

deepline Whether to make a deep clone.

See Also

Dictionary of Filters: `mlr_filters`


Examples

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

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

### Description

Double input symmetrical relevance filter calling `praznik::DISR()` from package `praznik`. This filter supports partial scoring (see `Filter`).

### Details

As the scores calculated by the `praznik` package are not monotone due to the greedy forward fashion, the returned scores simply reflect the selection order: 1, (k-1)/k, ..., 1/k where k is the number of selected features.

### Super class

`mlr3filters::Filter` -> `FilterDISR`

### Methods

**Public methods:**

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

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

**Usage:**

```r
FilterDISR$new()
```

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

**Usage:**

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

**Arguments:**

depth Whether to make a deep clone.

### See Also

Dictionary of Filters: `mlr_filters`

mlr_filters_find_correlation

Examples

```r
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()`

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

Examples

```r
## 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)
```

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:

```r
FilterImportance$new(learner = mlr3::lrn("classif.rpart"))
```

Arguments:

- `learner` (mlr3::Learner)

Learner to extract the importance values from.

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

Usage:

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

Arguments:

- `deep` Whether to make a deep clone.
See Also

Dictionary of Filters: mlr_filters

Examples

```r
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()`

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.
**mnr\_filters\_jmi**

**See Also**

Dictionary of Filters: mnr\_filters

Other Filter: Filter, mnr\_filters\_anova, mnr\_filters\_auc, mnr\_filters\_carscore, mnr\_filters\_cmim, mnr\_filters\_correlation, mnr\_filters\_disr, mnr\_filters\_find\_correlation, mnr\_filters\_importance, mnr\_filters\_jmim, mnr\_filters\_jmi, mnr\_filters\_kruskal\_test, mnr\_filters\_mim, mnr\_filters\_mrmr, mnr\_filters\_njmim, mnr\_filters\_performance, mnr\_filters\_permutation, mnr\_filters\_relief, mnr\_filters\_variance, mnr\_filters

**Examples**

```r
## 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)
```

---

**mnr\_filters\_jmi**  
*Joint Mutual Information Filter*

**Description**

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

This filter supports partial scoring (see Filter).

**Details**

As the scores calculated by the praznik package are not monotone due to the greedy forward fashion, the returned scores simply reflect the selection order: 1, (k-1)/k, ..., 1/k where k is the number of selected features.

**Super class**

mlr3filters::Filter -> FilterJMI

**Methods**

**Public methods:**

- FilterJMI$new()
- FilterJMI$clone()
Method `new()`: Create a FilterJMI object.

Usage:
FilterJMI$new()

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`


Examples

```r
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`).

Details

As the scores calculated by the `praznik` package are not monotone due to the greedy forward fashion, the returned scores simply reflect the selection order: 1, (k-1)/k, ..., 1/k where k is the number of selected features.

Super class

`mlr3filters::Filter` -> `FilterJMIM`
**Methods**

**Public methods:**
- `FilterJMIM$new()`
- `FilterJMIM$clone()`

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

**Usage:**
`FilterJMIM$new()`

**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, mlr3filters::Filter`, `mlr3filters::FilterKruskalTest`, `mlr3filters::FilterAnova`, `mlr3filters::FilterAuc`, `mlr3filters::FilterCarscore`, `mlr3filters::FilterCmim`, `mlr3filters::FilterCorrelation`, `mlr3filters::FilterDisr`, `mlr3filters::FilterFindCorrelation`, `mlr3filters::FilterImportance`, `mlr3filters::FilterInformationGain`, `mlr3filters::FilterJMi`, `mlr3filters::FilterKruskalTest`, `mlr3filters::FilterMim`, `mlr3filters::FilterMrMr`, `mlr3filters::FilterNjim`, `mlr3filters::FilterPerformance`, `mlr3filters::FilterPermutation`, `mlr3filters::FilterRelief`, `mlr3filters::FilterVariance`, `mlr3filters::Filter`  

**Examples**

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

---

**mlr3filters::FilterKruskalTest**

*Kruskal-Wallis Test Filter*

**Description**

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

The filter value is $-\log_{10}(p)$ 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()`

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

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

Arguments:
dee p Whether to make a deep clone.

See Also

Dictionary of Filters: `mlr_filters`


Examples

```r
# 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`).
Details

As the scores calculated by the praznik package are not monotone due to the greedy forward fashion, the returned scores simply reflect the selection order: 1, (k−1)/k, ..., 1/k where k is the number of selected features.

Super class

mlr3filters::Filter -> FilterMIM

Methods

Public methods:

• FilterMIM$new()
• FilterMIM$clone()

Method new(): Create a FilterMIM object.

Usage:
FilterMIM$new()

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

Usage:
FilterMIM$clone(deep = FALSE)

Arguments:
deepl Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters


Examples

```r
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::M RM R()` in package `praznik`.

This filter supports partial scoring (see `Filter`).

**Details**

As the scores calculated by the `praznik` package are not monotone due to the greedy forward fashion, the returned scores simply reflect the selection order: 1, (k-1)/k, ..., 1/k where k is the number of selected features.

**Super class**

`mlr3filters::Filter` -> `FilterMRMR`

**Methods**

**Public methods:**

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

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

*Usage:*

`FilterMRMR$new()`

**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`

Examples

```r
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`).

**Details**

As the scores calculated by the `praznik` package are not monotone due to the greedy forward fashion, the returned scores simply reflect the selection order: 1, (k-1)/k, ..., 1/k where k is the number of selected features.

**Super class**

`mlr3filters::Filter` \rightarrow `FilterNJMIM`

**Methods**

**Public methods:**

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

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

*Usage:*

`FilterNJMIM$new()`

**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

Examples

```r
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(
  learner = mlr3::lrn("classif.rpart"),
  resampling = mlr3::rsmp("holdout"),
  measure = NULL
)

Arguments:
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_reliefe.

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 permuted 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(
  learner = mlr3::lrn("classif.rpart"),
  resampling = mlr3::rsmp("holdout"),
  measure = NULL
)

Arguments:
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


---

mlr_filters_relieff  Information Gain Filter

Description

Information gain filter calling FSelectorRcpp::relief() in package FSelectorRcpp.

Super class

mlr3filters::Filter -> FilterRelief

Methods

Public methods:

- FilterRelief$new()
- FilterRelief$clone()

Method new(): Create a FilterRelief object.

Usage:

FilterRelief$new()

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

Usage:

FilterRelief$clone(deep = FALSE)

Arguments:

deept Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Examples

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

```r
FilterVariance$new()
```

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

**Usage:**

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

**Arguments:**

- `deep` Whether to make a deep clone.

**See Also**

Dictionary of Filters: `mlr_filters`

Other Filter: `Filter, mlr3_filters_anova, mlr3_filters_auc, mlr3_filters_carscore, mlr3_filters_cmim, mlr3_filters_correlation, mlr3_filters_disr, mlr3_filters_find_correlation, mlr3_filters_importance, mlr3_filters_information_gain, mlr3_filters_jmim, mlr3_filters_jmi, mlr3_filters_kruskal_test, mlr3_filters_mim, mlr3_filters_mrmr, mlr3_filters_njimim, mlr3_filters_performance, mlr3_filters_permutation, mlr3_filters_relief, mlr3_filters`
Examples

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

* Dictionary
  - mlr_filters, 6

* Filter
  - Filter, 3
  - Filter (mlr_filters), 6
  - FilterAnova (mlr_filters_anova), 7
  - FilterAUC (mlr_filters_auc), 8
  - FilterCarScore (mlr_filters_carscore), 9
  - FilterCMIM (mlr_filters_cmim), 10
  - FilterCorrelation (mlr_filters_correlation), 12
  - FilterDISR (mlr_filters_disr), 13
  - FilterFindCorrelation (mlr_filters_find_correlation), 14
  - FilterImportance (mlr_filters_importance), 15
  - FilterInformationGain (mlr_filters_information_gain), 16
  - FilterJMI (mlr_filters_jmi), 17
  - FilterJMIM (mlr_filters_jmim), 18
  - FilterKruskalTest (mlr_filters_kruskal_test), 19
  - FilterMIM (mlr_filters_mim), 20
  - FilterMRMR (mlr_filters_mrmr), 22
  - FilterNJMIM (mlr_filters_njmim), 23
  - FilterPerformance (mlr_filters_performance), 24
  - FilterPermutation (mlr_filters_permutation), 25
  - FilterRelief (mlr_filters_relief), 27
  - FilterVariance (mlr_filters_variance), 28

* datasets
  - mlr_filters, 6

- care::carscore(), 9
- caret::findCorrelation(), 14
- character(), 4

Dictionary, 6–14, 16–22, 24, 25, 27, 28
dictionary, 3, 6

Filter, 3, 6–14, 16–25, 27, 28
FilterAnova (mlr_filters_anova), 7
FilterAUC (mlr_filters_auc), 8
FilterCarScore (mlr_filters_carscore), 9
FilterCMIM (mlr_filters_cmim), 10
FilterDISR (mlr_filters_disr), 13
FilterFindCorrelation (mlr_filters_find_correlation), 14
FilterImportance (mlr_filters_importance), 15
FilterInformationGain (mlr_filters_information_gain), 16
FilterJMI (mlr_filters_jmi), 17
FilterJMIM (mlr_filters_jmim), 18
FilterKruskalTest (mlr_filters_kruskal_test), 19
FilterMIM (mlr_filters_mim), 20
FilterMRMR (mlr_filters_mrmr), 22
FilterNJMIM (mlr_filters_njmim), 23
FilterPerformance (mlr_filters_performance), 24
FilterPermutation (mlr_filters_permutation), 25
FilterRelief (mlr_filters_relief), 27
Filters, 8–14, 16–22, 24, 25, 27, 28
FilterVariance (mlr_filters_variance), 28

flt, 6
flt(), 6
flts(flt), 6
FSelectorRcpp::information_gain(), 16
FSelectorRcpp::relief(), 27

integer(), 5

mlr3::Learner, 15, 24–26
mlr3::Measure, 24–26
mlr3::mlr_sugar, 6
mlr3::resample(), 24, 25
mlr3::Resampling, 24–26
mlr3::Task, 3–5, 25