Package ‘mlr3filters’

March 12, 2020

Title Filter Based Feature Selection for ‘mlr3’

Version 0.2.0

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 (>= 0.1.2), mlr3misc, paradox, R6

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

Encoding UTF-8

NeedsCompilation no

RoxygenNote 7.0.2

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' '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 2020-03-12 12:40:13 UTC
**R topics documented:**

mlr3filters-package .................................................. 2
Filter ................................................................. 3
flt ................................................................. 5
mlr_filters .......................................................... 6
mlr_filters_anova .................................................... 7
mlr_filters_auc ....................................................... 8
mlr_filters_carscore ............................................... 10
mlr_filters_cmim ..................................................... 11
mlr_filters_correlation ............................................. 13
mlr_filters_disr ..................................................... 14
mlr_filters_find_correlation ..................................... 16
mlr_filters_information_gain .................................. 17
mlr_filters_jmi ...................................................... 19
mlr_filters_jmim .................................................... 20
mlr_filters_kruskal_test ........................................ 22
mlr_filters_mim ..................................................... 23
mlr_filters_mrmr ................................................... 25
mlr_filters_njmim .................................................. 26
mlr_filters_performance ......................................... 27
mlr_filters_variable_importance .............................. 29
mlr_filters_variance .............................................. 31

**Index** ........................................................................... 33

---

**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>
Filter

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())

mlr3::Task task 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.

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$print()
• Filter$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()
)

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 format(): Format helper for Filter class

Usage:
Filter$format()

Method print(): Printer for Filter class

Usage:
Filter$print()

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_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_variable_importance, mlr_filters_variance,
mlr_filters

---

**flt** Syntactic Sugar for Filter Construction

**Description**

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

**Usage**

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

**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 `para-
dox::ParamSet`, or to be set as public field. See `mlr3misc::dictionary_sugar_get()` for more details.

**Value**

Filter.
Examples

```r
flt("correlation", method = "kendall")
```

---

**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**

```r
mlr_filters
```

**Format**

R6Class object

**Usage**

See Dictionary.

**See Also**


**Examples**

```r
mlr_filters$keys()
as.data.table(mlr_filters)
mlr_filters$get("mim")
flt("anova")
```
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 \(\rightarrow\) 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:
deepl Whether to make a deep clone.

See Also

Dictionary of Filters: 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


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

```r
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:
deepl Whether to make a deep clone.

See Also

Dictionary of Filters: mlr_filters

Examples

```r
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` \rightarrow `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


Examples

  task = mlr3::tsk("iris")
  filter = flt("cmim")
  filter$calculate(task, nfeat = 2)
  as.data.table(filter)
**mlr3filters::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:**

```r
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(1))
  - Types of the task the filter can operator on. E.g., "classif" or "regr".
- `param_set` (`paradox::ParamSet`)
  - Set of hyperparameters.
- `packages` (character(1))
  - Set of required packages. Note that these packages will be loaded via `requireNamespace()`, and are not attached.
- `feature_types` (character(1))
  - 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:
depth Whether to make a deep clone.

See Also
Dictionary of Filters: mlr_filters

Examples

## Pearson (default)
```r
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)
```

---

### 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_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_variable_importance, mlr_filters_variance,
  mlr_filters

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

```r
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


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_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:
depth 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_jmim,
mlr_filters_jmi, mlr_filters_kruskal_test, mlr_filters_mim, mlr_filters_mrmr, mlr_filters_njmim,
mlr_filters_performance, mlr_filters_variable_importance, mlr_filters_variance, mlr_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)
```

---

**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`\rightarrow FilterJMI

Methods

Public methods:

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

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

Usage:

```r
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

Examples

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

---

### 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


Examples

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

Kruskal-Wallis rank sum test filter calling \texttt{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

\texttt{mlr3filters::Filter} -> FilterKruskalTest

Methods

Public methods:

- \texttt{FilterKruskalTest$\texttt{new}()}
- \texttt{FilterKruskalTest$\texttt{clone}()}

Method \texttt{new}(): Create a FilterKruskalTest object.

Usage:

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

Arguments:

- \texttt{id} (character(1))
  - Identifier for the filter.
- \texttt{task_type} (character())
  - Types of the task the filter can operator on. E.g., "classif" or "regr".
- \texttt{param_set} (paradox::ParamSet)
  - Set of hyperparameters.
- \texttt{packages} (character())
  - Set of required packages. Note that these packages will be loaded via \texttt{requireNamespace()}, and are not attached.
- \texttt{feature_types} (character())
  - Feature types the filter operates on. Must be a subset of \texttt{mlr_reflections$task_feature_types}.

Method \texttt{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


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:

deeep 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_cmicm,
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_mrmr, mlr_filters_njmim,
mlr_filters_performance, mlr_filters_variable_importance, 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


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`).

**Super class**

`mlr3filters::Filter` -> `FilterNJMIM`

**Methods**

**Public methods:**

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

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

**Usage:**

```r
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 operate 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_information_gain,
   mlr_filters_jnim, mlr_filters_jmi, mlr_filters_kruskal_test, mlr_filters_mim, mlr_filters_mrmr,
   mlr_filters_performance, mlr_filters_variable_importance, mlr_filters_variance, mlr_filters

Examples

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

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.
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:
FilterPerformance$clone(deep = FALSE)

Arguments:
depth 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_variable_importance, 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_variable_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 operate 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

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

Variance filter calling \texttt{stats\::var()}. Argument \texttt{na.rm} defaults to \texttt{TRUE} here.

Super class

\texttt{mlr3filters\::Filter -> FilterVariance}

Methods

Public methods:

\begin{itemize}
  \item \texttt{FilterVariance\$new()}
  \item \texttt{FilterVariance\$clone()}
\end{itemize}

Method \texttt{new()}: Create a FilterVariance object.

Usage:

\begin{verbatim}
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")
)
\end{verbatim}

Arguments:

\begin{itemize}
  \item \texttt{id} (character(1))
    \hspace{1em} Identifier for the filter.
  \item \texttt{task_type} (character())
    \hspace{1em} Types of the task the filter can operator on. E.g., "classif" or "regr".
  \item \texttt{param_set} (\texttt{paradox\::ParamSet})
    \hspace{1em} Set of hyperparameters.
  \item \texttt{packages} (character())
    \hspace{1em} Set of required packages. Note that these packages will be loaded via \texttt{requireNamespace()}, and are not attached.
  \item \texttt{feature_types} (character())
    \hspace{1em} Feature types the filter operates on. Must be a subset of \texttt{mlr_reflections\$task_feature_types}.
\end{itemize}

Method \texttt{clone()}: The objects of this class are cloneable with this method.

Usage:

\begin{verbatim}
FilterVariance\$clone(deep = FALSE)
\end{verbatim}

Arguments:

\begin{itemize}
  \item \texttt{deep} \hspace{1em} Whether to make a deep clone.
\end{itemize}
See Also

Dictionary of Filters: mlr_filters


Examples

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

*Topic **datasets**
  mlr_filters, 6

  care::carscore(), 10
caret::findCorrelation(), 16
character(), 3

dictionary, 6, 8, 9, 11, 12, 14, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32
dictionary, 3, 5

Filter, 3, 5, 6, 8, 9, 11, 12, 14, 15, 17–21, 23–27, 29, 30, 32
FilterAnova (mlr_filters_anova), 7
FilterAUC (mlr_filters_auc), 8
FilterCarScore (mlr_filters_carscore), 10
FilterCMIM (mlr_filters_cmim), 11
FilterCorrelation
  (mlr_filters_correlation), 13
FilterDISR (mlr_filters_disr), 14
FilterFindCorrelation
  (mlr_filters_find_correlation), 16
FilterImportance
  (mlr_filters_variable_importance), 29
FilterInformationGain
  (mlr_filters_information_gain), 17
FilterJMI (mlr_filters_jmi), 19
FilterJMIM (mlr_filters_jmim), 20
FilterKruskalTest
  (mlr_filters_kruskal_test), 22
FilterMIM (mlr_filters_mim), 23
FilterMRMR (mlr_filters_mrmar), 25
FilterNJMIM (mlr_filters_njmim), 26
FilterPerformance
  (mlr_filters_performance), 27

Filters, 8, 9, 11, 12, 14, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32
FilterVariance (mlr_filters_variance), 31
flt, 5
flt(), 6
FSelectorRcpp::information_gain(), 17

integer(), 5

mlr3::Learner, 27–30
mlr3::Measure, 27, 28
mlr3::mlr_sugar, 5
mlr3::resample(), 27
mlr3::Resampling, 28
mlr3::Task, 3–5, 7, 9
mlr3filters (mlr3filters-package), 2
mlr3filters-package, 2
mlr3filters::Filter, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20, 22, 23, 25–27, 29, 31
mlr3measures::auc(), 8
mlr3misc::dictionary_sugar_get(), 5
mlr3filters, 3, 5, 6, 8, 9, 11, 12, 14, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32
mlr3filters_anova, 5, 6, 7, 9, 11, 12, 14, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32
mlr3filters_auc, 5, 6, 8, 11, 12, 14, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32
mlr3filters_carscore, 5, 6, 8, 9, 10, 12, 14, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32
mlr3filters_cmim, 5, 6, 8, 9, 11, 14, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32
mlr3filters_correlation, 5, 6, 8, 9, 11, 12, 13, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32