Package 'mlr3fselect'

January 24, 2021

Title Feature Selection for 'mlr3'

Version 0.5.0

Description Implements methods for feature selection with 'mlr3', e.g. random search and sequential selection. Various termination criteria can be set and combined. The class 'AutoFSelector' provides a convenient way to perform nested resampling in combination with 'mlr3'.

License LGPL-3

URL https://mlr3fselect.mlr-org.com, https://github.com/mlr-org/mlr3fselect

BugReports https://github.com/mlr-org/mlr3fselect/issues

Depends R (>= 3.1.0)

Imports bbotk (>= 0.3.0), checkmate (>= 2.0.0), data.table, lgr, mlr3 (>= 0.7.0), mlr3misc (>= 0.7.0), mlr3pipelines (>= 0.3.0), paradox (>= 0.7.0), R6

Suggests genalg, rpart, testthat (>= 3.0.0)

Encoding UTF-8

NeedsCompilation no

Config/testthat/edition 3

Config/testthat/parallel true

RoxygenNote 7.1.1

Collate 'AutoFSelector.R' 'ArchiveFSelect.R' 'ObjectiveFSelect.R' 'mlr_fselectors.R' 'FSelector.R' 'FSelectorFromOptimizer.R' 'FSelectorExhaustiveSearch.R' 'FSelectorRFE.R' 'FSelectorRandomSearch.R' 'FSelectorSequential.R' 'FSelectorDesignPoints.R' 'FSelectorGeneticSearch.R' 'FSelectInstanceMultiCrit.R' 'FSelectInstanceSingleCrit.R' 'reexports.R' 'sugar.R' 'bibentries.R' 'zzz.R'

Author Marc Becker [aut, cre] (<https://orcid.org/0000-0002-8115-0400>), Patrick Schratz [aut] (<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>)

Maintainer Marc Becker <marcbecker@posteo.de>

Repository CRAN

Date/Publication 2021-01-24 14:50:03 UTC

R topics documented:

`mlr3fselect-package` *mlr3fselect: Feature Selection for 'mlr3'*

Description

Implements methods for feature selection with 'mlr3', e.g. random search and sequential selection. Various termination criteria can be set and combined. The class 'AutoFSelector' provides a convenient way to perform nested resampling in combination with 'mlr3'.

Author(s)

Maintainer: Marc Becker <marcbecker@posteo.de> (ORCID)

Authors:

- Patrick Schratz <patrick.schratz@gmail.com> (ORCID)
- Michel Lang <michellang@gmail.com> (ORCID)
- Bernd Bischl <bernd_bischl@gmx.net> (ORCID)

See Also

Useful links:

- https://mlr3fselect.mlr-org.com
- https://github.com/mlr-org/mlr3fselect
- Report bugs at https://github.com/mlr-org/mlr3fselect/issues

---

`ArchiveFSelect` *Logging object for objective function evaluations*

Description

Container around a data.table::data.table which stores all performed function calls of the Objective and the associated mlr3::BenchmarkResult.
$benchmark_result stores a mlr3::BenchmarkResult which contains the mlr3::ResampleResult of all performed function calls. The mlr3::BenchmarkResult is connected to the data.table::data.table via the `uhash` column.

Super class

`bbotk::Archive` -> `ArchiveFSelect`

Public fields

`benchmark_result` (mlr3::BenchmarkResult)
Stores benchmark result.

Methods

Public methods:

- `ArchiveFSelect$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

AutoFSelector *AutoFSelector*

Description

The AutoFSelector is a mlr3::Learner which wraps another mlr3::Learner and performs the following steps during $train():

1. The wrapped (inner) learner is trained on the feature subsets via resampling. The feature selection can be specified by providing a FSelector, a bbotk::Terminator, a mlr3::Resampling and a mlr3::Measure.
2. A final model is fit on the complete training data with the best found feature subset.

During $predict() the AutoFSelector just calls the predict method of the wrapped (inner) learner.

Note that this approach allows to perform nested resampling by passing an AutoFSelector object to `mlr3::resample()` or `mlr3::benchmark()`. To access the inner resampling results, set `store_fselect_instance = TRUE` and execute `mlr3::resample()` or `mlr3::benchmark()` with `store_models = TRUE`.

Super class

`mlr3::Learner` -> AutoFSelector

Public fields

`instance_args` (`list()`)
: All arguments from construction to create the FSelectInstanceSingleCrit.

`fselector` (FSelector)
: Stores the feature selection algorithm.

Active bindings

`archive` ([ArchiveFSelect)
: Returns FSelectInstanceSingleCrit archive.

`learner` (mlr3::Learner)
: Trained learner.

`fselect_instance` (FSelectInstanceSingleCrit)
: Internally created feature selection instance with all intermediate results.

`fselect_result` (named `list()`)
: Short-cut to $result from FSelectInstanceSingleCrit.

Methods

Public methods:

- AutoFSelector$new()
- AutoFSelector$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
AutoFSelector$new(
  learner,
  resampling,
  measure,
  terminator,
  fselector,
  store_fselect_instance = TRUE,
  store_benchmark_result = TRUE,
  store_models = FALSE,
  check_values = FALSE
)
```

Arguments:

`learner` (mlr3::Learner)
: Learner to optimize the feature subset for, see FSelectInstanceSingleCrit.

`resampling` (mlr3::Resampling)
: Resampling strategy during feature selection, see FSelectInstanceSingleCrit. This mlr3::Resampling is meant to be the **inner** resampling, operating on the training set of an arbitrary outer resampling. For this reason it is not feasible to pass an instantiated mlr3::Resampling here.

`measure` (mlr3::Measure)
: Performance measure to optimize.

`terminator` (bbotk::Terminator)
: When to stop feature selection, see FSelectInstanceSingleCrit.

`fselector` (FSelector)
: Feature selection algorithm to run.

`store_fselect_instance (logical(1))`
: If `TRUE` (default), stores the internally created FSelectInstanceSingleCrit with all intermediate results in slot $fselect_instance.

`store_benchmark_result (logical(1))`
: Store benchmark result in archive?

`store_models (logical(1))`. Store models in benchmark result?

`check_values (logical(1))`
: Check the parameters before the evaluation and the results for validity?

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3)

task = tsk("iris")
learner = lrn("classif.rpart")
resampling = rsmp("holdout")
measure = msr("classif.ce")

terminator = trm("evals", n_evals = 3)
fselector = fs("exhaustive_search")
afs = AutoFSelector$new(learner, resampling, measure, terminator, fselector,
  store_fselect_instance = TRUE)

afs$train(task)
afs$model
afs$learner
```

`fs` *Syntactic Sugar for FSelect Construction*

Description

This function complements mlr_fselectors with functions in the spirit of mlr3::mlr_sugar.

Usage

```
fs(.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 paradox::ParamSet, or to be set as public field. See `mlr3misc::dictionary_sugar_get()` for more details.

Value

FSelector.

Examples

```
fs("sequential", max_features = 4)
```

FSelectInstanceMultiCrit

Multi Criterion Feature Selection Instance

Description

Specifies a general feature selection scenario, including objective function and archive for feature selection algorithms to act upon. This class stores an ObjectiveFSelect object that encodes the black box objective function which an FSelector has to optimize. It allows the basic operations of querying the objective at feature subsets ($eval_batch()), storing the evaluations in the internal bbotk::Archive and accessing the final result ($result).

Evaluations of feature subsets are performed in batches by calling `mlr3::benchmark()` internally. Before a batch is evaluated, the bbotk::Terminator is queried for the remaining budget. If the available budget is exhausted, an exception is raised, and no further evaluations can be performed from this point on.

The FSelector is also supposed to store its final result, consisting of the selected feature subsets and associated estimated performance values, by calling the method `instance$assign_result()`.

Super classes

`bbotk::OptimInstance` -> `bbotk::OptimInstanceMultiCrit` -> `FSelectInstanceMultiCrit`

Active bindings

`result_feature_set` (`list()` of `character()`)
Feature sets for task subsetting.

Methods

Public methods:

- `FSelectInstanceMultiCrit$new()`
- `FSelectInstanceMultiCrit$assign_result()`
- `FSelectInstanceMultiCrit$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
FSelectInstanceMultiCrit$new(
  task,
  learner,
  resampling,
  measures,
  terminator,
  store_models = FALSE,
  check_values = TRUE,
  store_benchmark_result = TRUE
)
```

Arguments:

`task` (mlr3::Task)
Task to operate on.

`learner` (mlr3::Learner).

`resampling` (mlr3::Resampling)
Uninstantiated resamplings are instantiated during construction so that all configurations are evaluated on the same data splits.

`measures` (list of mlr3::Measure)
Measures to optimize. If `NULL`, **mlr3**'s default measure is used.

`terminator` (bbotk::Terminator).

`store_models` (`logical(1)`). Store models in benchmark result?

`check_values` (`logical(1)`)
Check the parameters before the evaluation and the results for validity?

`store_benchmark_result` (`logical(1)`)
Store benchmark result in archive?

Method `assign_result()`**:** The FSelector object writes the best found feature subsets and estimated performance values here. For internal use.

Usage:

```
FSelectInstanceMultiCrit$assign_result(xdt, ydt)
```

Arguments:

`xdt` (`data.table::data.table()`)
x values as `data.table`. Each row is one point. Contains the value in the *search space* of the FSelectInstanceMultiCrit object. Can contain additional columns for extra information.

`ydt` (`data.table::data.table()`)
Optimal outcomes, e.g. the Pareto front.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3)
library(data.table)

# Objects required to define the performance evaluator
task = tsk("iris")
measures = msrs(c("classif.ce", "classif.acc"))
learner = lrn("classif.rpart")
resampling = rsmp("cv")
terminator = trm("evals", n_evals = 8)

inst = FSelectInstanceMultiCrit$new(
```

```
  task = task,
  learner = learner,
  resampling = resampling,
  measures = measures,
  terminator = terminator
)

# Try some feature subsets
xdt = data.table(
  Petal.Length = c(TRUE, FALSE),
  Petal.Width = c(FALSE, TRUE),
  Sepal.Length = c(TRUE, FALSE),
  Sepal.Width = c(FALSE, TRUE)
)

inst$eval_batch(xdt)

# Get archive data
as.data.table(inst$archive)
```

`FSelectInstanceSingleCrit`

Single Criterion Feature Selection Instance

Description

Specifies a general feature selection scenario, including objective function and archive for feature selection algorithms to act upon. This class stores an ObjectiveFSelect object that encodes the black box objective function which an FSelector has to optimize. It allows the basic operations of querying the objective at feature subsets ($eval_batch()), storing the evaluations in the internal bbotk::Archive and accessing the final result ($result).

Evaluations of feature subsets are performed in batches by calling `mlr3::benchmark()` internally. Before a batch is evaluated, the bbotk::Terminator is queried for the remaining budget. If the available budget is exhausted, an exception is raised, and no further evaluations can be performed from this point on.

The FSelector is also supposed to store its final result, consisting of a selected feature subset and associated estimated performance values, by calling the method `instance$assign_result()`.

Super classes

`bbotk::OptimInstance` -> `bbotk::OptimInstanceSingleCrit` -> `FSelectInstanceSingleCrit`

Active bindings

`result_feature_set (character())`
: Feature set for task subsetting.

Methods

Public methods:

- FSelectInstanceSingleCrit$new()
- FSelectInstanceSingleCrit$assign_result()
- FSelectInstanceSingleCrit$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
FSelectInstanceSingleCrit$new(
  task,
  learner,
  resampling,
  measure,
  terminator,
  store_models = FALSE,
  check_values = TRUE,
  store_benchmark_result = TRUE
)
```

Arguments:

`task` (mlr3::Task)
: Task to operate on.

`learner` (mlr3::Learner).

`resampling` (mlr3::Resampling)
: Uninstantiated resamplings are instantiated during construction so that all configurations are evaluated on the same data splits.

`measure` (mlr3::Measure)
: Measure to optimize.

`terminator` (bbotk::Terminator).

`store_models` (`logical(1)`). Store models in benchmark result?

`check_values` (`logical(1)`)
: Check the parameters before the evaluation and the results for validity?

`store_benchmark_result` (`logical(1)`)
: Store benchmark result in archive?

Method `assign_result()`**:** The FSelector writes the best found feature subset and estimated performance value here. For internal use.

Usage:

```
FSelectInstanceSingleCrit$assign_result(xdt, y)
```

Arguments:

`xdt` (`data.table::data.table()`)
: x values as `data.table`. Each row is one point. Contains the value in the *search space* of the FSelectInstanceMultiCrit object. Can contain additional columns for extra information.

`y` (`numeric(1)`)
: Optimal outcome.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3)
library(data.table)

# Objects required to define the objective function
task = tsk("iris")
measure = msr("classif.ce")
learner = lrn("classif.rpart")
resampling = rsmp("cv")

# Create instance
terminator = trm("evals", n_evals = 8)
inst = FSelectInstanceSingleCrit$new(
  task = task,
  learner = learner,
  resampling = resampling,
  measure = measure,
  terminator = terminator
)

# Try some feature subsets
xdt = data.table(
  Petal.Length = c(TRUE, FALSE),
  Petal.Width = c(FALSE, TRUE),
  Sepal.Length = c(TRUE, FALSE),
  Sepal.Width = c(FALSE, TRUE)
)

inst$eval_batch(xdt)

# Get archive data
as.data.table(inst$archive)
```

`FSelector` *FSelector*

Description

Abstract `FSelector` class that implements the base functionality each fselector must provide. A `FSelector` object describes the feature selection strategy, i.e. how to optimize the black-box function and its feasible set defined by the FSelectInstanceSingleCrit / FSelectInstanceMultiCrit object.

A fselector must write its result into the FSelectInstanceSingleCrit / FSelectInstanceMultiCrit using the `assign_result` method of the bbotk::OptimInstance at the end of its selection in order to store the best selected feature subset and its estimated performance vector.

Private Methods

- `.optimize(instance)` -> `NULL`
  Abstract base method. Implement to specify feature selection of your subclass. See technical details sections.
- `.assign_result(instance)` -> `NULL`
  Abstract base method. Implement to specify how the final feature subset is selected. See technical details sections.

Technical Details and Subclasses

A subclass is implemented in the following way:

- Inherit from `FSelector`.
- Specify the private abstract method $.optimize() and use it to call into your optimizer.
- You need to call `instance$eval_batch()` to evaluate feature subsets.
- The batch evaluation is requested at the FSelectInstanceSingleCrit / FSelectInstanceMultiCrit object `instance`, so each batch is possibly executed in parallel via `mlr3::benchmark()`, and all evaluations are stored inside of `instance$archive`.
- Before the batch evaluation, the bbotk::Terminator is checked, and if it is positive, an exception of class `"terminated_error"` is generated. In the later case the current batch of evaluations is still stored in `instance`, but the numeric scores are not sent back to the handling optimizer as it has lost execution control.
- After such an exception was caught we select the best feature subset from `instance$archive` and return it.
- Note that therefore more points than specified by the bbotk::Terminator may be evaluated, as the Terminator is only checked before a batch evaluation, and not in-between evaluation in a batch. How many more depends on the setting of the batch size.
- Overwrite the private super-method `.assign_result()` if you want to decide yourself how to estimate the final feature subset in the instance and its estimated performance. The default behavior is: We pick the best resample-experiment, regarding the given measure, then assign its feature subset and aggregated performance to the instance.

Public fields

`param_set` (paradox::ParamSet).

`param_classes` (`character()`).

`properties` (`character()`).

`packages` (`character()`).

Methods

Public methods:

- FSelector$new()
- FSelector$format()
- FSelector$print()
- FSelector$optimize()
- FSelector$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
FSelector$new(param_set, properties, packages = character(0))
```

Arguments:

`param_set` paradox::ParamSet
Set of control parameters for fselector.

`properties` (`character()`)
Set of properties of the fselector. Must be a subset of `mlr_reflections$fselect_properties`.

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

Method `format()`**:** Helper for print outputs.

Usage:

```
FSelector$format()
```

Returns: (`character()`).

Method `print()`**:** Print method.

Usage:

```
FSelector$print()
```

Returns: (`character()`).

Method `optimize()`**:** Performs the feature selection on a FSelectInstanceSingleCrit or FSelectInstanceMultiCrit until termination. The single evaluations will be written into the ArchiveFSelect that resides in the FSelectInstanceSingleCrit / FSelectInstanceMultiCrit. The result will be written into the instance object.

Usage:

```
FSelector$optimize(inst)
```

Arguments:

`inst` (FSelectInstanceSingleCrit|FSelectInstanceMultiCrit).

Returns: data.table::data.table.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3)

terminator = trm("evals", n_evals = 3)

instance = FSelectInstanceSingleCrit$new(
  task = tsk("iris"),
  learner = lrn("classif.rpart"),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  terminator = terminator
)

# swap this line to use a different FSelector
fselector = fs("random_search")

# modifies the instance by reference
fselector$optimize(instance)

# returns best feature subset and best performance
instance$result

# allows access of data.table / benchmark result of full path of all evaluations
instance$archive
```

FSelectorDesignPoints *Feature Selection via Design Points*

Description

FSelectorDesignPoints class that implements feature selection w.r.t. fixed feature sets. We simply search over a set of feature subsets fully specified by the user. The feature sets are evaluated in order as given.

In order to support general termination criteria and parallelization, we evaluate feature sets in a batch-fashion of size batch_size. Larger batches mean we can parallelize more, smaller batches imply a more fine-grained checking of termination criteria.

Dictionary

This FSelector can be instantiated via the dictionary mlr_fselectors or with the associated sugar function fs():

```
mlr_fselectors$get("design_points")
fs("design_points")
```

Parameters

`batch_size integer(1)`
: Maximum number of configurations to try in a batch.

`design` data.table::data.table
: Design points to try in search, one per row.

Super classes

`mlr3fselect::FSelector` -> `mlr3fselect::FSelectorFromOptimizer` -> `FSelectorDesignPoints`

Methods

Public methods:

- `FSelectorDesignPoints$new()`
- `FSelectorDesignPoints$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
FSelectorDesignPoints$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3)
library(data.table)

terminator = trm("evals", n_evals = 10)

instance = FSelectInstanceSingleCrit$new(
  task = tsk("iris"),
  learner = lrn("classif.rpart"),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  terminator = terminator
)

design = data.table(Petal.Length = c(TRUE, FALSE),
  Petal.Width = c(TRUE, FALSE),
  Sepal.Length = c(FALSE, TRUE),
  Sepal.Width = c(FALSE, TRUE))

fselector = fs("design_points", design = design)
```

```
# Modifies the instance by reference
fselector$optimize(instance)

# Returns best scoring evaluation
instance$result

# Allows access of data.table of full path of all evaluations
as.data.table(instance$archive)
```

`FSelectorExhaustiveSearch`

Feature Selection via Exhaustive Search

Description

`FSelectorExhaustiveSearch` class that implements an Exhaustive Search.

In order to support general termination criteria and parallelization, feature sets are evaluated in batches. The size of the feature sets is increased by 1 in each batch.

Dictionary

This FSelector can be instantiated via the dictionary mlr_fselectors or with the associated sugar function `fs()`:

```
mlr_fselectors$get("exhaustive_search")
fs("exhaustive_search")
```

Parameters

`max_features integer(1)`
: Maximum number of features. By default, number of features in mlr3::Task.

Super class

`mlr3fselect::FSelector` -> `FSelectorExhaustiveSearch`

Methods

Public methods:

- `FSelectorExhaustiveSearch$new()`
- `FSelectorExhaustiveSearch$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
FSelectorExhaustiveSearch$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
library(mlr3)

terminator = trm("evals", n_evals = 5)

instance = FSelectInstanceSingleCrit$new(
  task = tsk("iris"),
  learner = lrn("classif.rpart"),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  terminator = terminator
)

fselector = fs("exhaustive_search")

# Modifies the instance by reference
fselector$optimize(instance)

# Returns best scoring evaluation
instance$result

# Allows access of data.table of full path of all evaluations
as.data.table(instance$archive)
```

FSelectorGeneticSearch

Feature Selection via Genetic Search

Description

`FSelectorGeneticSearch` class that implements an Genetic Search. Calls `genalg::rbga.bin()` from package **genalg**.

Dictionary

This FSelector can be instantiated via the dictionary mlr_fselectors or with the associated sugar function `fs()`:

```
mlr_fselectors$get("genetic_search")
fs("genetic_search")
```

Parameters

`suggestions list()`

`popSize integer(1)`

`mutationChance numeric(1)`

`elitism integer(1)`

`zeroToOneRatio integer(1)`

`iters integer(1)`

For the meaning of the control parameters, see `genalg::rbga.bin()`. `genalg::rbga.bin()` internally terminates after `iters` iteration. We set `iters = 100000` to allow the termination via our terminators. If more iterations are needed, set `iters` to a higher value in the parameter set.

Super class

`mlr3fselect::FSelector` -> `FSelectorGeneticSearch`

Methods

Public methods:

- `FSelectorGeneticSearch$new()`
- `FSelectorGeneticSearch$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
FSelectorGeneticSearch$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
library(mlr3)

terminator = trm("evals", n_evals = 5)

instance = FSelectInstanceSingleCrit$new(
  task = tsk("iris"),
  learner = lrn("classif.rpart"),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  terminator = terminator
)

fselector = fs("genetic_search", popSize = 10L)
```

```
# Modifies the instance by reference
fselector$optimize(instance)

# Returns best scoring evaluation
instance$result

# Allows access of data.table of full path of all evaluations
as.data.table(instance$archive)
```

`FSelectorRandomSearch` *Feature Selection via Random Search*

Description

`FSelectorRandomSearch` class that implements a simple Random Search.

In order to support general termination criteria and parallelization, we evaluate feature sets in a batch-fashion of size `batch_size`. Larger batches mean we can parallelize more, smaller batches imply a more fine-grained checking of termination criteria.

Dictionary

This FSelector can be instantiated via the dictionary mlr_fselectors or with the associated sugar function `fs()`:

```
mlr_fselectors$get("random_search")
fs("random_search")
```

Parameters

`max_features integer(1)`
: Maximum number of features. By default, number of features in mlr3::Task.

`batch_size integer(1)`
: Maximum number of feature sets to try in a batch.

Super class

`mlr3fselect::FSelector` -> `FSelectorRandomSearch`

Methods

Public methods:

- `FSelectorRandomSearch$new()`
- `FSelectorRandomSearch$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
FSelectorRandomSearch$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Source

Bergstra J, Bengio Y (2012). "Random Search for Hyper-Parameter Optimization." *Journal of Machine Learning Research*, **13**(10), 281–305. https://jmlr.csail.mit.edu/papers/v13/bergstra12a.html.

Examples

```
library(mlr3)

terminator = trm("evals", n_evals = 5)

instance = FSelectInstanceSingleCrit$new(
  task = tsk("iris"),
  learner = lrn("classif.rpart"),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  terminator = terminator
)

fselector = fs("random_search")

# Modifies the instance by reference
fselector$optimize(instance)

# Returns best scoring evaluation
instance$result

# Allows access of data.table of full path of all evaluations
as.data.table(instance$archive)
```

FSelectorRFE *Feature Selection via Recursive Feature Elimination*

Description

`FSelectorRFE` class that implements Recursive Feature Elimination (RFE). The recursive algorithm (`recursive = TRUE`) recomputes the feature importance on the reduced feature set in every iteration. The non-recursive algorithm (`recursive = FALSE`) only uses the feature importance of the model fitted with all features to eliminate the next most unimportant features in every iteration.

Dictionary

This FSelector can be instantiated via the dictionary mlr_fselectors or with the associated sugar function `fs()`:

```
mlr_fselectors$get("rfe")
fs("rfe")
```

Parameters

`min_features integer(1)`
: The minimum number of features to select, default is `1L`.

`feature_fraction double(1)`
: Fraction of features to retain in each iteration, default is `0.5`.

`feature_number integer(1)`
: Number of features to remove in each iteration.

`subset_sizes integer()`
: Vector of number of features to retain in each iteration. Must be sorted in decreasing order.

`recursive logical(1)`
: Use the recursive version? Default is `FALSE`.

The parameter `feature_fraction`, `feature_number` and `subset_sizes` are mutually exclusive.

Super class

`mlr3fselect::FSelector` -> `FSelectorRFE`

Public fields

`importance numeric()`
: Stores the feature importance of the model with all variables if `recrusive` is set to `FALSE`

Methods

Public methods:

- `FSelectorRFE$new()`
- `FSelectorRFE$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
FSelectorRFE$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3)

terminator = trm("evals", n_evals = 10)
instance = FSelectInstanceSingleCrit$new(
  task = tsk("iris"),
  learner = lrn("classif.rpart"),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  terminator = terminator,
  store_models = TRUE
)

fselector = fs("rfe")

# Modifies the instance by reference
fselector$optimize(instance)

# Returns best scoring evaluation
instance$result

# Allows access of data.table of full path of all evaluations
as.data.table(instance$archive)
```

FSelectorSequential *Feature Selection via Sequential Selection*

Description

`FSelectorSequential` class that implements sequential feature selection. The sequential forward selection (`strategy = fsf`) extends the feature set in each step with the feature that increases the models performance the most. The sequential backward selection (`strategy = fsb`) starts with the complete future set and removes in each step the feature that decreases the models performance the least.

Dictionary

This FSelector can be instantiated via the dictionary mlr_fselectors or with the associated sugar function `fs()`:

```
mlr_fselectors$get("sequential")
fs("sequential")
```

Parameters

`max_features integer(1)`
: Maximum number of features. By default, number of features in mlr3::Task.

`strategy character(1)`
: Search method `sfs` (forward search) or `sbs` (backward search).

Super class

`mlr3fselect::FSelector` -> `FSelectorSequential`

Methods

Public methods:

- `FSelectorSequential$new()`
- `FSelectorSequential$optimization_path()`
- `FSelectorSequential$clone()`

Method `new()`**:** Creates a new instance of this R6 class.‘

Usage:

```
FSelectorSequential$new()
```

Method `optimization_path()`**:** Returns the optimization path.

Usage:

```
FSelectorSequential$optimization_path(inst)
```

Arguments:

`inst` (FSelectInstanceSingleCrit)
Instance optimized with FSelectorSequential.

Returns: data.table::data.table

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

Feature sets are evaluated in batches, where each batch is one step in the sequential feature selection.

Examples

```
library(mlr3)

terminator = trm("evals", n_evals = 5)

instance = FSelectInstanceSingleCrit$new(
  task = tsk("iris"),
  learner = lrn("classif.rpart"),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  terminator = terminator
)

fselector = fs("sequential")
```

```
# Modifies the instance by reference
fselector$optimize(instance)

# Returns best scoring evaluation
instance$result

# Allows access of data.table of full path of all evaluations
as.data.table(instance$archive)
```

`mlr_fselectors` *mlr_fselectors*

Description

A mlr3misc::Dictionary storing objects of class FSelector.

Usage

```
mlr_fselectors
```

Format

An object of class `DictionaryFSelect` (inherits from `Dictionary`, `R6`) of length 13.

`ObjectiveFSelect` *ObjectiveFSelect*

Description

Stores the objective function that estimates the performance of feature subsets. This class is usually constructed internally by by the FSelectInstanceSingleCrit / FSelectInstanceMultiCrit.

Super class

`bbotk::Objective` -> `ObjectiveFSelect`

Public fields

`task` (mlr3::Task)

`learner` (mlr3::Learner)

`resampling` (mlr3::Resampling)

`measures` (list of mlr3::Measure)

`store_models` (`logical(1)`).

`store_benchmark_result` (`logical(1)`).

`archive` (ArchiveFSelect).

Methods

Public methods:

- `ObjectiveFSelect$new()`
- `ObjectiveFSelect$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Creates a new instance of this R6 class.

Usage:

```
ObjectiveFSelect$new(
  task,
  learner,
  resampling,
  measures,
  check_values = TRUE,
  store_benchmark_result = TRUE,
  store_models = FALSE
)
```

Arguments:

`task` (mlr3::Task)
: Task to operate on.

`learner` (mlr3::Learner).

`resampling` (mlr3::Resampling)
: Uninstantiated resamplings are instantiated during construction so that all configurations are evaluated on the same data splits.

`measures` (list of mlr3::Measure)
: Measures to optimize. If `NULL`, **mlr3**'s default measure is used.

`check_values` (`logical(1)`)
: Check the parameters before the evaluation and the results for validity?

`store_benchmark_result` (`logical(1)`)
: Store benchmark result in archive?

`store_models` (`logical(1)`). Store models in benchmark result?

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Index