Package ‘mlr3fselect’

December 21, 2022

Title Feature Selection for 'mlr3'

Version 0.9.0

Description Feature selection package of the 'mlr3' ecosystem. It selects the optimal feature set for any 'mlr3' learner. The package works with several optimization algorithms e.g. Random Search, Recursive Feature Elimination, and Genetic Search. Moreover, it can automatically optimize learners and estimate the performance of optimized feature sets with nested resampling.

License LGPL-3

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

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

Depends mlr3 (>= 0.12.0), R (>= 3.1.0)

Imports bbotk (>= 0.7.2), checkmate (>= 2.0.0), data.table, lgr, mlr3misc (>= 0.9.4), mlr3pipelines (>= 0.3.0), paradox (>= 0.7.0), R6

Suggests genalg, rpart, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

NeedsCompilation no

RoxygenNote 7.2.3

Collate 'ArchiveFSelect.R' 'AutoFSelector.R' 'CallbackFSelect.R' 'ContextEval.R' 'FSelectInstanceSingleCrit.R' 'FSelectInstanceMultiCrit.R' 'mlr_fselectors.R' 'FSelector.R' 'FSelectorDesignPoints.R' 'FSelectorExhaustiveSearch.R' 'FSelectorFromOptimizer.R' 'FSelectorGeneticSearch.R' 'FSelectorRFE.R' 'FSelectorRandomSearch.R' 'FSelectorSequential.R' 'FSelectorShadowVariableSearch.R'

'ObjectiveFSelect.R' 'assertions.R' 'auto_fselector.R'
'bibentries.R' 'extract_inner_fselect_archives.R'
'extract_inner_fselect_results.R' 'fselect.R'
'fselect_nested.R' 'helper.R' 'mlr_callbacks.R' 'reexports.R'
'sugar.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 2022-12-21 14:00:02 UTC

R topics documented:

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

Description

Feature selection package of the 'mlr3' ecosystem. It selects the optimal feature set for any 'mlr3' learner. The package works with several optimization algorithms e.g. Random Search, Recursive Feature Elimination, and Genetic Search. Moreover, it can automatically optimize learners and estimate the performance of optimized feature sets with nested resampling.

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 *Class for Logging Evaluated Feature Sets*

Description

The ArchiveFSelect stores all evaluated feature sets and performance scores.

Details

The ArchiveFSelect is a container around a `data.table::data.table()`. Each row corresponds to a single evaluation of a feature set. See the section on Data Structure for more information. The archive stores additionally a mlr3::BenchmarkResult (`$benchmark_result`) that records the resampling experiments. Each experiment corresponds to to a single evaluation of a feature set. The table (`$data`) and the benchmark result (`$benchmark_result`) are linked by the `uhash` column. If the archive is passed to `as.data.table()`, both are joined automatically.

Data structure

The table (`$data`) has the following columns:

- One column for each feature of the task (`$search_space`).
- One column for each performance measure (`$codomain`).
- `runtime_learners` (`numeric(1)`)
  Sum of training and predict times logged in learners per mlr3::ResampleResult / evaluation. This does not include potential overhead time.
- `timestamp` (`POSIXct`)
  Time stamp when the evaluation was logged into the archive.
- `batch_nr` (`integer(1)`)
  Feature sets are evaluated in batches. Each batch has a unique batch number.
- `uhash` (`character(1)`)
  Connects each feature set to the resampling experiment stored in the mlr3::BenchmarkResult.

Analysis

For analyzing the feature selection results, it is recommended to pass the archive to `as.data.table()`. The returned data table is joined with the benchmark result which adds the mlr3::ResampleResult for each feature set.

The archive provides various getters (e.g. `$learners()`) to ease the access. All getters extract by position (`i`) or unique hash (`uhash`). For a complete list of all getters see the methods section.

The benchmark result (`$benchmark_result`) allows to score the feature sets again on a different measure. Alternatively, measures can be supplied to `as.data.table()`.

S3 Methods

- `as.data.table.ArchiveFSelect(x, exclude_columns = "uhash", measures = NULL)`
  Returns a tabular view of all evaluated feature sets.
  ArchiveFSelect -> `data.table::data.table()`

  - `x` (ArchiveFSelect)
  - `exclude_columns` (`character()`)
    Exclude columns from table. Set to `NULL` if no column should be excluded.
  - `measures` (list of mlr3::Measure)
    Score feature sets on additional measures.

Super class

`bbotk::Archive` -> `ArchiveFSelect`

Public fields

`benchmark_result` (mlr3::BenchmarkResult)
Benchmark result.

Methods

Public methods:

- ArchiveFSelect$new()
- ArchiveFSelect$learner()
- ArchiveFSelect$learners()
- ArchiveFSelect$predictions()
- ArchiveFSelect$resample_result()
- ArchiveFSelect$print()
- ArchiveFSelect$clone()

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

Usage:

```
ArchiveFSelect$new(search_space, codomain, check_values = TRUE)
```

Arguments:

`search_space` (paradox::ParamSet)
: Search space. Internally created from provided mlr3::Task by instance.

`codomain` (bbotk::Codomain)
: Specifies codomain of objective function i.e. a set of performance measures. Internally created from provided mlr3::Measures by instance.

`check_values` (`logical(1)`)
: If `TRUE` (default), hyperparameter configurations are check for validity.

Method `learner()`**:** Retrieve mlr3::Learner of the i-th evaluation, by position or by unique hash `uhash`. `i` and `uhash` are mutually exclusive. Learner does not contain a model. Use `$learners()` to get learners with models.

Usage:

```
ArchiveFSelect$learner(i = NULL, uhash = NULL)
```

Arguments:

`i` (`integer(1)`)
: The iteration value to filter for.

`uhash` (`logical(1)`)
: The `uhash` value to filter for.

Method `learners()`**:** Retrieve list of trained mlr3::Learner objects of the i-th evaluation, by position or by unique hash `uhash`. `i` and `uhash` are mutually exclusive.

Usage:

```
ArchiveFSelect$learners(i = NULL, uhash = NULL)
```

Arguments:

`i` (`integer(1)`)
: The iteration value to filter for.

`uhash` (`logical(1)`)
: The `uhash` value to filter for.

Method `predictions()`: Retrieve list of mlr3::Prediction objects of the i-th evaluation, by position or by unique hash `uhash`. `i` and `uhash` are mutually exclusive.

Usage:

```
ArchiveFSelect$predictions(i = NULL, uhash = NULL)
```

Arguments:

`i` (`integer(1)`)
The iteration value to filter for.

`uhash` (`logical(1)`)
The `uhash` value to filter for.

Method `resample_result()`: Retrieve mlr3::ResampleResult of the i-th evaluation, by position or by unique hash `uhash`. `i` and `uhash` are mutually exclusive.

Usage:

```
ArchiveFSelect$resample_result(i = NULL, uhash = NULL)
```

Arguments:

`i` (`integer(1)`)
The iteration value to filter for.

`uhash` (`logical(1)`)
The `uhash` value to filter for.

Method `print()`: Printer.

Usage:

```
ArchiveFSelect$print()
```

Arguments:

`...` (ignored).

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` *Class for Automatic Feature Selection*

Description

The AutoFSelector wraps a mlr3::Learner and augments it with an automatic feature selection. The `auto_fselector()` function creates an AutoFSelector object.

Details

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.

Resources

- book chapter on automatic feature selection.

Nested Resampling

Nested resampling can be performed 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` (see examples). The mlr3::Resampling passed to the AutoFSelector 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.

Super class

`mlr3::Learner` -> `AutoFSelector`

Public fields

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

`fselector` (FSelector)
: Optimization 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` (data.table::data.table)
: Short-cut to `$result` from FSelectInstanceSingleCrit.

`predict_type` (`character(1)`)
: Stores the currently active predict type, e.g. `"response"`. Must be an element of `$predict_types`.

`hash` (`character(1)`)
: Hash (unique identifier) for this object.

Methods

Public methods:

- `AutoFSelector$new()`
- `AutoFSelector$base_learner()`
- `AutoFSelector$importance()`
- `AutoFSelector$selected_features()`
- `AutoFSelector$oob_error()`
- `AutoFSelector$loglik()`
- `AutoFSelector$print()`
- `AutoFSelector$clone()`

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

Usage:

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

Arguments:

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

`resampling` (mlr3::Resampling)
: Resampling that is used to evaluated the performance of the feature subsets. Uninstantiated resamplings are instantiated during construction so that all feature subsets are evaluated on the same data splits. Already instantiated resamplings are kept unchanged.

`measure` (mlr3::Measure)
: Measure to optimize. If `NULL`, default measure is used.

`terminator` (Terminator)
: Stop criterion of the feature selection.

`fselector` (FSelector)
: Optimization algorithm.

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

`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?

`callbacks` (list of CallbackFSelect)
List of callbacks.

Method `base_learner()`: Extracts the base learner from nested learner objects like `GraphLearner` in **mlr3pipelines**. If `recursive = 0`, the (tuned) learner is returned.

Usage:

```
AutoFSelector$base_learner(recursive = Inf)
```

Arguments:

`recursive` (`integer(1)`)
Depth of recursion for multiple nested objects.

Returns: Learner.

Method `importance()`: The importance scores of the final model.

Usage:

```
AutoFSelector$importance()
```

Returns: Named `numeric()`.

Method `selected_features()`: The selected features of the final model. These features are selected internally by the learner.

Usage:

```
AutoFSelector$selected_features()
```

Returns: `character()`.

Method `oob_error()`: The out-of-bag error of the final model.

Usage:

```
AutoFSelector$oob_error()
```

Returns: `numeric(1)`.

Method `loglik()`: The log-likelihood of the final model.

Usage:

```
AutoFSelector$loglik()
```

Returns: `logLik`. Printer.

Method `print()`:

Usage:

```
AutoFSelector$print()
```

Arguments:

`...` (ignored).

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

```
# Automatic Feature Selection


# split to train and external set
task = tsk("penguins")
split = partition(task, ratio = 0.8)

# create auto fselector
afs = auto_fselector(
  method = fs("random_search"),
  learner = lrn("classif.rpart"),
  resampling = rsmp ("holdout"),
  measure = msr("classif.ce"),
  term_evals = 4)

# optimize feature subset and fit final model
afs$train(task, row_ids = split$train)

# predict with final model
afs$predict(task, row_ids = split$test)

# show result
afs$fselect_result

# model slot contains trained learner and fselect instance
afs$model

# shortcut trained learner
afs$learner

# shortcut fselect instance
afs$fselect_instance


# Nested Resampling

afs = auto_fselector(
  method = fs("random_search"),
  learner = lrn("classif.rpart"),
  resampling = rsmp ("holdout"),
  measure = msr("classif.ce"),
  term_evals = 4)

resampling_outer = rsmp("cv", folds = 3)
rr = resample(task, afs, resampling_outer, store_models = TRUE)

# retrieve inner feature selection results.
extract_inner_fselect_results(rr)

# performance scores estimated on the outer resampling
rr$score()
```

```
# unbiased performance of the final model trained on the full data set
rr$aggregate()
```

---

auto_fselector *Function for Automatic Feature Selection*

Description

The AutoFSelector wraps a mlr3::Learner and augments it with an automatic feature selection. The `auto_fselector()` function creates an AutoFSelector object.

Usage

```
auto_fselector(
  method,
  learner,
  resampling,
  measure = NULL,
  term_evals = NULL,
  term_time = NULL,
  terminator = NULL,
  store_fselect_instance = TRUE,
  store_benchmark_result = TRUE,
  store_models = FALSE,
  check_values = FALSE,
  callbacks = list(),
  ...
)
```

Arguments

`method` (`character(1)` | FSelector)
Key to retrieve fselector from mlr_fselectors dictionary or FSelector object.

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

`resampling` (mlr3::Resampling)
Resampling that is used to evaluated the performance of the feature subsets. Uninstantiated resamplings are instantiated during construction so that all feature subsets are evaluated on the same data splits. Already instantiated resamplings are kept unchanged.

`measure` (mlr3::Measure)
Measure to optimize. If `NULL`, default measure is used.

`term_evals` (`integer(1)`)
Number of allowed evaluations.

term_time
: (integer(1))
Maximum allowed time in seconds.

terminator
: (Terminator)
Stop criterion of the feature selection.

store_fselect_instance
: (logical(1))
If TRUE (default), stores the internally created FSelectInstanceSingleCrit with all intermediate results in slot $fselect_instance. Is set to TRUE, if store_models = TRUE

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?

callbacks
: (list of CallbackFSelect)
List of callbacks.

...
: (named list())
Named arguments to be set as parameters of the fselector.

Details

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.

Value

AutoFSelector.

Resources

- book chapter on automatic feature selection.

Nested Resampling

Nested resampling can be performed 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 (see examples). The mlr3::Resampling passed to the AutoFSelector 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.

Examples

```
# Automatic Feature Selection


# split to train and external set
task = tsk("penguins")
split = partition(task, ratio = 0.8)

# create auto fselector
afs = auto_fselector(
  method = fs("random_search"),
  learner = lrn("classif.rpart"),
  resampling = rsmp ("holdout"),
  measure = msr("classif.ce"),
  term_evals = 4)

# optimize feature subset and fit final model
afs$train(task, row_ids = split$train)

# predict with final model
afs$predict(task, row_ids = split$test)

# show result
afs$fselect_result

# model slot contains trained learner and fselect instance
afs$model

# shortcut trained learner
afs$learner

# shortcut fselect instance
afs$fselect_instance


# Nested Resampling

afs = auto_fselector(
  method = fs("random_search"),
  learner = lrn("classif.rpart"),
  resampling = rsmp ("holdout"),
  measure = msr("classif.ce"),
  term_evals = 4)

resampling_outer = rsmp("cv", folds = 3)
rr = resample(task, afs, resampling_outer, store_models = TRUE)

# retrieve inner feature selection results.
extract_inner_fselect_results(rr)

# performance scores estimated on the outer resampling
rr$score()
```

```
# unbiased performance of the final model trained on the full data set
rr$aggregate()
```

`CallbackFSelect` *Create Feature Selection Callback*

Description

Specialized bbotk::CallbackOptimization for feature selection. Callbacks allow to customize the behavior of processes in mlr3fselect. The `callback_fselect()` function creates a CallbackFSelect. Predefined callbacks are stored in the dictionary mlr_callbacks and can be retrieved with `clbk()`. For more information on callbacks see `callback_fselect()`.

Super classes

`mlr3misc::Callback` -> `bbotk::CallbackOptimization` -> `CallbackFSelect`

Public fields

`on_eval_after_design (function())`
: Stage called after design is created. Called in `ObjectiveFSelect$eval_many()`.

`on_eval_after_benchmark (function())`
: Stage called after feature sets are evaluated. Called in `ObjectiveFSelect$eval_many()`.

`on_eval_before_archive (function())`
: Stage called before performance values are written to the archive. Called in `ObjectiveFSelect$eval_many()`.

Methods

Public methods:

- `CallbackFSelect$clone()`

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

Usage:

`CallbackFSelect$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Write archive to disk
callback_fselect("mlr3fselect.backup",
  on_optimization_end = function(callback, context) {
    saveRDS(context$instance$archive, "archive.rds")
  }
)
```

`callback_fselect` *Create Feature Selection Callback*

Description

Function to create a CallbackFSelect. Predefined callbacks are stored in the dictionary mlr_callbacks and can be retrieved with `clbk()`.

Feature selection callbacks can be called from different stages of feature selection. The stages are prefixed with on_*.

```
Start Feature Selection
     - on_optimization_begin
    Start FSelect Batch
         - on_optimizer_before_eval
        Start Evaluation
             - on_eval_after_design
             - on_eval_after_benchmark
             - on_eval_before_archive
        End Evaluation
         - on_optimizer_after_eval
    End FSelect Batch
     - on_result
     - on_optimization_end
End Feature Selection
```

See also the section on parameters for more information on the stages. A feature selection callback works with bbotk::ContextOptimization and ContextEval.

Usage

```
callback_fselect(
  id,
  label = NA_character_,
  man = NA_character_,
  on_optimization_begin = NULL,
  on_optimizer_before_eval = NULL,
  on_eval_after_design = NULL,
  on_eval_after_benchmark = NULL,
  on_eval_before_archive = NULL,
  on_optimizer_after_eval = NULL,
  on_result = NULL,
  on_optimization_end = NULL
)
```

Arguments

- `id` (character(1))
  Identifier for the new instance.
- `label` (character(1))
  Label for the new instance.
- `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 `$help()`.
- `on_optimization_begin` (function())
  Stage called at the beginning of the optimization. Called in `Optimizer$optimize()`. The context available is bbotk::ContextOptimization.
- `on_optimizer_before_eval` (function())
  Stage called after the optimizer proposes points. Called in `OptimInstance$eval_batch()`. The context available is bbotk::ContextOptimization.
- `on_eval_after_design` (function())
  Stage called after design is created. Called in `ObjectiveFSelect$eval_many()`. The context available is ContextEval.
- `on_eval_after_benchmark` (function())
  Stage called after feature sets are evaluated. Called in `ObjectiveFSelect$eval_many()`. The context available is ContextEval.
- `on_eval_before_archive` (function())
  Stage called before performance values are written to the archive. Called in `ObjectiveFSelect$eval_many()`. The context available is ContextEval.
- `on_optimizer_after_eval` (function())
  Stage called after points are evaluated. Called in `OptimInstance$eval_batch()`. The context available is bbotk::ContextOptimization.
- `on_result` (function())
  Stage called after result are written. Called in `OptimInstance$assign_result()`. The context available is bbotk::ContextOptimization.
- `on_optimization_end` (function())
  Stage called at the end of the optimization. Called in `Optimizer$optimize()`. The context available is bbotk::ContextOptimization.

Details

When implementing a callback, each functions must have two arguments named `callback` and `context`.

A callback can write data to the state (`$state`), e.g. settings that affect the callback itself. Avoid writing large data the state. This can slow down the feature selection when the evaluation of configurations is parallelized.

Feature selection callbacks access two different contexts depending on the stage. The stages on_eval_after_design, on_eval_after_benchmark, on_eval_before_archive access ContextEval. This context can be used to customize the evaluation of a batch of feature sets. Changes to the state of callback are lost after the evaluation of a batch and changes to the fselect instance or the fselector are not possible. Persistent data should be written to the archive via $aggregated_performance (see ContextEval). The other stages access ContextOptimization. This context can be used to modify the fselect instance, archive, fselector and final result. There are two different contexts because the evaluation can be parallelized i.e. multiple instances of ContextEval exists on different workers at the same time.

Examples

```
# Write archive to disk
callback_fselect("mlr3fselect.backup",
  on_optimization_end = function(callback, context) {
    saveRDS(context$instance$archive, "archive.rds")
  }
)
```

ContextEval *Evaluation Context*

Description

The ContextEval allows CallbackFSelects to access and modify data while a batch of feature sets is evaluated. See section on active bindings for a list of modifiable objects. See callback_fselect() for a list of stages which access ContextEval.

Details

This context is re-created each time a new batch of feature sets is evaluated. Changes to $objective_fselect, $design $benchmark_result are discarded after the function is finished. Modification on the data table in $aggregated_performance are written to the archive. Any number of columns can be added.

Super class

mlr3misc::Context -> ContextEval

Public fields

objective_fselect ObjectiveFSelect.

Active bindings

`xss` (list())
: The feature sets of the latest batch.

`design` (data.table::data.table)
: The benchmark design of the latest batch.

`benchmark_result` (mlr3::BenchmarkResult)
: The benchmark result of the latest batch.

`aggregated_performance` (data.table::data.table)
: Aggregated performance scores and training time of the latest batch. This data table is passed to the archive. A callback can add additional columns which are also written to the archive.

Methods

Public methods:

- `ContextEval$new()`
- `ContextEval$clone()`

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

Usage:

```
ContextEval$new(objective_fselect)
```

Arguments:

`objective_fselect` ObjectiveFSelect.

`id` (`character(1)`)
: Identifier for the new callback.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

---

extract_inner_fselect_archives

Extract Inner Feature Selection Archives

Description

Extract inner feature selection archives of nested resampling. Implemented for mlr3::ResampleResult and mlr3::BenchmarkResult. The function iterates over the AutoFSelector objects and binds the archives to a `data.table::data.table()`. AutoFSelector must be initialized with `store_fselect_instance = TRUE` and `resample()` or `benchmark()` must be called with `store_models = TRUE`.

Usage

```
extract_inner_fselect_archives(x, exclude_columns = "uhash")
```

Arguments

`x` (mlr3::ResampleResult | mlr3::BenchmarkResult).

`exclude_columns`
(`character()`)
Exclude columns from result table. Set to `NULL` if no column should be excluded.

Value

`data.table::data.table()`.

Data structure

The returned data table has the following columns:

- `experiment` (integer(1))
  Index, giving the according row number in the original benchmark grid.
- `iteration` (integer(1))
  Iteration of the outer resampling.
- One column for each feature of the task.
- One column for each performance measure.
- `runtime_learners` (`numeric(1)`)
  Sum of training and predict times logged in learners per mlr3::ResampleResult / evaluation. This does not include potential overhead time.
- `timestamp` (`POSIXct`)
  Time stamp when the evaluation was logged into the archive.
- `batch_nr` (`integer(1)`)
  Feature sets are evaluated in batches. Each batch has a unique batch number.
- `resample_result` (mlr3::ResampleResult)
  Resample result of the inner resampling.
- `task_id` (`character(1)`).
- `learner_id` (`character(1)`).
- `resampling_id` (`character(1)`).

Examples

```
# Nested Resampling on Palmer Penguins Data Set

# create auto fselector
at = auto_fselector(
  method = fs("random_search"),
  learner = lrn("classif.rpart"),
  resampling = rsmp ("holdout"),
```

```
  measure = msr("classif.ce"),
  term_evals = 4)

resampling_outer = rsmp("cv", folds = 2)
rr = resample(tsk("penguins"), at, resampling_outer, store_models = TRUE)

# extract inner archives
extract_inner_fselect_archives(rr)
```

extract_inner_fselect_results — *Extract Inner Feature Selection Results*

Description

Extract inner feature selection results of nested resampling. Implemented for mlr3::ResampleResult and mlr3::BenchmarkResult.

Usage

```
extract_inner_fselect_results(x, fselect_instance, ...)
```

Arguments

`x`	(mlr3::ResampleResult | mlr3::BenchmarkResult).
`fselect_instance`	(`logical(1)`) If `TRUE`, instances are added to the table.
`...`	(any) Additional arguments.

Details

The function iterates over the AutoFSelector objects and binds the feature selection results to a `data.table::data.table()`. AutoFSelector must be initialized with `store_fselect_instance = TRUE` and `resample()` or `benchmark()` must be called with `store_models = TRUE`. Optionally, the instance can be added for each iteration.

Value

`data.table::data.table()`.

Data structure

The returned data table has the following columns:

- `experiment` (integer(1))
  Index, giving the according row number in the original benchmark grid.
- `iteration` (integer(1))
  Iteration of the outer resampling.
- One column for each feature of the task.
- One column for each performance measure.
- `features` (character())
  Vector of selected feature set.
- `task_id` (`character(1)`).
- `learner_id` (`character(1)`).
- `resampling_id` (`character(1)`).

Examples

```
# Nested Resampling on Palmer Penguins Data Set

# create auto fselector
at = auto_fselector(
  method = fs("random_search"),
  learner = lrn("classif.rpart"),
  resampling = rsmp ("holdout"),
  measure = msr("classif.ce"),
  term_evals = 4)

resampling_outer = rsmp("cv", folds = 2)
rr = resample(tsk("iris"), at, resampling_outer, store_models = TRUE)

# extract inner results
extract_inner_fselect_results(rr)
```

`fs` *Syntactic Sugar for FSelect Construction*

Description

Functions to retrieve objects, set parameters and assign to fields in one go. Relies on `mlr3misc::dictionary_sugar_get()` to extract objects from the respective mlr3misc::Dictionary:

- `fs()` for a FSelector from mlr_fselectors.
- `fss()` for a list of FSelectors from mlr_fselectors.
- `trm()` for a Terminator from mlr_terminators.
- `trms()` for a list of Terminators from mlr_terminators.

Usage

```
fs(.key, ...)

fss(.keys, ...)
```

Arguments

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

Value

R6::R6Class object of the respective type, or a list of R6::R6Class objects for the plural versions.

Examples

```
# random search with batch size of 5
fs("random_search", batch_size = 5)

# run time terminator with 20 seconds
trm("run_time", secs = 20)
```

`fselect` *Function for Feature Selection*

Description

Function to optimize the features of a mlr3::Learner. The function internally creates a FSelectInstanceSingleCrit or FSelectInstanceMultiCrit which describe the feature selection problem. It executes the feature selection with the FSelector (`method`) and returns the result with the fselect instance (`$result`). The ArchiveFSelect (`$archive`) stores all evaluated hyperparameter configurations and performance scores.

Usage

```
fselect(
  method,
  task,
  learner,
  resampling,
  measures = NULL,
```

```
  term_evals = NULL,
  term_time = NULL,
  terminator = NULL,
  store_benchmark_result = TRUE,
  store_models = FALSE,
  check_values = FALSE,
  callbacks = list(),
  ...
)
```

Arguments

`method` (`character(1)` | FSelector)
Key to retrieve fselector from mlr_fselectors dictionary or FSelector object.

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

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

`resampling` (mlr3::Resampling)
Resampling that is used to evaluated the performance of the feature subsets. Uninstantiated resamplings are instantiated during construction so that all feature subsets are evaluated on the same data splits. Already instantiated resamplings are kept unchanged.

`measures` (mlr3::Measure or list of mlr3::Measure)
A single measure creates a FSelectInstanceSingleCrit and multiple measures a FSelectInstanceMultiCrit. If `NULL`, default measure is used.

`term_evals` (`integer(1)`)
Number of allowed evaluations.

`term_time` (`integer(1)`)
Maximum allowed time in seconds.

`terminator` (Terminator)
Stop criterion of the feature selection.

`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?

`callbacks` (list of CallbackFSelect)
List of callbacks.

`...` (named `list()`)
Named arguments to be set as parameters of the fselector.

Details

The mlr3::Task, mlr3::Learner, mlr3::Resampling, mlr3::Measure and Terminator are used to construct a FSelectInstanceSingleCrit. If multiple performance Measures are supplied, a FSelectInstanceMultiCrit is created. The parameter `term_evals` and `term_time` are shortcuts to create a Terminator. If both parameters are passed, a TerminatorCombo is constructed. For other Terminators, pass one with `terminator`. If no termination criterion is needed, set `term_evals`, `term_time` and `terminator` to `NULL`.

Value

FSelectInstanceSingleCrit | FSelectInstanceMultiCrit

Resources

- book chapter on feature selection.
- gallery post on feature selection on the Titanic data set.

Analysis

For analyzing the feature selection results, it is recommended to pass the archive to `as.data.table()`. The returned data table is joined with the benchmark result which adds the mlr3::ResampleResult for each feature set.

The archive provides various getters (e.g. `$learners()`) to ease the access. All getters extract by position (`i`) or unique hash (`uhash`). For a complete list of all getters see the methods section.

The benchmark result (`$benchmark_result`) allows to score the feature sets again on a different measure. Alternatively, measures can be supplied to `as.data.table()`.

Examples

```
# Feature selection on the Palmer Penguins data set
task = tsk("pima")
learner = lrn("classif.rpart")

# Run feature selection
instance = fselect(
  method = "random_search",
  task = task,
  learner = learner,
  resampling = rsmp ("holdout"),
  measures = msr("classif.ce"),
  term_evals = 4)

# Subset task to optimized feature set
task$select(instance$result_feature_set)

# Train the learner with optimal feature set on the full data set
learner$train(task)

# Inspect all evaluated configurations
as.data.table(instance$archive)
```

FSelectInstanceMultiCrit

Class for Multi Criteria Feature Selection

Description

The FSelectInstanceMultiCrit specifies a feature selection problem for FSelectors. The function `fsi()` creates a FSelectInstanceMultiCrit and the function `fselect()` creates an instance internally.

Resources

- book chapter on feature selection.
- gallery post on feature selection on the Titanic data set.

Analysis

For analyzing the feature selection results, it is recommended to pass the archive to `as.data.table()`. The returned data table is joined with the benchmark result which adds the mlr3::ResampleResult for each feature set.

The archive provides various getters (e.g. `$learners()`) to ease the access. All getters extract by position (`i`) or unique hash (`uhash`). For a complete list of all getters see the methods section.

The benchmark result (`$benchmark_result`) allows to score the feature sets again on a different measure. Alternatively, measures can be supplied to `as.data.table()`.

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$print()`
- `FSelectInstanceMultiCrit$clone()`

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

Usage:

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

Arguments:

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

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

`resampling` (mlr3::Resampling)
: Resampling that is used to evaluated the performance of the feature subsets. Uninstantiated resamplings are instantiated during construction so that all feature subsets are evaluated on the same data splits. Already instantiated resamplings are kept unchanged.

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

`terminator` (Terminator)
: Stop criterion of the feature selection.

`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?

`callbacks` (list of CallbackFSelect)
: List of callbacks.

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 `print()`**:** Printer.

Usage:

```
FSelectInstanceMultiCrit$print(...)
```

Arguments:

`...` (ignored).

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

```
# Feature selection on Palmer Penguins data set


task = tsk("penguins")

# Construct feature selection instance
instance = fsi(
  task = task,
  learner = lrn("classif.rpart"),
  resampling = rsmp("cv", folds = 3),
  measures = msrs(c("classif.ce", "time_train")),
  terminator = trm("evals", n_evals = 4)
)

# Choose optimization algorithm
fselector = fs("random_search", batch_size = 2)

# Run feature selection
fselector$optimize(instance)

# Optimal feature sets
instance$result_feature_set

# Inspect all evaluated sets
as.data.table(instance$archive)
```

`FSelectInstanceSingleCrit` *Class for Single Criterion Feature Selection*

Description

The FSelectInstanceSingleCrit specifies a feature selection problem for FSelectors. The function `fsi()` creates a FSelectInstanceSingleCrit and the function `fselect()` creates an instance internally.

The instance contains an ObjectiveFSelect object that encodes the black box objective function a FSelector has to optimize. The instance allows the basic operations of querying the objective at design points (`$eval_batch()`). This operation is usually done by the FSelector. Evaluations of feature subsets are performed in batches by calling `mlr3::benchmark()` internally. The evaluated feature subsets are stored in the Archive (`$archive`). 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()`.

Resources

- book chapter on feature selection.
- gallery post on feature selection on the Titanic data set.

Analysis

For analyzing the feature selection results, it is recommended to pass the archive to `as.data.table()`. The returned data table is joined with the benchmark result which adds the mlr3::ResampleResult for each feature set.

The archive provides various getters (e.g. `$learners()`) to ease the access. All getters extract by position (`i`) or unique hash (`uhash`). For a complete list of all getters see the methods section.

The benchmark result (`$benchmark_result`) allows to score the feature sets again on a different measure. Alternatively, measures can be supplied to `as.data.table()`.

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$print()`
- `FSelectInstanceSingleCrit$clone()`

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

Usage:

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

Arguments:

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

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

`resampling` (mlr3::Resampling)
: Resampling that is used to evaluated the performance of the feature subsets. Uninstantiated resamplings are instantiated during construction so that all feature subsets are evaluated on the same data splits. Already instantiated resamplings are kept unchanged.

`measure` (mlr3::Measure)
: Measure to optimize. If `NULL`, default measure is used.

`terminator` (Terminator)
: Stop criterion of the feature selection.

`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?

`callbacks` (list of CallbackFSelect)
: List of callbacks.

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 `print()`: Printer.

Usage:

```
FSelectInstanceSingleCrit$print(...)
```

Arguments:

`...` (ignored).

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

```
# Feature selection on Palmer Penguins data set


task = tsk("penguins")
learner = lrn("classif.rpart")

# Construct feature selection instance
instance = fsi(
  task = task,
  learner = learner,
  resampling = rsmp("cv", folds = 3),
  measures = msr("classif.ce"),
  terminator = trm("evals", n_evals = 4)
)

# Choose optimization algorithm
fselector = fs("random_search", batch_size = 2)

# Run feature selection
fselector$optimize(instance)

# Subset task to optimal feature set
task$select(instance$result_feature_set)

# Train the learner with optimal feature set on the full data set
learner$train(task)

# Inspect all evaluated sets
as.data.table(instance$archive)
```

`FSelector` *Class for Feature Selection Algorithms*

Description

The FSelector implements the optimization algorithm.

Details

FSelector is a abstract base class that implements the base functionality each fselector must provide. 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 design points.
- 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$archive`.
- Before the batch evaluation, the bbotk::Terminator is checked, and if it is positive, an exception of class `"terminated_error"` is generated. In the later case the current batch of evaluations is still stored in `instance`, but the numeric scores are not sent back to the handling optimizer as it has lost execution control.
- After such an exception was caught we select the best set from `instance$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 set in the instance and its estimated performance. The default behavior is: We pick the best resample-experiment, regarding the given measure, then assign its set and aggregated performance to the instance.

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.

Resources

- book section on feature selection algorithms.

Public fields

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

Active bindings

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

`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.

`label` (`character(1)`)
Label for this object. Can be used in tables, plot and text output instead of the ID.

`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()`.

Methods

Public methods:

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

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

Usage:

```
FSelector$new(
  id = "fselector",
  param_set,
  properties,
  packages = character(),
  label = NA_character_,
  man = NA_character_
)
```

Arguments:

`id` (`character(1)`)
Identifier for the new instance.

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

`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.

`label` (`character(1)`)
Label for this object. Can be used in tables, plot and text output instead of the ID.

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

Usage:

```
FSelector$format(...)
```

Arguments:

`...` (ignored).

Returns: (`character()`).

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

Usage:

```
FSelector$print()
```

Returns: (`character()`).

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

Usage:

```
FSelector$help()
```

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.

`fselect_nested` *Function for Nested Resampling*

Description

Function to conduct nested resampling.

Usage

```
fselect_nested(
  method,
  task,
  learner,
  inner_resampling,
  outer_resampling,
  measure,
  term_evals = NULL,
  term_time = NULL,
  ...
)
```

Arguments

`method`	(`character(1)`) Key to retrieve fselector from mlr_fselectors dictionary.
`task`	(mlr3::Task) Task to operate on.
`learner`	(mlr3::Learner) Learner to optimize the feature subset for.
`inner_resampling`	(mlr3::Resampling) Resampling used for the inner loop.
`outer_resampling`	mlr3::Resampling) Resampling used for the outer loop.
`measure`	(mlr3::Measure) Measure to optimize. If `NULL`, default measure is used.
`term_evals`	(`integer(1)`) Number of allowed evaluations.
`term_time`	(`integer(1)`) Maximum allowed time in seconds.
`...`	(named `list()`) Named arguments to be set as parameters of the fselector.

Value

mlr3::ResampleResult

Examples

```
# Nested resampling on Palmer Penguins data set
rr = fselect_nested(
  method = "random_search",
  task = tsk("penguins"),
  learner = lrn("classif.rpart"),
```

```
  inner_resampling = rsmp ("holdout"),
  outer_resampling = rsmp("cv", folds = 2),
  measure = msr("classif.ce"),
  term_evals = 4)

# Performance scores estimated on the outer resampling
rr$score()

# Unbiased performance of the final model trained on the full data set
rr$aggregate()
```

`fsi` *Syntactic Sugar for Instance Construction*

Description

Function to construct a FSelectInstanceSingleCrit or FSelectInstanceMultiCrit.

Usage

```
fsi(
  task,
  learner,
  resampling,
  measures = NULL,
  terminator,
  store_benchmark_result = TRUE,
  store_models = FALSE,
  check_values = FALSE
)
```

Arguments

`task`	(mlr3::Task) Task to operate on.
`learner`	(mlr3::Learner) Learner to optimize the feature subset for.
`resampling`	(mlr3::Resampling) Resampling that is used to evaluated the performance of the feature subsets. Uninstantiated resamplings are instantiated during construction so that all feature subsets are evaluated on the same data splits. Already instantiated resamplings are kept unchanged.
`measures`	(mlr3::Measure or list of mlr3::Measure) A single measure creates a FSelectInstanceSingleCrit and multiple measures a FSelectInstanceMultiCrit. If `NULL`, default measure is used.
`terminator`	(Terminator) Stop criterion of the feature selection.

`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?

Resources

- book chapter on feature selection.
- gallery post on feature selection on the Titanic data set.

Examples

```
# Feature selection on Palmer Penguins data set


task = tsk("penguins")
learner = lrn("classif.rpart")

# Construct feature selection instance
instance = fsi(
  task = task,
  learner = learner,
  resampling = rsmp("cv", folds = 3),
  measures = msr("classif.ce"),
  terminator = trm("evals", n_evals = 4)
)

# Choose optimization algorithm
fselector = fs("random_search", batch_size = 2)

# Run feature selection
fselector$optimize(instance)

# Subset task to optimal feature set
task$select(instance$result_feature_set)

# Train the learner with optimal feature set on the full data set
learner$train(task)

# Inspect all evaluated sets
as.data.table(instance$archive)
```

`mlr3fselect.backup` *Backup Benchmark Result Callback*

Description

This CallbackFSelect writes the mlr3::BenchmarkResult after each batch to disk.

Examples

```
clbk("mlr3fselect.backup", path = "backup.rds")

# Run feature selection on the Palmer Penguins data set
instance = fselect(
  method = "random_search",
  task = tsk("pima"),
  learner = lrn("classif.rpart"),
  resampling = rsmp ("holdout"),
  measures = msr("classif.ce"),
  term_evals = 4,
  callbacks = clbk("mlr3fselect.backup", path = tempfile(fileext = ".rds")))
```

---

mlr_fselectors *Dictionary of FSelectors*

Description

A mlr3misc::Dictionary storing objects of class FSelector. Each fselector has an associated help page, see `mlr_fselectors_[id]`.

For a more convenient way to retrieve and construct fselectors, see `fs()`/`fss()`.

Format

R6::R6Class object inheriting from mlr3misc::Dictionary.

Methods

See mlr3misc::Dictionary.

S3 methods

- `as.data.table(dict, ..., objects = FALSE)`
  mlr3misc::Dictionary -> `data.table::data.table()`
  Returns a `data.table::data.table()` with fields "key", "label", "properties" and "packages" as columns. If `objects` is set to `TRUE`, the constructed objects are returned in the list column named `object`.

See Also

Sugar functions: `fs()`, `fss()`

Other FSelector: `mlr_fselectors_design_points`, `mlr_fselectors_exhaustive_search`, `mlr_fselectors_genetic_s` `mlr_fselectors_random_search`, `mlr_fselectors_rfe`, `mlr_fselectors_sequential`, `mlr_fselectors_shadow_vari`

Examples

```
as.data.table(mlr_fselectors)
mlr_fselectors$get("random_search")
fs("random_search")
```

---

mlr_fselectors_design_points

Feature Selection with Design Points

Description

Feature selection using user-defined feature sets.

Details

The feature sets are evaluated in order as given.

The feature selection terminates itself when all feature sets are evaluated. It is not necessary to set a termination criterion.

Dictionary

This FSelector can be instantiated with the associated sugar function `fs()`:

```
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.

See Also

Other FSelector: `mlr_fselectors_exhaustive_search`, `mlr_fselectors_genetic_search`, `mlr_fselectors_random_s` `mlr_fselectors_rfe`, `mlr_fselectors_sequential`, `mlr_fselectors_shadow_variable_search`, `mlr_fselectors`

Examples

```
# Feature Selection


# retrieve task and load learner
task = tsk("pima")
learner = lrn("classif.rpart")

# create design
design = mlr3misc::rowwise_table(
  ~age, ~glucose, ~insulin, ~mass, ~pedigree, ~pregnant, ~pressure, ~triceps,
  TRUE, FALSE,    TRUE,     TRUE,  FALSE,     TRUE,       FALSE,    TRUE,
  TRUE, TRUE,     FALSE,    TRUE,  FALSE,     TRUE,       FALSE,    FALSE,
  TRUE, FALSE,    TRUE,     TRUE,  FALSE,     TRUE,       FALSE,    FALSE,
  TRUE, FALSE,    TRUE,     TRUE,  FALSE,     TRUE,       TRUE,     TRUE
)

# run feature selection on the Pima Indians diabetes data set
instance = fselect(
  method = fs("design_points", design = design),
  task = task,
  learner = learner,
  resampling = rsmp("holdout"),
  measure = msr("classif.ce")
)

# best performing feature set
instance$result

# all evaluated feature sets
as.data.table(instance$archive)

# subset the task and fit the final model
task$select(instance$result_feature_set)
learner$train(task)
```

mlr_fselectors_exhaustive_search
Feature Selection with Exhaustive Search

Description

Feature Selection using the Exhaustive Search Algorithm. Exhaustive Search generates all possible feature sets.

Details

The feature selection terminates itself when all feature sets are evaluated. It is not necessary to set a termination criterion.

Dictionary

This FSelector can be instantiated with the associated sugar function `fs()`:

```
fs("exhaustive_search")
```

Control 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.

See Also

Other FSelector: mlr_fselectors_design_points, mlr_fselectors_genetic_search, mlr_fselectors_random_search, mlr_fselectors_rfe, mlr_fselectors_sequential, mlr_fselectors_shadow_variable_search, mlr_fselectors

Examples

```
# Feature Selection


# retrieve task and load learner
task = tsk("penguins")
learner = lrn("classif.rpart")

# run feature selection on the Palmer Penguins data set
instance = fselect(
  method = "exhaustive_search",
  task = task,
  learner = learner,
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  term_evals = 10
)

# best performing feature set
instance$result

# all evaluated feature sets
as.data.table(instance$archive)

# subset the task and fit the final model
task$select(instance$result_feature_set)
learner$train(task)
```

---

mlr_fselectors_genetic_search
Feature Selection with Genetic Search

Description

Feature selection using the Genetic Algorithm from the package **genalg**.

Dictionary

This FSelector can be instantiated with the associated sugar function `fs()`:

```
fs("genetic_search")
```

Control Parameters

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.

See Also

Other FSelector: `mlr_fselectors_design_points`, `mlr_fselectors_exhaustive_search`, `mlr_fselectors_random_se` `mlr_fselectors_rfe`, `mlr_fselectors_sequential`, `mlr_fselectors_shadow_variable_search`, `mlr_fselectors`

Examples

```
# Feature Selection


# retrieve task and load learner
task = tsk("penguins")
learner = lrn("classif.rpart")

# run feature selection on the Palmer Penguins data set
instance = fselect(
  method = "genetic_search",
  task = task,
  learner = learner,
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  term_evals = 10
)
```

```
# best performing feature set
instance$result

# all evaluated feature sets
as.data.table(instance$archive)

# subset the task and fit the final model
task$select(instance$result_feature_set)
learner$train(task)
```

---

mlr_fselectors_random_search

Feature Selection with Random Search

Description

Feature selection using Random Search Algorithm.

Details

The feature sets are randomly drawn. The sets are evaluated in batches 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 with the associated sugar function `fs()`:

```
fs("random_search")
```

Control 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.

See Also

Other FSelector: `mlr_fselectors_design_points`, `mlr_fselectors_exhaustive_search`, `mlr_fselectors_genetic_s`
`mlr_fselectors_rfe`, `mlr_fselectors_sequential`, `mlr_fselectors_shadow_variable_search`, `mlr_fselectors`

Examples

```
# Feature Selection


# retrieve task and load learner
task = tsk("penguins")
learner = lrn("classif.rpart")

# run feature selection on the Palmer Penguins data set
instance = fselect(
  method = fs("random_search"),
  task = task,
  learner = learner,
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  term_evals = 10
)

# best performing feature subset
instance$result
```

```
# all evaluated feature subsets
as.data.table(instance$archive)

# subset the task and fit the final model
task$select(instance$result_feature_set)
learner$train(task)
```

---

mlr_fselectors_rfe *Feature Selection with Recursive Feature Elimination*

Description

Feature selection using the Recursive Feature Elimination Algorithm (RFE). Recursive feature elimination iteratively removes features with a low importance score. Only works with Learners that can calculate importance scores (see section on optional extractors in Learner).

Details

The learner is trained on all features at the start and importance scores are calculated for each feature . Then the least important feature is removed and the learner is trained on the reduced feature set. The importance scores are calculated again and the procedure is repeated until the desired number of features is reached. The non-recursive option (`recursive = FALSE`) only uses the importance scores calculated in the first iteration.

The feature selection terminates itself when `n_features` is reached. It is not necessary to set a termination criterion.

Dictionary

This FSelector can be instantiated with the associated sugar function `fs()`:

```
fs("rfe")
```

Control Parameters

`n_features` `integer(1)`
: The number of features to select. By default half of the features are selected.

`feature_fraction` `double(1)`
: Fraction of features to retain in each iteration. The default 0.5 retrains half of the features.

`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)`
: If `TRUE` (default), the feature importance is calculated in each iteration.

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 recursive 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.

See Also

Other FSelector: mlr_fselectors_design_points, mlr_fselectors_exhaustive_search, mlr_fselectors_genetic_s mlr_fselectors_random_search, mlr_fselectors_sequential, mlr_fselectors_shadow_variable_search, mlr_fselectors

Examples

```
# Feature Selection


# retrieve task and load learner
task = tsk("penguins")
learner = lrn("classif.rpart")

# run feature selection on the Palmer Penguins data set
instance = fselect(
  method = fs("rfe"),
  task = task,
  learner = learner,
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  store_models = TRUE
)
```

```
# best performing feature subset
instance$result

# all evaluated feature subsets
as.data.table(instance$archive)

# subset the task and fit the final model
task$select(instance$result_feature_set)
learner$train(task)
```

---

mlr_fselectors_sequential

Feature Selection with Sequential Search

Description

Feature selection using Sequential Search Algorithm.

Details

Sequential forward selection (`strategy = fsf`) extends the feature set in each iteration with the feature that increases the models performance the most. Sequential backward selection (`strategy = fsb`) follows the same idea but starts with all features and removes features from the set.

The feature selection terminates itself when `min_features` or `max_features` is reached. It is not necessary to set a termination criterion.

Dictionary

This FSelector can be instantiated with the associated sugar function `fs()`:

```
fs("sequential")
```

Control Parameters

`min_features` `integer(1)`
: Minimum number of features. By default, 1.

`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, include_uhash = FALSE)
```

Arguments:

`inst` (FSelectInstanceSingleCrit)
Instance optimized with FSelectorSequential.

`include_uhash` (`logical(1)`)
Include uhash column?

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.

See Also

Other FSelector: `mlr_fselectors_design_points`, `mlr_fselectors_exhaustive_search`, `mlr_fselectors_genetic_s` `mlr_fselectors_random_search`, `mlr_fselectors_rfe`, `mlr_fselectors_shadow_variable_search`, `mlr_fselectors`

Examples

```
# Feature Selection


# retrieve task and load learner
task = tsk("penguins")
learner = lrn("classif.rpart")

# run feature selection on the Palmer Penguins data set
instance = fselect(
  method = "sequential",
  task = task,
  learner = learner,
```

```
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  term_evals = 10
)

# best performing feature set
instance$result

# all evaluated feature sets
as.data.table(instance$archive)

# subset the task and fit the final model
task$select(instance$result_feature_set)
learner$train(task)
```

`mlr_fselectors_shadow_variable_search`
Feature Selection with Shadow Variable Search

Description

Feature selection using the Shadow Variable Search Algorithm. Shadow variable search creates for each feature a permutated copy and stops when one of them is selected.

Details

The feature selection terminates itself when the first shadow variable is selected. It is not necessary to set a termination criterion.

Dictionary

This FSelector can be instantiated with the associated sugar function `fs()`:

```
fs("shadow_variable_search")
```

Super class

`mlr3fselect::FSelector` -> `FSelectorShadowVariableSearch`

Methods

Public methods:

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

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

Usage:

```
FSelectorShadowVariableSearch$new()
```

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

Usage:

```
FSelectorShadowVariableSearch$optimization_path(inst)
```

Arguments:

`inst` (FSelectInstanceSingleCrit)
Instance optimized with FSelectorShadowVariableSearch.

Returns: data.table::data.table

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Source

Thomas J, Hepp T, Mayr A, Bischl B (2017). "Probing for Sparse and Fast Variable Selection with Model-Based Boosting." *Computational and Mathematical Methods in Medicine*, **2017**, 1–8. doi:10.1155/2017/1421409.

Wu Y, Boos DD, Stefanski LA (2007). "Controlling Variable Selection by the Addition of Pseudovariables." *Journal of the American Statistical Association*, **102**(477), 235–243. doi:10.1198/016214506000000843.

See Also

Other FSelector: `mlr_fselectors_design_points`, `mlr_fselectors_exhaustive_search`, `mlr_fselectors_genetic_s` `mlr_fselectors_random_search`, `mlr_fselectors_rfe`, `mlr_fselectors_sequential`, `mlr_fselectors`

Examples

```
# Feature Selection


# retrieve task and load learner
task = tsk("penguins")
learner = lrn("classif.rpart")

# run feature selection on the Palmer Penguins data set
instance = fselect(
  method = fs("shadow_variable_search"),
  task = task,
  learner = learner,
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
)
```

```
# best performing feature subset
instance$result

# all evaluated feature subsets
as.data.table(instance$archive)

# subset the task and fit the final model
task$select(instance$result_feature_set)
learner$train(task)
```

`ObjectiveFSelect` *Class for Feature Selection Objective*

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).

`callbacks` (List of CallbackFSelects).

Methods

Public methods:

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

Method `new()`**:** 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,
  archive = NULL,
  callbacks = list()
)
```

Arguments:

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

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

`resampling` (mlr3::Resampling)
: Resampling that is used to evaluated the performance of the feature subsets. Uninstantiated resamplings are instantiated during construction so that all feature subsets are evaluated on the same data splits. Already instantiated resamplings are kept unchanged.

`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?

`archive` (ArchiveFSelect)
: Reference to the archive of FSelectInstanceSingleCrit | FSelectInstanceMultiCrit. If `NULL` (default), benchmark result and models cannot be stored.

`callbacks` (list of CallbackFSelect)
: List of callbacks.

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