Package ‘mlr3fselect’

September 9, 2024

Title Feature Selection for 'mlr3'

Version 1.1.0

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 (>= 1.0.0), checkmate (>= 2.0.0), data.table, lgr, mlr3misc (>= 0.15.1), paradox (>= 1.0.0), R6, stabm

Suggests e1071, genalg, mlr3learners, mlr3pipelines, rpart, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

NeedsCompilation no

RoxygenNote 7.3.2

Collate 'ArchiveBatchFSelect.R' 'AutoFSelector.R' 'CallbackBatchFSelect.R' 'ContextBatchFSelect.R' 'EnsembleFSResult.R' 'FSelectInstanceBatchSingleCrit.R' 'FSelectInstanceBatchMultiCrit.R' 'mlr_fselectors.R' 'FSelector.R' 'FSelectorBatch.R' 'FSelectorBatchDesignPoints.R' 'FSelectorBatchExhaustiveSearch.R' 'FSelectorBatchFromOptimizerBatch.R'

'FSelectorBatchGeneticSearch.R' 'FSelectorBatchRFE.R'
'FSelectorBatchRFECV.R' 'FSelectorBatchRandomSearch.R'
'FSelectorBatchSequential.R'
'FSelectorBatchShadowVariableSearch.R' 'ObjectiveFSelect.R'
'ObjectiveFSelectBatch.R' 'assertions.R' 'auto_fselector.R'
'bibentries.R' 'ensemble_fselect.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>),
John Zobolas [aut] (<https://orcid.org/0000-0002-3609-8674>)

Maintainer Marc Becker <marcbecker@posteo.de>

Repository CRAN

Date/Publication 2024-09-09 19:00:12 UTC

Contents

`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)
- John Zobolas `<bblodfon@gmail.com>` (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`

ArchiveBatchFSelect *Class for Logging Evaluated Feature Sets*

Description

The ArchiveBatchFSelect stores all evaluated feature sets and performance scores.

Details

The ArchiveBatchFSelect 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 (`$benchmark_result`) that records the resampling experiments. Each experiment corresponds to a single evaluation of a feature set. The table (`$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.ArchiveBatchFSelect(x, exclude_columns = "uhash", measures = NULL)`
  Returns a tabular view of all evaluated feature sets.
  ArchiveBatchFSelect -> `data.table::data.table()`

  - `x` (ArchiveBatchFSelect)
  - `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 classes

`bbotk::Archive` -> `bbotk::ArchiveBatch` -> `ArchiveBatchFSelect`

Public fields

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

Active bindings

`ties_method` (`character(1)`)
Method to handle ties.

Methods

Public methods:

- `ArchiveBatchFSelect$new()`
- `ArchiveBatchFSelect$add_evals()`
- `ArchiveBatchFSelect$learner()`
- `ArchiveBatchFSelect$learners()`
- `ArchiveBatchFSelect$predictions()`
- `ArchiveBatchFSelect$resample_result()`
- `ArchiveBatchFSelect$print()`
- `ArchiveBatchFSelect$best()`
- `ArchiveBatchFSelect$clone()`

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

Usage:

```
ArchiveBatchFSelect$new(
  search_space,
  codomain,
  check_values = TRUE,
  ties_method = "least_features"
)
```

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.

`ties_method` (`character(1)`)
: The method to break ties when selecting sets while optimizing and when selecting the best set. Can be `"least_features"` or `"random"`. The option `"least_features"` (default) selects the feature set with the least features. If there are multiple best feature sets with the same number of features, one is selected randomly. The `random` method returns a random feature set from the best feature sets. Ignored if multiple measures are used.

Method `add_evals()`**:** Adds function evaluations to the archive table.

Usage:

```
ArchiveBatchFSelect$add_evals(xdt, xss_trafoed = NULL, 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 FSelectInstanceBatchMultiCrit object. Can contain additional columns for extra information.

`xss_trafoed` (`list()`)
: Ignored in feature selection.

`ydt` (`data.table::data.table()`)
: Optimal outcome.

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:

```
ArchiveBatchFSelect$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:

```
ArchiveBatchFSelect$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:

```
ArchiveBatchFSelect$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:

```
ArchiveBatchFSelect$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:

```
ArchiveBatchFSelect$print()
```

Arguments:

`...` (ignored).

Method `best()`: Returns the best scoring feature sets.

Usage:

```
ArchiveBatchFSelect$best(batch = NULL, ties_method = NULL)
```

Arguments:

`batch` (`integer()`)
The batch number(s) to limit the best results to. Default is all batches.

`ties_method` (`character(1)`)
Method to handle ties. If `NULL` (default), the global ties method set during initialization is used. The default global ties method is `least_features` which selects the feature set with the least features. If there are multiple best feature sets with the same number of features, one is selected randomly. The `random` method returns a random feature set from the best feature sets.

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

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

Usage:

```
ArchiveBatchFSelect$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

There are several sections about feature selection in the mlr3book.

- Estimate Model Performance with nested resampling.

The gallery features a collection of case studies and demos about optimization.

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 FSelectInstanceBatchSingleCrit.

`fselector` (FSelector)
  Optimization algorithm.

Active bindings

`archive` ([ArchiveBatchFSelect)
  Returns FSelectInstanceBatchSingleCrit archive.

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

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

`fselect_result` (data.table::data.table)
  Short-cut to `$result` from FSelectInstanceBatchSingleCrit.

`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.

`phash` (`character(1)`)
  Hash (unique identifier) for this partial object, excluding some components which are varied systematically during tuning (parameter values) or feature selection (feature names).

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(
  fselector,
  learner,
  resampling,
  measure = NULL,
  terminator,
```

```
  store_fselect_instance = TRUE,
  store_benchmark_result = TRUE,
  store_models = FALSE,
  check_values = FALSE,
  callbacks = NULL,
  ties_method = "least_features"
)
```

Arguments:

`fselector` (FSelector)
: Optimization algorithm.

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

`store_fselect_instance` (`logical(1)`)
: If `TRUE` (default), stores the internally created FSelectInstanceBatchSingleCrit 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 CallbackBatchFSelect)
: List of callbacks.

`ties_method` (`character(1)`)
: The method to break ties when selecting sets while optimizing and when selecting the best set. Can be `"least_features"` or `"random"`. The option `"least_features"` (default) selects the feature set with the least features. If there are multiple best feature sets with the same number of features, one is selected randomly. The `random` method returns a random feature set from the best feature sets. Ignored if multiple measures are used.

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: mlr3::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(
  fselector = 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(
  fselector = 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(
  fselector,
  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 = NULL,
  ties_method = "least_features"
)
```

Arguments

`fselector` (FSelector)
Optimization algorithm.

`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. Ignored if `terminator` is passed.

`term_time` (`integer(1)`)
Maximum allowed time in seconds. Ignored if `terminator` is passed.

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

`store_fselect_instance`
(`logical(1)`)
If `TRUE` (default), stores the internally created FSelectInstanceBatchSingleCrit 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 CallbackBatchFSelect)
List of callbacks.

ties_method
: (character(1))
The method to break ties when selecting sets while optimizing and when selecting the best set. Can be "least_features" or "random". The option "least_features" (default) selects the feature set with the least features. If there are multiple best feature sets with the same number of features, one is selected randomly. The random method returns a random feature set from the best feature sets. Ignored if multiple measures are used.

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

There are several sections about feature selection in the mlr3book.

- Estimate Model Performance with nested resampling.

The gallery features a collection of case studies and demos about optimization.

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(
  fselector = 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(
  fselector = 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()
```

`CallbackBatchFSelect` *Create Feature Selection Callback*

Description

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

Super classes

`mlr3misc::Callback` -> `bbotk::CallbackBatch` -> `CallbackBatchFSelect`

Public fields

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

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

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

Methods

Public methods:

- `CallbackBatchFSelect$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

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

callback_batch_fselect — *Create Feature Selection Callback*

Description

Function to create a CallbackBatchFSelect. 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::ContextBatch and ContextBatchFSelect.

Usage

```
callback_batch_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()`.
- `on_optimizer_before_eval` — (`function()`)
  Stage called after the optimizer proposes points. Called in `OptimInstance$eval_batch()`.
- `on_eval_after_design` — (`function()`)
  Stage called after design is created. Called in `ObjectiveFSelectBatch$eval_many()`.
- `on_eval_after_benchmark` — (`function()`)
  Stage called after feature sets are evaluated. Called in `ObjectiveFSelectBatch$eval_many()`.
- `on_eval_before_archive` — (`function()`)
  Stage called before performance values are written to the archive. Called in `ObjectiveFSelectBatch$eval_many()`.
- `on_optimizer_after_eval` — (`function()`)
  Stage called after points are evaluated. Called in `OptimInstance$eval_batch()`.
- `on_result` — (`function()`)
  Stage called after result are written. Called in `OptimInstance$assign_result()`.
- `on_optimization_end` — (`function()`)
  Stage called at the end of the optimization. Called in `Optimizer$optimize()`.

Details

When implementing a callback, each function 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 ContextBatchFSelect. 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 ContextBatchFSelect). The other stages access bbotk::ContextBatch. 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 ContextBatchFSelect exists on different workers at the same time.

Examples

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

ContextBatchFSelect *Evaluation Context*

Description

The ContextBatchFSelect allows CallbackBatchFSelects to access and modify data while a batch of feature sets is evaluated. See the section on active bindings for a list of modifiable objects. See `callback_batch_fselect()` for a list of stages that access ContextBatchFSelect.

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 classes

`mlr3misc::Context` -> `bbotk::ContextBatch` -> `ContextBatchFSelect`

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:

- `ContextBatchFSelect$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`ensemble_fselect` *Ensemble Feature Selection*

Description

Ensemble feature selection using multiple learners. The ensemble feature selection method is designed to identify the most informative features from a given dataset by leveraging multiple machine learning models and resampling techniques. Returns an EnsembleFSResult.

Usage

```
ensemble_fselect(
  fselector,
  task,
  learners,
  init_resampling,
  inner_resampling,
  measure,
  terminator,
  callbacks = NULL,
  store_benchmark_result = TRUE,
  store_models = TRUE
)
```

Arguments

`fselector` (FSelector)
Optimization algorithm.

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

`learners` (list of mlr3::Learner)
The learners to be used for feature selection.

`init_resampling` (mlr3::Resampling)
The initial resampling strategy of the data, from which each train set will be passed on to the learners. Can only be mlr3::ResamplingSubsampling or mlr3::ResamplingBootstrap.

`inner_resampling`
: (mlr3::Resampling)
The inner resampling strategy used by the FSelector.

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

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

`callbacks`
: (list of lists of CallbackBatchFSelect)
Callbacks to be used for each learner. The lists must have the same length as the number of learners.

`store_benchmark_result`
: (`logical(1)`)
Whether to store the benchmark result in EnsembleFSResult or not.

`store_models`
: (`logical(1)`)
Whether to store models in auto_fselector or not.

Details

The method begins by applying an initial resampling technique specified by the user, to create **multiple subsamples** from the original dataset. This resampling process helps in generating diverse subsets of data for robust feature selection.

For each subsample generated in the previous step, the method performs **wrapped-based feature selection** (auto_fselector) using each provided learner, the given inner resampling method, performance measure and optimization algorithm. This process generates the best feature subset for each combination of subsample and learner. Results are stored in an EnsembleFSResult.

Value

an EnsembleFSResult object.

Source

Saeys, Yvan, Abeel, Thomas, Van De Peer, Yves (2008). "Robust feature selection using ensemble feature selection techniques." *Machine Learning and Knowledge Discovery in Databases*, **5212 LNAI**, 313–325. doi:10.1007/9783540874812_21.

Abeel, Thomas, Helleputte, Thibault, Van de Peer, Yves, Dupont, Pierre, Saeys, Yvan (2010). "Robust biomarker identification for cancer diagnosis with ensemble feature selection methods." *Bioinformatics*, **26**, 392–398. ISSN 1367-4803, doi:10.1093/BIOINFORMATICS/BTP630.

Pes, Barbara (2020). "Ensemble feature selection for high-dimensional data: a stability analysis across multiple domains." *Neural Computing and Applications*, **32**(10), 5951–5973. ISSN 14333058, doi:10.1007/s00521019040823.

Examples

```
efsr = ensemble_fselect(
  fselector = fs("random_search"),
  task = tsk("sonar"),
  learners = lrns(c("classif.rpart", "classif.featureless")),
```

```
    init_resampling = rsmp("subsampling", repeats = 2),
    inner_resampling = rsmp("cv", folds = 3),
    measure = msr("classif.ce"),
    terminator = trm("evals", n_evals = 10)
  )
  efsr
```

`ensemble_fs_result` *Ensemble Feature Selection Result*

Description

The `EnsembleFSResult` stores the results of ensemble feature selection. It includes methods for evaluating the stability of the feature selection process and for ranking the selected features among others. The function `ensemble_fselect()` returns an object of this class.

S3 Methods

- `as.data.table.EnsembleFSResult(x, benchmark_result = TRUE)`
  Returns a tabular view of the ensemble feature selection.
  EnsembleFSResult -> `data.table::data.table()`

  - `x` (EnsembleFSResult)
  - `benchmark_result` (`logical(1)`)
    Whether to add the learner, task and resampling information from the benchmark result.

Public fields

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

`man` (`character(1)`)
: Manual page for this object.

Active bindings

`result` (data.table::data.table)
: Returns the result of the ensemble feature selection.

`n_learners` (`numeric(1)`)
: Returns the number of learners used in the ensemble feature selection.

`measure` (`character(1)`)
: Returns the measure id used in the ensemble feature selection.

Methods

Public methods:

- `EnsembleFSResult$new()`
- `EnsembleFSResult$format()`
- `EnsembleFSResult$print()`
- `EnsembleFSResult$help()`
- `EnsembleFSResult$feature_ranking()`
- `EnsembleFSResult$stability()`
- `EnsembleFSResult$pareto_front()`
- `EnsembleFSResult$knee_points()`
- `EnsembleFSResult$clone()`

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

Usage:

```
EnsembleFSResult$new(
  result,
  features,
  benchmark_result = NULL,
  measure_id,
  minimize = TRUE
)
```

Arguments:

`result` (data.table::data.table)
The result of the ensemble feature selection. Column names should include `"resampling_iteration"`, `"learner_id"`, `"features"` and `"n_features"`.

`features` (`character()`)
The vector of features of the task that was used in the ensemble feature selection.

`benchmark_result` (mlr3::BenchmarkResult)
The benchmark result object.

`measure_id` (`character(1)`)
Column name of `"result"` that corresponds to the measure used.

`minimize` (`logical(1)`)
If `TRUE` (default), lower values of the measure correspond to higher performance.

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

Usage:

```
EnsembleFSResult$format(...)
```

Arguments:

`...` (ignored).

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

Usage:

```
EnsembleFSResult$print(...)
```

Arguments:

`...` (ignored).

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

Usage:

```
EnsembleFSResult$help()
```

Method `feature_ranking()`**:** Calculates the feature ranking.

Usage:

```
EnsembleFSResult$feature_ranking(method = "approval_voting")
```

Arguments:

`method` (`character(1)`)
The method to calculate the feature ranking.

Details: The feature ranking process is built on the following framework: models act as voters, features act as candidates, and voters select certain candidates (features). The primary objective is to compile these selections into a consensus ranked list of features, effectively forming a committee. Currently, only `"approval_voting"` method is supported, which selects the candidates/features that have the highest approval score or selection frequency, i.e. appear the most often.

Returns: A data.table::data.table listing all the features, ordered by decreasing inclusion probability scores (depending on the `method`)

Method `stability()`**:** Calculates the stability of the selected features with the **stabm** package. The results are cached. When the same stability measure is requested again with different arguments, the cache must be reset.

Usage:

```
EnsembleFSResult$stability(
  stability_measure = "jaccard",
  stability_args = NULL,
  global = TRUE,
  reset_cache = FALSE
)
```

Arguments:

`stability_measure` (`character(1)`)
The stability measure to be used. One of the measures returned by `stabm::listStabilityMeasures()` in lower case. Default is `"jaccard"`.

`stability_args` (`list`)
Additional arguments passed to the stability measure function.

`global` (`logical(1)`)
Whether to calculate the stability globally or for each learner.

`reset_cache` (`logical(1)`)
If `TRUE`, the cached results are ignored.

Returns: A `numeric()` value representing the stability of the selected features. Or a `numeric()` vector with the stability of the selected features for each learner.

Method `pareto_front()`**:** This function identifies the **Pareto front** of the ensemble feature selection process, i.e., the set of points that represent the trade-off between the number of features and performance (e.g. classification error).

Usage:

```
EnsembleFSResult$pareto_front(type = "empirical")
```

Arguments:

`type` (`character(1)`)
Specifies the type of Pareto front to return. See details.

Details: Two options are available for the Pareto front:

- `"empirical"` (default): returns the empirical Pareto front.
- `"estimated"`: the Pareto front points are estimated by fitting a linear model with the inversed of the number of features $(1/x)$ as input and the associated performance scores as output. This method is useful when the Pareto points are sparse and the front assumes a convex shape if better performance corresponds to lower measure values (e.g. classification error), or a concave shape otherwise (e.g. classification accuracy). The `estimated` Pareto front will include points for a number of features ranging from 1 up to the maximum number found in the empirical Pareto front.

Returns: A data.table::data.table with columns the number of features and the performance that together form the Pareto front.

Method `knee_points()`**:** This function implements various *knee* point identification (KPI) methods, which select points in the Pareto front, such that an optimal trade-off between performance and number of features is achieved. In most cases, only one such point is returned.

Usage:

```
EnsembleFSResult$knee_points(method = "NBI", type = "empirical")
```

Arguments:

`method` (`character(1)`)
Type of method to use to identify the knee point. See details.

`type` (`character(1)`)
Specifies the type of Pareto front to use for the identification of the knee point. See `pareto_front()` method for more details.

Details: The available KPI methods are:

- `"NBI"` (default): The **Normal-Boundary Intersection** method is a geometry-based method which calculates the perpendicular distance of each point from the line connecting the first and last points of the Pareto front. The knee point is determined as the Pareto point with the maximum distance from this line, see Das (1999).

Returns: A data.table::data.table with the knee point(s) of the Pareto front.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Das, I (1999). "On characterizing the 'knee' of the Pareto curve based on normal-boundary intersection." *Structural Optimization*, **18**(1-2), 107–115. ISSN 09344373.

Examples

```
efsr = ensemble_fselect(
  fselector = fs("rfe", n_features = 2, feature_fraction = 0.8),
  task = tsk("sonar"),
  learners = lrns(c("classif.rpart", "classif.featureless")),
  init_resampling = rsmp("subsampling", repeats = 2),
  inner_resampling = rsmp("cv", folds = 3),
  measure = msr("classif.ce"),
  terminator = trm("none")
)

# contains the benchmark result
efsr$benchmark_result

# contains the selected features for each iteration
efsr$result

# returns the stability of the selected features
efsr$stability(stability_measure = "jaccard")

# returns a ranking of all features
head(efsr$feature_ranking())

# returns the empirical pareto front (nfeatures vs error)
efsr$pareto_front()
```

---

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(
  fselector = 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(
  fselector = 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 a FSelector from mlr_fselectors.
- trm() for a bbotk::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.
`...`	(any) Additional arguments.
`.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 FSelectInstanceBatchSingleCrit or FSelectInstanceBatchMultiCrit which describes the feature selection problem. It executes the feature selection with the FSelector (`method`) and returns the result with the fselect instance (`$result`). The ArchiveBatchFSelect (`$archive`) stores all evaluated hyperparameter configurations and performance scores.

Usage

```
fselect(
  fselector,
  task,
  learner,
  resampling,
  measures = NULL,
  term_evals = NULL,
  term_time = NULL,
  terminator = NULL,
  store_benchmark_result = TRUE,
  store_models = FALSE,
  check_values = FALSE,
  callbacks = NULL,
  ties_method = "least_features"
)
```

Arguments

`fselector` (FSelector)
Optimization algorithm.

`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 FSelectInstanceBatchSingleCrit and multiple measures a FSelectInstanceBatchMultiCrit. If `NULL`, default measure is used.

`term_evals` (`integer(1)`)
Number of allowed evaluations. Ignored if `terminator` is passed.

`term_time` (`integer(1)`)
Maximum allowed time in seconds. Ignored if `terminator` is passed.

`terminator` (bbotk::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 CallbackBatchFSelect)
List of callbacks.

`ties_method` (`character(1)`)
The method to break ties when selecting sets while optimizing and when selecting the best set. Can be `"least_features"` or `"random"`. The option `"least_features"` (default) selects the feature set with the least features. If there are multiple best feature sets with the same number of features, one is selected randomly. The `random` method returns a random feature set from the best feature sets. Ignored if multiple measures are used.

Details

The mlr3::Task, mlr3::Learner, mlr3::Resampling, mlr3::Measure and bbotk::Terminator are used to construct a FSelectInstanceBatchSingleCrit. If multiple performance Measures are supplied, a FSelectInstanceBatchMultiCrit is created. The parameter `term_evals` and `term_time` are shortcuts to create a bbotk::Terminator. If both parameters are passed, a bbotk::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

FSelectInstanceBatchSingleCrit | FSelectInstanceBatchMultiCrit

Resources

There are several sections about feature selection in the mlr3book.

- Getting started with wrapper feature selection.
- Do a sequential forward selection Palmer Penguins data set.

The gallery features a collection of case studies and demos about optimization.

- Utilize the built-in feature importance of models with Recursive Feature Elimination.
- Run a feature selection with Shadow Variable Search.
- 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(
  fselector = fs("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)
```

`FSelectInstanceBatchMultiCrit` *Class for Multi Criteria Feature Selection*

Description

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

Resources

There are several sections about feature selection in the mlr3book.

- Learn about multi-objective optimization.

The gallery features a collection of case studies and demos about optimization.

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::OptimInstanceBatch` -> `bbotk::OptimInstanceBatchMultiCrit` -> `FSelectInstanceBatchMultiCrit`

Active bindings

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

Methods

Public methods:

- `FSelectInstanceBatchMultiCrit$new()`
- `FSelectInstanceBatchMultiCrit$assign_result()`
- `FSelectInstanceBatchMultiCrit$print()`
- `FSelectInstanceBatchMultiCrit$clone()`

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

Usage:

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

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` (bbotk::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 CallbackBatchFSelect)
List of callbacks.

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

Usage:

```
FSelectInstanceBatchMultiCrit$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 FSelectInstanceBatchMultiCrit object. Can contain additional columns for extra information.

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

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

Usage:

`FSelectInstanceBatchMultiCrit$print(...)`

Arguments:

`...` (ignored).

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

Usage:

`FSelectInstanceBatchMultiCrit$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)
```

`FSelectInstanceBatchSingleCrit`

Class for Single Criterion Feature Selection

Description

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

The instance contains an ObjectiveFSelectBatch 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()`.

Default Measures

If no measure is passed, the default measure is used. The default measure depends on the task type.

Task	Default Measure	Package
`"classif"`	`"classif.ce"`	**mlr3**
`"regr"`	`"regr.mse"`	**mlr3**
`"surv"`	`"surv.cindex"`	**mlr3proba**
`"dens"`	`"dens.logloss"`	**mlr3proba**
`"classif_st"`	`"classif.ce"`	**mlr3spatial**
`"regr_st"`	`"regr.mse"`	**mlr3spatial**
`"clust"`	`"clust.dunn"`	**mlr3cluster**

Resources

There are several sections about feature selection in the mlr3book.

- Getting started with wrapper feature selection.
- Do a sequential forward selection Palmer Penguins data set.

The gallery features a collection of case studies and demos about optimization.

- Utilize the built-in feature importance of models with Recursive Feature Elimination.
- Run a feature selection with Shadow Variable Search.
- 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::OptimInstanceBatch` -> `bbotk::OptimInstanceBatchSingleCrit` -> `FSelectInstanceBatchSingleCrit`

Active bindings

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

Methods

Public methods:

- `FSelectInstanceBatchSingleCrit$new()`
- `FSelectInstanceBatchSingleCrit$assign_result()`
- `FSelectInstanceBatchSingleCrit$print()`
- `FSelectInstanceBatchSingleCrit$clone()`

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

Usage:

```
FSelectInstanceBatchSingleCrit$new(
  task,
  learner,
  resampling,
  measure,
  terminator,
  store_benchmark_result = TRUE,
  store_models = FALSE,
  check_values = FALSE,
  callbacks = NULL,
  ties_method = "least_features"
)
```

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` (bbotk::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 CallbackBatchFSelect)
List of callbacks.

`ties_method` (`character(1)`)
The method to break ties when selecting sets while optimizing and when selecting the best set. Can be `"least_features"` or `"random"`. The option `"least_features"` (default) selects the feature set with the least features. If there are multiple best feature sets with the same number of features, one is selected randomly. The `random` method returns a random feature set from the best feature sets. Ignored if multiple measures are used.

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

Usage:

```
FSelectInstanceBatchSingleCrit$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 FSelectInstanceBatchMultiCrit object. Can contain additional columns for extra information.

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

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

Usage:

```
FSelectInstanceBatchSingleCrit$print(...)
```

Arguments:

`...` (ignored).

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

Usage:

```
FSelectInstanceBatchSingleCrit$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` *FSelector*

Description

The ‘FSelector“ implements the optimization algorithm.

Details

`FSelector` is an abstract base class that implements the base functionality each fselector must provide.

Resources

There are several sections about feature selection in the mlr3book.

- Learn more about fselectors.

The gallery features a collection of case studies and demos about optimization.

- Utilize the built-in feature importance of models with Recursive Feature Elimination.
- Run a feature selection with Shadow Variable Search.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other FSelector: `mlr_fselectors`, `mlr_fselectors_design_points`, `mlr_fselectors_exhaustive_search`, `mlr_fselectors_genetic_search`, `mlr_fselectors_random_search`, `mlr_fselectors_rfe`, `mlr_fselectors_rfecv`, `mlr_fselectors_sequential`, `mlr_fselectors_shadow_variable_search`

FSelectorBatch *Class for Batch Feature Selection Algorithms*

Description

The FSelectorBatch implements the optimization algorithm.

Details

FSelectorBatch is an abstract base class that implements the base functionality each fselector must provide. A subclass is implemented in the following way:

- Inherit from FSelectorBatch.
- 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 FSelectInstanceBatchSingleCrit/FSelectInstanceBatchMultiCrit object `instance`, so each batch is possibly executed in parallel via `mlr3::benchmark()`, and all evaluations are stored inside of `instance$archive`.
- Before the batch evaluation, the bbotk::Terminator is checked, and if it is positive, an exception of class `"terminated_error"` is generated. In the latter case the current batch of evaluations is still stored in `instance`, but the numeric scores are not sent back to the handling optimizer as it has lost execution control.
- After such an exception was caught we select the best set from `instance$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 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

There are several sections about feature selection in the mlr3book.

- Learn more about fselectors.

The gallery features a collection of case studies and demos about optimization.

- Utilize the built-in feature importance of models with Recursive Feature Elimination.
- Run a feature selection with Shadow Variable Search.

Super class

`mlr3fselect::FSelector` -> `FSelectorBatch`

Methods

Public methods:

- `FSelectorBatch$new()`
- `FSelectorBatch$optimize()`
- `FSelectorBatch$clone()`

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

Usage:

```
FSelectorBatch$new(
  id = "fselector_batch",
  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 `optimize()`: Performs the feature selection on a FSelectInstanceBatchSingleCrit or FSelectInstanceBatchMultiCrit until termination. The single evaluations will be written into the ArchiveBatchFSelect that resides in the FSelectInstanceBatchSingleCrit / FSelectInstanceBatch-MultiCrit. The result will be written into the instance object.

Usage:

```
FSelectorBatch$optimize(inst)
```

Arguments:

`inst` (FSelectInstanceBatchSingleCrit | FSelectInstanceBatchMultiCrit).

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

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

Usage:

```
FSelectorBatch$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(
  fselector,
  task,
  learner,
  inner_resampling,
  outer_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 = NULL,
  ties_method = "least_features"
)
```

Arguments

`fselector`
: (FSelector)
Optimization algorithm.

`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. Ignored if `terminator` is passed.

`term_time`
: (`integer(1)`)
Maximum allowed time in seconds. Ignored if `terminator` is passed.

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

`store_fselect_instance`
: (`logical(1)`)
If `TRUE` (default), stores the internally created FSelectInstanceBatchSingleCrit 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 CallbackBatchFSelect)
List of callbacks.

`ties_method`
: (`character(1)`)
The method to break ties when selecting sets while optimizing and when selecting the best set. Can be `"least_features"` or `"random"`. The option `"least_features"` (default) selects the feature set with the least features. If there are multiple best feature sets with the same number of features, one is selected randomly. The `random` method returns a random feature set from the best feature sets. Ignored if multiple measures are used.

Value

mlr3::ResampleResult

Examples

```
# Nested resampling on Palmer Penguins data set
rr = fselect_nested(
  fselector = fs("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 FSelectInstanceBatchSingleCrit or FSelectInstanceBatchMultiCrit.

Usage

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

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 FSelectInstanceBatchSingleCrit and multiple measures a FSelectInstanceBatchMultiCrit. If `NULL`, default measure is used.
- `terminator` (bbotk::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 CallbackBatchFSelect)
  List of callbacks.
- `ties_method` (`character(1)`)
  The method to break ties when selecting sets while optimizing and when selecting the best set. Can be `"least_features"` or `"random"`. The option `"least_features"` (default) selects the feature set with the least features. If there are multiple best feature sets with the same number of features, one is selected randomly. The `random` method returns a random feature set from the best feature sets. Ignored if multiple measures are used.

Resources

There are several sections about feature selection in the mlr3book.

- Getting started with wrapper feature selection.
- Do a sequential forward selection Palmer Penguins data set.

The gallery features a collection of case studies and demos about optimization.

- Utilize the built-in feature importance of models with Recursive Feature Elimination.
- Run a feature selection with Shadow Variable Search.
- Feature Selection on the Titanic data set.

Default Measures

If no measure is passed, the default measure is used. The default measure depends on the task type.

Task	Default Measure	Package
`"classif"`	`"classif.ce"`	**mlr3**
`"regr"`	`"regr.mse"`	**mlr3**
`"surv"`	`"surv.cindex"`	**mlr3proba**

"dens"	"dens.logloss"	**mlr3proba**
"classif_st"	"classif.ce"	**mlr3spatial**
"regr_st"	"regr.mse"	**mlr3spatial**
"clust"	"clust.dunn"	**mlr3cluster**

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 CallbackBatchFSelect 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(
  fselector = fs("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")))
```

---

mlr3fselect.one_se_rule

One Standard Error Rule Callback

Description

Selects the smallest feature set within one standard error of the best as the result. If there are multiple such feature sets with the same number of features, the first one is selected. If the sets have exactly the same performance but different number of features, the one with the smallest number of features is selected.

Source

Kuhn, Max, Johnson, Kjell (2013). "Applied Predictive Modeling." In chapter Over-Fitting and Model Tuning, 61–92. Springer New York, New York, NY. ISBN 978-1-4614-6849-3.

Examples

```
clbk("mlr3fselect.one_se_rule")

# Run feature selection on the pima data set with the callback
instance = fselect(
  fselector = fs("random_search"),
  task = tsk("pima"),
  learner = lrn("classif.rpart"),
  resampling = rsmp ("cv", folds = 3),
  measures = msr("classif.ce"),
  term_evals = 10,
  callbacks = clbk("mlr3fselect.one_se_rule"))
# Smallest feature set within one standard error of the best
instance$result
```

mlr3fselect.svm_rfe *SVM-RFE Callback*

Description

Runs a recursive feature elimination with a mlr3learners::LearnerClassifSVM. The SVM must be configured with `type = "C-classification"` and `kernel = "linear"`.

Source

Guyon I, Weston J, Barnhill S, Vapnik V (2002). "Gene Selection for Cancer Classification using Support Vector Machines." *Machine Learning*, **46**(1), 389–422. ISSN 1573-0565, doi:10.1023/A:1012487302797.

Examples

```
clbk("mlr3fselect.svm_rfe")

library(mlr3learners)

# Create instance with classification svm with linear kernel
instance = fsi(
  task = tsk("sonar"),
  learner = lrn("classif.svm", type = "C-classification", kernel = "linear"),
  resampling = rsmp("cv", folds = 3),
  measures = msr("classif.ce"),
  terminator = trm("none"),
  callbacks = clbk("mlr3fselect.svm_rfe"),
  store_models = TRUE
)

fselector = fs("rfe", feature_number = 5, n_features = 10)

# Run recursive feature elimination on the Sonar data set
fselector$optimize(instance)
```

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: `FSelector`, `mlr_fselectors_design_points`, `mlr_fselectors_exhaustive_search`, `mlr_fselectors_genetic_search`, `mlr_fselectors_random_search`, `mlr_fselectors_rfe`, `mlr_fselectors_rfecv`, `mlr_fselectors_sequential`, `mlr_fselectors_shadow_variable_search`

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::FSelectorBatch` -> `mlr3fselect::FSelectorBatchFromOptimizerBatch` -> `FSelectorBatchDesignPoints`

Methods

Public methods:

- `FSelectorBatchDesignPoints$new()`
- `FSelectorBatchDesignPoints$clone()`

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

Usage:

```
FSelectorBatchDesignPoints$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other FSelector: `FSelector`, `mlr_fselectors`, `mlr_fselectors_exhaustive_search`, `mlr_fselectors_genetic_search`, `mlr_fselectors_random_search`, `mlr_fselectors_rfe`, `mlr_fselectors_rfecv`, `mlr_fselectors_sequential`, `mlr_fselectors_shadow_variable_search`

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(
  fselector = 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 classes

mlr3fselect::FSelector -> mlr3fselect::FSelectorBatch -> FSelectorBatchExhaustiveSearch

Methods

Public methods:

- FSelectorBatchExhaustiveSearch$new()
- FSelectorBatchExhaustiveSearch$clone()

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

Usage:

```
FSelectorBatchExhaustiveSearch$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

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

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(
  fselector = fs("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 classes

`mlr3fselect::FSelector` -> `mlr3fselect::FSelectorBatch` -> `FSelectorBatchGeneticSearch`

Methods

Public methods:

- `FSelectorBatchGeneticSearch$new()`
- `FSelectorBatchGeneticSearch$clone()`

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

Usage:

```
FSelectorBatchGeneticSearch$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other FSelector: FSelector, mlr_fselectors, mlr_fselectors_design_points, mlr_fselectors_exhaustive_search, mlr_fselectors_random_search, mlr_fselectors_rfe, mlr_fselectors_rfecv, mlr_fselectors_sequential, mlr_fselectors_shadow_variable_search

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(
  fselector = fs("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 classes

mlr3fselect::FSelector -> mlr3fselect::FSelectorBatch -> FSelectorBatchRandomSearch

Methods

Public methods:

- FSelectorBatchRandomSearch$new()
- FSelectorBatchRandomSearch$clone()

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

Usage:

```
FSelectorBatchRandomSearch$new()
```

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

Usage:

```
FSelectorBatchRandomSearch$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: FSelector, mlr_fselectors, mlr_fselectors_design_points, mlr_fselectors_exhaustive_search, mlr_fselectors_genetic_search, mlr_fselectors_rfe, mlr_fselectors_rfecv, mlr_fselectors_sequential, mlr_fselectors_shadow_variable_search

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(
  fselector = 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 (RFE) algorithm. Recursive feature elimination iteratively removes features with a low importance score. Only works with mlr3::Learners that can calculate importance scores (see the section on optional extractors in mlr3::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.

When using a cross-validation resampling strategy, the importance scores of the resampling iterations are aggregated. The parameter `aggregation` determines how the importance scores are aggregated. By default ("rank"), the importance score vector of each fold is ranked and the feature with the lowest average rank is removed. The option "mean" averages the score of each feature across the resampling iterations and removes the feature with the lowest average score. Averaging the scores is not appropriate for most importance measures.

Archive

The ArchiveBatchFSelect holds the following additional columns:

- "importance" (numeric())
  The importance score vector of the feature subset.

Resources

The gallery features a collection of case studies and demos about optimization.

- Utilize the built-in feature importance of models with Recursive Feature Elimination.

Dictionary

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

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

Control Parameters

n_features integer(1)
: The minimum number of features to select, by default half of the features.

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

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

subset_sizes integer()
: Vector of the 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.

aggregation character(1)
: The aggregation method for the importance scores of the resampling iterations. See details.

The parameter feature_fraction, feature_number and subset_sizes are mutually exclusive.

Super classes

mlr3fselect::FSelector -> mlr3fselect::FSelectorBatch -> FSelectorBatchRFE

Methods

Public methods:

- FSelectorBatchRFE$new()
- FSelectorBatchRFE$clone()

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

Usage:

```
FSelectorBatchRFE$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Source

Guyon I, Weston J, Barnhill S, Vapnik V (2002). "Gene Selection for Cancer Classification using Support Vector Machines." *Machine Learning*, **46**(1), 389–422. ISSN 1573-0565, doi:10.1023/A:1012487302797.

See Also

Other FSelector: FSelector, mlr_fselectors, mlr_fselectors_design_points, mlr_fselectors_exhaustive_search, mlr_fselectors_genetic_search, mlr_fselectors_random_search, mlr_fselectors_rfecv, mlr_fselectors_sequential, mlr_fselectors_shadow_variable_search

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(
  fselector = 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_rfecv` *Feature Selection with Recursive Feature Elimination with Cross Validation*

Description

Feature selection using the Recursive Feature Elimination with Cross-Validation (RFE-CV) algorithm. See FSelectorBatchRFE for a description of the base algorithm. RFE-CV runs a recursive feature elimination in each iteration of a cross-validation to determine the optimal number of features. Then a recursive feature elimination is run again on the complete dataset with the optimal number of features as the final feature set size. The performance of the optimal feature set is calculated on the complete data set and should not be reported as the performance of the final model. Only works with mlr3::Learners that can calculate importance scores (see the section on optional extractors in mlr3::Learner).

Details

The resampling strategy is changed during the feature selection. The resampling strategy passed to the instance (`resampling`) is used to determine the optimal number of features. Usually, a cross-validation strategy is used and a recursive feature elimination is run in each iteration of the cross-validation. Internally, mlr3::ResamplingCustom is used to emulate this part of the algorithm. In the final recursive feature elimination run the resampling strategy is changed to mlr3::ResamplingInsample i.e. the complete data set is used for training and testing.

The feature selection terminates itself when the optimal number of features is reached. It is not necessary to set a termination criterion.

Archive

The ArchiveBatchFSelect holds the following additional columns:

- `"iteration"` (`integer(1)`)
  The resampling iteration in which the feature subset was evaluated.
- `"importance"` (`numeric()`)
  The importance score vector of the feature subset.

Resources

The gallery features a collection of case studies and demos about optimization.

- Utilize the built-in feature importance of models with Recursive Feature Elimination.

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 classes

`mlr3fselect::FSelector` -> `mlr3fselect::FSelectorBatch` -> `FSelectorBatchRFECV`

Methods

Public methods:

- `FSelectorBatchRFECV$new()`
- `FSelectorBatchRFECV$clone()`

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

Usage:

```
FSelectorBatchRFECV$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

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

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(
  fselector = fs("rfecv"),
  task = task,
  learner = learner,
  resampling = rsmp("cv", folds = 3),
  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 model's 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 classes

`mlr3fselect::FSelector` -> `mlr3fselect::FSelectorBatch` -> `FSelectorBatchSequential`

Methods

Public methods:

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

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

Usage:

```
FSelectorBatchSequential$new()
```

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

Usage:

```
FSelectorBatchSequential$optimization_path(inst, include_uhash = FALSE)
```

Arguments:

`inst` (FSelectInstanceBatchSingleCrit)
: Instance optimized with FSelectorBatchSequential.

`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:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other FSelector: `FSelector`, `mlr_fselectors`, `mlr_fselectors_design_points`, `mlr_fselectors_exhaustive_search`, `mlr_fselectors_genetic_search`, `mlr_fselectors_random_search`, `mlr_fselectors_rfe`, `mlr_fselectors_rfecv`, `mlr_fselectors_shadow_variable_search`

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(
  fselector = fs("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.

Resources

The gallery features a collection of case studies and demos about optimization.

- Run a feature selection with Shadow Variable Search.

Dictionary

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

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

Super classes

`mlr3fselect::FSelector` -> `mlr3fselect::FSelectorBatch` -> `FSelectorBatchShadowVariableSearch`

Methods

Public methods:

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

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

Usage:

```
FSelectorBatchShadowVariableSearch$new()
```

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

Usage:

```
FSelectorBatchShadowVariableSearch$optimization_path(inst)
```

Arguments:

`inst` (FSelectInstanceBatchSingleCrit)
Instance optimized with FSelectorBatchShadowVariableSearch.

Returns: data.table::data.table

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

Usage:

```
FSelectorBatchShadowVariableSearch$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: FSelector, mlr_fselectors, mlr_fselectors_design_points, mlr_fselectors_exhaustive_search, mlr_fselectors_genetic_search, mlr_fselectors_random_search, mlr_fselectors_rfe, mlr_fselectors_rfecv, mlr_fselectors_sequential

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(
  fselector = 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 the FSelectInstanceBatchSingleCrit / FSelectInstanceBatchMultiCrit.

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)`).

`callbacks` (List of CallbackBatchFSelects).

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,
  callbacks = NULL
)
```

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?

`callbacks` (list of CallbackBatchFSelect)
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.

`ObjectiveFSelectBatch` *Class for Feature Selection Objective*

Description

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

Super classes

`bbotk::Objective` -> `mlr3fselect::ObjectiveFSelect` -> `ObjectiveFSelectBatch`

Public fields

`archive` (ArchiveBatchFSelect).

Methods

Public methods:

- `ObjectiveFSelectBatch$new()`
- `ObjectiveFSelectBatch$clone()`

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

Usage:

```
ObjectiveFSelectBatch$new(
  task,
  learner,
  resampling,
  measures,
  check_values = TRUE,
  store_benchmark_result = TRUE,
  store_models = FALSE,
  archive = NULL,
  callbacks = NULL
)
```

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` (ArchiveBatchFSelect)
Reference to the archive of FSelectInstanceBatchSingleCrit | FSelectInstanceBatchMultiCrit. If `NULL` (default), benchmark result and models cannot be stored.

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Index