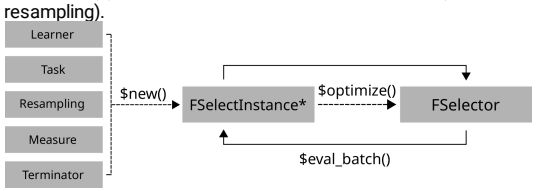
Feature Selection with mlr3fselect::CHEAT SHEET

Class Overview

The package provides a set of R6 classes which allow to (a) define general feature selection instances; (b) run black-box optimzers; (c) combine learners with feature selection (for nested resampling).



[NB: In many table prints we suppres cols for readability.]

Terminators - When to stop

Construction: `trm(.key, ...)`

- `evals` (`n_evals`)
  After iterations.
- `run_time` (`secs`)
  After training time.
- `clock_time` (`stop_time`)
  At given timepoint.
- `perf_reached` (`level`)
  After performance was reached.
- `stagnation` (`iters`, `threshold`)
  After performance stagnated.
- `stagnation_batch` (n, `threshold`)
  After performance stagnated for batches.
- `combo` (list_of_terms, `any=TRUE`)
  Combine terminators with AND or OR.

```
as.data.table(mlr_terminators) # list all
```

Lists all available terminators.

FSelectInstance* - Search Scenario

Evaluator and container for resampled performances of feature subsets. The (internal) function `eval_batch(xdt)` calls `benchmark()` to evaluate a table of feature subsets. Stores archive of all evaluated feature subsets and the final result.

```
instance = FSelectInstanceSingleCrit$new(task,
  learner, resampling, measure, terminator)
```

`store_benchmark_result = TRUE` to store resampled evals and `store_models = TRUE` for fitted models.

Example

```
instance = FSelectInstanceSingleCrit$new(task, learner, resampling, measure,
  terminator)
fselector = fs("random_search", batch_size = 10)
fselector$optimize(instance)
instance$result
# >    Petal.Length Petal.Width Sepal.Length Sepal.Width classif.ce
# > 1:        FALSE        TRUE         TRUE        TRUE       0.06
```

Use `FSelectInstanceMultiCrit` for multi-criteria feature selection.

FSelector - Search Strategy

Generates feature subsets and passes to instance for evaluation until termination. Creation: `fs(.key, ...)`

- `random_search` (`batch_size`)
  Random search.
- `exhaustive_search` (`max_features`)
  Exhaustive Search.
- `sequential` (`strategy`)
  Sequential Selection.
- `rfe` (`feature_fraction`, `recursive`)
  Recursive Feature Elimination.
- `design_points` (`batch_size`, `design`)
  User supplied feature subsets.

```
as.data.table(mlr_fselectors) # list all
```

Lists all available feature selection algorithms.

Logging and Parallelization

```
lgr::get_logger("bbotk")$set_threshold("<level>")
```

Change log-level only for mlr3fselect.

```
future::plan(strategy)
```

Sets the parallelization backend. Speeds up feature selection by running iterations in parallel.

Executing the Feature Selection

```
fselector$optimize(instance)
as.data.table(instance$archive)
## >    Petal.Length Petal.Width Sepal.Length Sepal.Width classif.ce
## > 1:         TRUE        TRUE         TRUE        TRUE 0.09333333
## > 2:         TRUE        TRUE         TRUE       FALSE 0.09333333
instance$result # datatable row with optimal feature subset and estimated perf
```

Get evaluated feature subsets and performances; and result.

```
task$select(instance$result_feature_set)
```

Set optimized feature subset in `Task`.

Example

```
instance = fselect(method = "random_search", task = tsk("iris"), learner = learner,
  resampling = rsmp ("holdout"), measure = msr("classif.ce"), term_evals = 20)
```

Use `fselect()`-shortcut.

AutoFSelector - Select before Train

Wraps learner and performs integrated feature selection.

```
afs = AutoFSelector$new(learner, resampling,
  measure, terminator, fselector)
```

Inherits from class `Learner`. Training starts feature selection on the training set. After completion the learner is trained with the "optimal" feature subset on the given task.

```
afs$train(task)
afs$predict(task, row_ids)
```

```
afs$learner
```

Returns learner trained on full data set with optimized feature subset.

```
afs$fselect_result
# >    Petal.Width Sepal.Length Sepal.Width classif.ce
# > 1:        TRUE         TRUE        TRUE       0.02
```

Access feature selection result.

```
afs = auto_fselector(method = "random_search",
  learner, resampling, measure, term_evals = 20)
```

Use shortcut to create `AutoFSelector`.

Nested Resampling

Just resample `AutoFSelector`; now has inner and outer loop.

Example

```
inner = rsmp("holdout")
afs = auto_fselector(method = "random_search", learner, inner, measure, term_evals =
  20)
outer = rsmp("cv", folds = 2)
rr = resample(task, afs, outer, store_models = TRUE)

as.data.table(rr)
## >             learner         resampling iteration              prediction
## > 1: <AutoFSelector[38]> <ResamplingCV[19]>         1 <PredictionClassif[19]>
## > 2: <AutoFSelector[38]> <ResamplingCV[19]>         2 <PredictionClassif[19]>
```

```
extract_inner_fselect_results(rr)
# >    iteration Petal.Width Sepal.Length Sepal.Width classif.ce
# > 1:         1        TRUE         TRUE       FALSE       0.04
# > 2:         2        TRUE         TRUE       FALSE       0.00
```

Check inner results for stable features.

```
rr$score()
# >              learner iteration             prediction classif.ce
# > 1: <AutoFSelector[40]>         1 <PredictionClassif[19]> 0.02666667
# > 2: <AutoFSelector[40]>         2 <PredictionClassif[19]> 0.08000000
```

Predictive performances estimated on the outer resampling.

```
extract_inner_fselect_archives(rr)
# >    iteration Petal.Width Sepal.Length Sepal.Width classif.ce
# > 1:         1       FALSE         TRUE       FALSE       0.36
# > 21:        2       FALSE         TRUE       FALSE       0.44
```

All evaluated feature subsets.

```
rr$aggregate()
# > classif.ce
# > 0.05333333
```

Aggregates performances of outer resampling iterations.

```
rr = fselect_nested(method = "random_search", task,
  learner, inner, outer, measure, term_evals = 20)
```

Use shortcut to execute nested resampling.