Package 'mcboost'

January 18, 2022

Type Package

Title Multi-Calibration Boosting

Version 0.4.0

Maintainer Florian Pfisterer <pfistererf@googlemail.com>

Description Implements 'Multi-Calibration Boosting' (2018) <https://proceedings.mlr.press/v80/hebert-johnson18a.html> and 'Multi-Accuracy Boosting' (2019) <arXiv:1805.12317> for the multi-calibration of a machine learning model's prediction.
'MCBoost' updates predictions for sub-groups in an iterative fashion in order to mitigate biases like poor calibration or large accuracy differences across subgroups.
Multi-Calibration works best in scenarios where the underlying data & labels are unbiased, but resulting models are.
This is often the case, e.g. when an algorithm fits a majority population while ignoring or underfitting minority populations.

License LGPL (>= 3)

URL https://github.com/mlr-org/mcboost

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

Encoding UTF-8

Depends R (>= 3.1.0)

Imports backports, checkmate (>= 2.0.0), data.table (>= 1.13.6), mlr3 (>= 0.10), mlr3misc (>= 0.8.0), mlr3pipelines (>= 0.3.0), mlr3proba (>= 0.4.0), R6 (>= 2.4.1), rpart, glmnet

Suggests curl, lgr, formattable, tidyverse, PracTools, mlr3learners, mlr3oml, neuralnet, paradox, testthat, knitr, ranger, rmarkdown, survival, xgboost, covr

RoxygenNote 7.1.2

VignetteBuilder knitr

Collate 'AuditorFitters.R' 'MCBoost.R' 'MCBoostSurv.R' 'PipelineMCBoost.R' 'PipeOpLearnerPred.R' 'PipeOpMCBoost.R' 'PipeOpMCBoostSurv.R' 'Predictor.R' 'ProbRange.R' 'ProbRange2D.R' 'helpers.R' 'zzz.R'

NeedsCompilation no

Author Florian Pfisterer [cre, aut] (<https://orcid.org/0000-0001-8867-762X>),
Susanne Dandl [ctb] (<https://orcid.org/0000-0003-4324-4163>),
Christoph Kern [ctb] (<https://orcid.org/0000-0001-7363-4299>),
Carolin Becker [ctb],
Bernd Bischl [ctb] (<https://orcid.org/0000-0001-6002-6980>)

Repository CRAN

Date/Publication 2022-01-18 17:52:43 UTC

R topics documented:

`mcboost-package` *mcboost: Multi-Calibration Boosting*

Description

Implements 'Multi-Calibration Boosting' (2018) <https://proceedings.mlr.press/v80/hebert-johnson18a.html> and 'Multi-Accuracy Boosting' (2019) <arXiv:1805.12317> for the multi-calibration of a machine learning model's prediction. 'MCBoost' updates predictions for sub-groups in an iterative fashion in order to mitigate biases like poor calibration or large accuracy differences across subgroups. Multi-Calibration works best in scenarios where the underlying data & labels are unbiased, but resulting models are. This is often the case, e.g. when an algorithm fits a majority population while ignoring or under-fitting minority populations.

Author(s)

Maintainer: Florian Pfisterer <pfistererf@googlemail.com> (ORCID)

Other contributors:

- Susanne Dandl <susanne.dandl@stat.uni-muenchen.de> (ORCID) [contributor]
- Christoph Kern <c.kern@uni-mannheim.de> (ORCID) [contributor]
- Carolin Becker [contributor]
- Bernd Bischl <bernd_bischl@gmx.net> (ORCID) [contributor]

References

Kim et al., 2019: Multiaccuracy: Black-Box Post-Processing for Fairness in Classification. Hebert-Johnson et al., 2018: Multicalibration: Calibration for the (Computationally-Identifiable) Masses. Pfisterer F, Kern C, Dandl S, Sun M, Kim M, Bischl B (2021). "mcboost: Multi-Calibration Boosting for R." *Journal of Open Source Software*, **6**(64), 3453. doi: 10.21105/joss.03453, https://joss.theoj.org/papers/10.21105/joss.03453.

See Also

Useful links:

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

`AuditorFitter` *AuditorFitter Abstract Base Class*

Description

Defines an `AuditorFitter` abstract base class.

Value

`list` with items

- `corr`: pseudo-correlation between residuals and learner prediction.
- `l`: the trained learner.

Methods

Public methods:

- AuditorFitter$new()
- AuditorFitter$fit_to_resid()
- AuditorFitter$fit()
- AuditorFitter$clone()

Method `new()`**:** Initialize a `AuditorFitter`. This is an abstract base class.

Usage:

```
AuditorFitter$new()
```

Method `fit_to_resid()`**:** Fit to residuals.

Usage:

```
AuditorFitter$fit_to_resid(data, resid, mask)
```

Arguments:

`data` `data.table`
: Features.

`resid` `numeric`
: Residuals (of same length as data).

`mask` `integer`
: Mask applied to the data. Only used for `SubgroupAuditorFitter`.

Method `fit()`**:** Fit (mostly used internally, use `fit_to_resid`).

Usage:

```
AuditorFitter$fit(data, resid, mask)
```

Arguments:

`data` `data.table`
: Features.

`resid` `numeric`
: Residuals (of same length as data).

`mask` `integer`
: Mask applied to the data. Only used for `SubgroupAuditorFitter`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

CVLearnerAuditorFitter

Cross-validated AuditorFitter from a Learner

Description

CVLearnerAuditorFitter returns the cross-validated predictions instead of the in-sample predictions.

Available data is cut into complementary subsets (folds). For each subset out-of-sample predictions are received by training a model on all other subsets and predicting afterwards on the left-out subset.

Value

AuditorFitter

list with items

- corr: pseudo-correlation between residuals and learner prediction.
- l: the trained learner.

Functions

- CVTreeAuditorFitter: Cross-Validated auditor based on rpart
- CVRidgeAuditorFitter: Cross-Validated auditor based on glmnet

Super class

mcboost::AuditorFitter -> CVLearnerAuditorFitter

Public fields

learner CVLearnerPredictor
: Learner used for fitting residuals.

Methods

Public methods:

- CVLearnerAuditorFitter$new()
- CVLearnerAuditorFitter$fit()
- CVLearnerAuditorFitter$clone()

Method new(): Define a CVAuditorFitter from a learner. Available instantiations: CVTreeAuditorFitter (rpart) and CVRidgeAuditorFitter (glmnet). See mlr3pipelines::PipeOpLearnerCV for more information on cross-validated learners.

Usage:

```
CVLearnerAuditorFitter$new(learner, folds = 3L)
```

Arguments:

`learner` `mlr3::Learner`
Regression Learner to use.

`folds` `integer`
Number of folds to use for PipeOpLearnerCV. Defaults to 3.

Method `fit()`**:** Fit the cross-validated learner and compute correlation

Usage:

```
CVLearnerAuditorFitter$fit(data, resid, mask)
```

Arguments:

`data` `data.table`
Features.

`resid` `numeric`
Residuals (of same length as data).

`mask` `integer`
Mask applied to the data. Only used for `SubgroupAuditorFitter`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`mcboost::AuditorFitter` -> `mcboost::CVLearnerAuditorFitter` -> `CVTreeAuditorFitter`

Methods

Public methods:

- `CVTreeAuditorFitter$new()`
- `CVTreeAuditorFitter$clone()`

Method `new()`**:** Define a cross-validated AuditorFitter from a rpart learner See `mlr3pipelines::PipeOpLearnerCV` for more information on cross-validated learners.

Usage:

```
CVTreeAuditorFitter$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

mcboost::AuditorFitter -> mcboost::CVLearnerAuditorFitter -> CVRidgeAuditorFitter

Methods

Public methods:

- CVRidgeAuditorFitter$new()
- CVRidgeAuditorFitter$clone()

Method new(): Define a cross-validated AuditorFitter from a glmnet learner. See mlr3pipelines::PipeOpLearnerCV for more information on cross-validated learners.

Usage:

```
CVRidgeAuditorFitter$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

Other AuditorFitter: LearnerAuditorFitter, SubgroupAuditorFitter, SubpopAuditorFitter

Other AuditorFitter: LearnerAuditorFitter, SubgroupAuditorFitter, SubpopAuditorFitter

Other AuditorFitter: LearnerAuditorFitter, SubgroupAuditorFitter, SubpopAuditorFitter

---

LearnerAuditorFitter *Create an AuditorFitter from a Learner*

Description

Instantiates an AuditorFitter that trains a mlr3::Learner on the data.

Value

AuditorFitter

list with items

- corr: pseudo-correlation between residuals and learner prediction.
- l: the trained learner.

Functions

- `TreeAuditorFitter`: Learner auditor based on rpart
- `RidgeAuditorFitter`: Learner auditor based on glmnet

Super class

`mcboost::AuditorFitter` -> `LearnerAuditorFitter`

Public fields

`learner LearnerPredictor`
Learner used for fitting residuals.

Methods

Public methods:

- `LearnerAuditorFitter$new()`
- `LearnerAuditorFitter$fit()`
- `LearnerAuditorFitter$clone()`

Method `new()`**:** Define an `AuditorFitter` from a Learner. Available instantiations: `TreeAuditorFitter` (rpart) and `RidgeAuditorFitter` (glmnet).

Usage:

```
LearnerAuditorFitter$new(learner)
```

Arguments:

`learner mlr3::Learner`
Regression learner to use.

Method `fit()`**:** Fit the learner and compute correlation

Usage:

```
LearnerAuditorFitter$fit(data, resid, mask)
```

Arguments:

`data data.table`
Features.

`resid numeric`
Residuals (of same length as data).

`mask integer`
Mask applied to the data. Only used for `SubgroupAuditorFitter`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

mcboost::AuditorFitter -> mcboost::LearnerAuditorFitter -> TreeAuditorFitter

Methods

Public methods:

- TreeAuditorFitter$new()
- TreeAuditorFitter$clone()

Method `new()`**:** Define a AuditorFitter from a rpart learner.

Usage:

```
TreeAuditorFitter$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

mcboost::AuditorFitter -> mcboost::LearnerAuditorFitter -> RidgeAuditorFitter

Methods

Public methods:

- RidgeAuditorFitter$new()
- RidgeAuditorFitter$clone()

Method `new()`**:** Define a AuditorFitter from a glmnet learner.

Usage:

```
RidgeAuditorFitter$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other AuditorFitter: CVLearnerAuditorFitter, SubgroupAuditorFitter, SubpopAuditorFitter

Other AuditorFitter: CVLearnerAuditorFitter, SubgroupAuditorFitter, SubpopAuditorFitter

Other AuditorFitter: CVLearnerAuditorFitter, SubgroupAuditorFitter, SubpopAuditorFitter

`make_survival_curve` *Make every row monotonically decreasing in order to obtain the survival property. Additionally, many predicitions need 1 as a first value and 0 as a last value. (e.g.* `PredictionSurv` *needs this attribute.)*

Description

Make every row monotonically decreasing in order to obtain the survival property. Additionally, many predicitions need 1 as a first value and 0 as a last value. (e.g. `PredictionSurv` needs this attribute.)

Usage

```
make_survival_curve(prediction)
```

Arguments

`prediction`	`data.table` Data.table with predictions. Every row is survival probability for the corresponding time. Every column corresponds to a specific time point.

Value

`data.table`

`MCBoost` *Multi-Calibration Boosting*

Description

Implements Multi-Calibration Boosting by Hebert-Johnson et al. (2018) and Multi-Accuracy Boosting by Kim et al. (2019) for the multi-calibration of a machine learning model's prediction. Multi-Calibration works best in scenarios where the underlying data & labels are unbiased but a bias is introduced within the algorithm's fitting procedure. This is often the case, e.g. when an algorithm fits a majority population while ignoring or under-fitting minority populations.
Expects initial models that fit binary outcomes or continuous outcomes with predictions that are in (or scaled to) the 0-1 range. The method defaults to Multi-Accuracy Boosting as described in Kim et al. (2019). In order to obtain behaviour as described in Hebert-Johnson et al. (2018) set `multiplicative=FALSE` and `num_buckets` to 10.

For additional details, please refer to the relevant publications:

- Hebert-Johnson et al., 2018. Multicalibration: Calibration for the (Computationally-Identifiable) Masses. Proceedings of the 35th International Conference on Machine Learning, PMLR 80:1939-1948. https://proceedings.mlr.press/v80/hebert-johnson18a.html.
- Kim et al., 2019. Multiaccuracy: Black-Box Post-Processing for Fairness in Classification. Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society (AIES '19). Association for Computing Machinery, New York, NY, USA, 247–254. https://dl.acm.org/doi/10.1145/3306618.3314287

Public fields

`max_iter` `integer`
: The maximum number of iterations of the multi-calibration/multi-accuracy method.

`alpha` `numeric`
: Accuracy parameter that determines the stopping condition.

`eta` `numeric`
: Parameter for multiplicative weight update (step size).

`num_buckets` `integer`
: The number of buckets to split into in addition to using the whole sample.

`bucket_strategy` `character`
: Currently only supports "simple", even split along probabilities. Only relevant for `num_buckets` > 1.

`rebucket` `logical`
: Should buckets be re-calculated at each iteration?

`eval_fulldata` `logical`
: Should auditor be evaluated on the full data?

`partition` `logical`
: True/False flag for whether to split up predictions by their "partition" (e.g., predictions less than 0.5 and predictions greater than 0.5).

`multiplicative` `logical`
: Specifies the strategy for updating the weights (multiplicative weight vs additive).

`iter_sampling` `character`
: Specifies the strategy to sample the validation data for each iteration.

`auditor_fitter` `AuditorFitter`
: Specifies the type of model used to fit the residuals.

`predictor` `function`
: Initial predictor function.

`iter_models` `list`
: Cumulative list of fitted models.

`iter_partitions` `list`
: Cumulative list of data partitions for models.

`iter_corr` `list`
: Auditor correlation in each iteration.

`auditor_effects` `list`
: Auditor effect in each iteration.

`bucket_strategies` `character`
: Possible bucket_strategies in McBoostSurv.

Methods

Public methods:

- `MCBoost$new()`
- `MCBoost$multicalibrate()`

- MCBoost$predict_probs()
- MCBoost$auditor_effect()
- MCBoost$print()
- MCBoost$clone()

Method `new()`**:** Initialize a multi-calibration instance.

Usage:

```
MCBoost$new(
  max_iter = 5,
  alpha = 1e-04,
  eta = 1,
  num_buckets = 2,
  partition = ifelse(num_buckets > 1, TRUE, FALSE),
  bucket_strategy = "simple",
  rebucket = FALSE,
  eval_fulldata = FALSE,
  multiplicative = TRUE,
  auditor_fitter = NULL,
  subpops = NULL,
  default_model_class = ConstantPredictor,
  init_predictor = NULL,
  iter_sampling = "none"
)
```

Arguments:

`max_iter` `integer`
: The maximum number of iterations of the multi-calibration/multi-accuracy method. Default `5L`.

`alpha` `numeric`
: Accuracy parameter that determines the stopping condition. Default `1e-4`.

`eta` `numeric`
: Parameter for multiplicative weight update (step size). Default `1.0`.

`num_buckets` `integer`
: The number of buckets to split into in addition to using the whole sample. Default `2L`.

`partition` `logical`
: True/False flag for whether to split up predictions by their "partition" (e.g., predictions less than 0.5 and predictions greater than 0.5). Defaults to `TRUE` (multi-accuracy boosting).

`bucket_strategy` `character`
: Currently only supports "simple", even split along probabilities. Only taken into account for `num_buckets` > 1.

`rebucket` `logical`
: Should buckets be re-done at each iteration? Default `FALSE`.

`eval_fulldata` `logical`
: Should the auditor be evaluated on the full data or on the respective bucket for determining the stopping criterion? Default `FALSE`, auditor is only evaluated on the bucket. This setting keeps the implementation closer to the Algorithm proposed in the corresponding multi-accuracy paper (Kim et al., 2019) where auditor effects are computed across the full sample (i.e. eval_fulldata = TRUE).

`multiplicative` `logical`
Specifies the strategy for updating the weights (multiplicative weight vs additive). Defaults to `TRUE` (multi-accuracy boosting). Set to `FALSE` for multi-calibration.

`auditor_fitter` `AuditorFitter|character|mlr3::Learner`
Specifies the type of model used to fit the residuals. The default is `RidgeAuditorFitter`. Can be a `character`, the name of a `AuditorFitter`, a `mlr3::Learner` that is then auto-converted into a `LearnerAuditorFitter` or a custom `AuditorFitter`.

`subpops` `list`
Specifies a collection of characteristic attributes and the values they take to define subpopulations e.g. list(age = c('20-29','30-39','40+'), nJobs = c(0,1,2,'3+'), ,..).

`default_model_class` `Predictor`
The class of the model that should be used as the init predictor model if `init_predictor` is not specified. Defaults to `ConstantPredictor` which predicts a constant value.

`init_predictor` `function|mlr3::Learner`
The initial predictor function to use (i.e., if the user has a pretrained model). If a `mlr3 Learner` is passed, it will be autoconverted using `mlr3_init_predictor`. This requires the `mlr3::Learner` to be trained.

`iter_sampling` `character`
How to sample the validation data for each iteration? Can be `bootstrap`, `split` or `none`.
"split" splits the data into `max_iter` parts and validates on each sample in each iteration.
"bootstrap" uses a new bootstrap sample in each iteration.
"none" uses the same dataset in each iteration.

Method `multicalibrate()`**:** Run multi-calibration.

Usage:

```
MCBoost$multicalibrate(data, labels, predictor_args = NULL, audit = FALSE, ...)
```

Arguments:

`data` `data.table`
Features.

`labels` `numeric`
One-hot encoded labels (of same length as data).

`predictor_args` `any`
Arguments passed on to `init_predictor`. Defaults to `NULL`.

`audit` `logical`
Perform auditing? Initialized to `TRUE`.

`...` `any`
Params passed on to other methods.

Returns: `NULL`

Method `predict_probs()`**:** Predict a dataset with multi-calibrated predictions

Usage:

```
MCBoost$predict_probs(x, t = Inf, predictor_args = NULL, audit = FALSE, ...)
```

Arguments:

`x` `data.table`
Prediction data.

`t` integer
: Number of multi-calibration steps to predict. Default: `Inf` (all).

`predictor_args` any
: Arguments passed on to `init_predictor`. Defaults to `NULL`.

`audit` logical
: Should audit weights be stored? Default `FALSE`.

`...` any
: Params passed on to the residual prediction model's predict method.

Returns: numeric
Numeric vector of multi-calibrated predictions.

Method `auditor_effect()`**:** Compute the auditor effect for each instance which are the cumulative absolute predictions of the auditor. It indicates "how much" each observation was affected by multi-calibration on average across iterations.

Usage:

```
MCBoost$auditor_effect(
  x,
  aggregate = TRUE,
  t = Inf,
  predictor_args = NULL,
  ...
)
```

Arguments:

`x` data.table
: Prediction data.

`aggregate` logical
: Should the auditor effect be aggregated across iterations? Defaults to `TRUE`.

`t` integer
: Number of multi-calibration steps to predict. Defaults to `Inf` (all).

`predictor_args` any
: Arguments passed on to `init_predictor`. Defaults to `NULL`.

`...` any
: Params passed on to the residual prediction model's predict method.

Returns: numeric
Numeric vector of auditor effects for each row in `x`.

Method `print()`**:** Prints information about multi-calibration.

Usage:

```
MCBoost$print(...)
```

Arguments:

`...` any
: Not used.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
# See vignette for more examples.
# Instantiate the object
## Not run:
mc = MCBoost$new()
# Run multi-calibration on training dataset.
mc$multicalibrate(iris[1:100, 1:4], factor(sample(c("A", "B"), 100, TRUE)))
# Predict on test set
mc$predict_probs(iris[101:150, 1:4])
# Get auditor effect
mc$auditor_effect(iris[101:150, 1:4])

## End(Not run)
```

MCBoostSurv *Multi-Calibration Boosting*

Description

Implements Multi-Calibration Boosting by Hebert-Johnson et al. (2018) and Multi-Accuracy Boosting by Kim et al. (2019) for the multi-calibration of a machine learning model's prediction for survival models. Multi-Calibration works best in scenarios where the underlying data & labels are unbiased but a bias is introduced within the algorithm's fitting procedure. This is often the case, e.g. when an algorithm fits a majority population while ignoring or under-fitting minority populations. Expects initial models that predict probobilities (between 0 and 1) for different time points. The method defaults to Multi-Accuracy Boosting as described in Kim et al. (2019). In order to obtain behaviour as described in Hebert-Johnson et al. (2018) set `multiplicative=FALSE` and `num_buckets` to 10.

For additional details, please refer to the relevant publications:

- Hebert-Johnson et al., 2018. Multicalibration: Calibration for the (Computationally-Identifiable) Masses. Proceedings of the 35th International Conference on Machine Learning, PMLR 80:1939-1948. https://proceedings.mlr.press/v80/hebert-johnson18a.html.
- Kim et al., 2019. Multiaccuracy: Black-Box Post-Processing for Fairness in Classification. Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society (AIES '19). Association for Computing Machinery, New York, NY, USA, 247–254. https://dl.acm.org/doi/10.1145/3306618.3314287

Super class

`mcboost::MCBoost` -> `MCBoostSurv`

Public fields

`max_iter` `integer`
: The maximum number of iterations of the multi-calibration/multi-accuracy method.

`alpha` `numeric`
: Accuracy parameter that determines the stopping condition.

`eta` `numeric`
: Parameter for multiplicative weight update (step size).

`num_buckets` `integer`
: The number of buckets to split into in addition to using the whole sample.

`bucket_strategy` `character`
: Currently only supports "simple", even split along probabilities. Only relevant for `num_buckets` > 1.

`rebucket` `logical`
: Should buckets be re-calculated at each iteration?

`eval_fulldata` `logical`
: Should auditor be evaluated on the full data?

`partition` `logical`
: True/False flag for whether to split up predictions by their "partition" (e.g., predictions less than 0.5 and predictions greater than 0.5).

`multiplicative` `logical`
: Specifies the strategy for updating the weights (multiplicative weight vs additive).

`iter_sampling` `character`
: Specifies the strategy to sample the validation data for each iteration.

`auditor_fitter` `AuditorFitter`
: Specifies the type of model used to fit the residuals.

`predictor` `function`
: Initial predictor function.

`iter_models` `list`
: Cumulative list of fitted models.

`iter_partitions` `list`
: Cumulative list of data partitions for models.

`iter_corr` `list`
: Auditor correlation in each iteration.

`auditor_effects` `list`
: Auditor effect in each iteration.

`time_points` `integer`
: Times included in the prediction (columnames)

`time_buckets` `integer`
: The number of buckets to split the time points (columns) of the prediction.

`bucket_strategies` `character`
: Possible bucket_strategies in McBoostSurv. Only relevant for `time_buckets` > 1. `even_splits`: split buckets evenly `quantiles`: split buckets by quantiles

`bucket_aggregation` `function`
: If not NULL, predictions are not selected by time/probability, but by time/individual. Individuals are selected by aggregated value per individual (e.g. mean). Only relevant for `time_buckets` > 1.

`max_time_quantile` `double`
: Time quantile which should be evaluated and multicalibrated. Similar to a 75%-Integrated Brier Score.

`time_points_eval` `integer`
: Vector of time_points that should be evaluated.

`loss` `character`
: Loss function which is optimized during boosting. `censored_brier`: censored version of the integrated brier score `brier`: uncensored version of the integrated brier score `censored_brier_proper`: proper version of the censored version of the integrated brier score For more details, we are referring to https://mlr3proba.mlr-org.com/reference/mlr_measures_surv.graf.html.

Methods

Public methods:

- `MCBoostSurv$new()`
- `MCBoostSurv$clone()`

Method `new()`**:** Initialize a multi-calibration instance.

Usage:

```
MCBoostSurv$new(
  max_iter = 25,
  alpha = 1e-04,
  eta = 0.1,
  num_buckets = 1,
  partition = ifelse(num_buckets > 1, TRUE, FALSE),
  time_buckets = 2L,
  max_time_quantile = 1,
  bucket_strategy = "even_splits",
  bucket_aggregation = NULL,
  rebucket = FALSE,
  eval_fulldata = FALSE,
  multiplicative = TRUE,
  auditor_fitter = "RidgeAuditorFitter",
  subpops = NULL,
  default_model_class = LearnerSurvKaplan,
  init_predictor = NULL,
  loss = "censored_brier",
  iter_sampling = "none"
)
```

Arguments:

`max_iter` `integer`
: The maximum number of iterations of the multi-calibration/multi-accuracy method. Default `5L`.

alpha numeric
: Accuracy parameter that determines the stopping condition. Default 1e-4.

eta numeric
: Parameter for multiplicative weight update (step size). Default 1.0.

num_buckets integer
: The number of buckets to split into in addition to using the whole sample. Default 2L.

partition logical
: True/False flag for whether to split up predictions by their "partition" (e.g., predictions less than 0.5 and predictions greater than 0.5). Defaults to TRUE (multi-accuracy boosting).

time_buckets integer
: The number of buckets to split the time points (columns) of the prediction.

max_time_quantile double
: Time quantile which should be evaluated and multicalibrated. Can be used to perform multi-calibration only up to the max_time_quantile percent of timepoints. Initialized to 1.

bucket_strategy character
: Bucketstragy for bucketing. even_splits: split buckets evenly quantiles: split buckets by quantiles

bucket_aggregation function
: If not NULL, predictions are not selected by time/probability, but by time/individual. Individuals are selected by aggregated value per individual (e.g. mean). Only relevant for time_buckets > 1.

rebucket logical
: Should buckets be re-done at each iteration? Default FALSE.

eval_fulldata logical
: Should the auditor be evaluated on the full data or on the respective bucket for determining the stopping criterion? Default FALSE, auditor is only evaluated on the bucket. This setting keeps the implementation closer to the Algorithm proposed in the corresponding multi-accuracy paper (Kim et al., 2019) where auditor effects are computed across the full sample (i.e. eval_fulldata = TRUE).

multiplicative logical
: Specifies the strategy for updating the weights (multiplicative weight vs additive). Defaults to TRUE (multi-accuracy boosting). Set to FALSE for multi-calibration.

auditor_fitter AuditorFitter|character|mlr3::Learner
: Specifies the type of model used to fit the residuals. The default is RidgeAuditorFitter. Can be a character, the name of a AuditorFitter, a mlr3::Learner that is then auto-converted into a LearnerAuditorFitter or a custom AuditorFitter.

subpops list
: Specifies a collection of characteristic attributes and the values they take to define subpopulations e.g. list(age = c('20-29','30-39','40+'), nJobs = c(0,1,2,'3+'), ,..).

default_model_class Predictor
: The class of the model that should be used as the init predictor model if init_predictor is not specified. Defaults to ConstantPredictor which predicts a constant value.

init_predictor function|mlr3::Learner
: The initial predictor function to use (i.e., if the user has a pretrained model). If a mlr3 Learner is passed, it will be autoconverted using mlr3_init_predictor. This requires the mlr3::Learner to be trained.

`loss` `character`
: #' Loss function which is optimized during boosting. `censored_brier`: censored version of the integrated brier score `brier`: uncensored version of the integrated brier score `censored_brier_proper`: proper version of the censored version of the integrated brier score For more details, we are referring to https://mlr3proba.mlr-org.com/reference/mlr_measures_surv.graf.html.

`iter_sampling` `character`
: How to sample the validation data for each iteration? Can be `bootstrap`, `split` or `none`.
"split" splits the data into `max_iter` parts and validates on each sample in each iteration.
"bootstrap" uses a new bootstrap sample in each iteration.
"none" uses the same dataset in each iteration.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`mlr3_init_predictor` *Create an initial predictor function from a trained mlr3 learner*

Description

Create an initial predictor function from a trained mlr3 learner

Usage

```
mlr3_init_predictor(learner)
```

Arguments

`learner`	`mlr3::Learner`	A trained learner used for initialization.

Value

`function`

Examples

```
## Not run:
library("mlr3")
l = lrn("classif.featureless")$train(tsk("sonar"))
mlr3_init_predictor(l)

## End(Not run)
```

mlr_pipeops_mcboost *Multi-Calibrate a Learner's Prediction*

Description

Post-process a learner prediction using multi-calibration. For more details, please refer to https://arxiv.org/pdf/1805.12317.pdf (Kim et al. 2018) or the help for MCBoost. If no init_predictor is provided, the preceding learner's predictions corresponding to the prediction slot are used as an initial predictor for MCBoost.

Format

R6Class inheriting from mlr3pipelines::PipeOp.

R6Class inheriting from mlr3pipelines::PipeOp.

Construction

```
PipeOpLearnerPred$new(learner, id = NULL, param_vals = list())

* `learner` :: [`Learner`][mlr3::Learner] \cr
  [`Learner`][mlr3::Learner] to  prediction, or a string identifying a
 [`Learner`][mlr3::Learner] in the [`mlr3::mlr_learners`] [`Dictionary`][mlr3misc::Dictionary].
* `id` :: `character(1)`
 Identifier of the resulting object, internally defaulting to the `id` of the [`Learner`][mlr3::Learner
* `param_vals` :: named `list`\cr
 List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set d


[mlr3::Learner]: R:mlr3::Learner
[mlr3::Learner]: R:mlr3::Learner
[mlr3::Learner]: R:mlr3::Learner
[`mlr3::mlr_learners`]: R:%60mlr3::mlr_learners%60
[mlr3misc::Dictionary]: R:mlr3misc::Dictionary
[mlr3::Learner]: R:mlr3::Learner


PipeOpMCBoost$new(id = "mcboost", param_vals = list())
```

- id :: character(1) Identifier of the resulting object, default "threshold".
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. See MCBoost for a comprehensive description of all hyperparameters.

Input and Output Channels

PipeOpLearnerPred has one input channel named "input", taking a Task specific to the Learner type given to learner during construction; both during training and prediction.

PipeOpLearnerPred has one output channel named "output", producing a Task specific to the Learner type given to learner during construction; both during training and prediction.

During training, the input and output are "data" and "prediction", two TaskClassif. A PredictionClassif is required as input and returned as output during prediction.

State

The $state is a MCBoost Object as obtained from MCBoost$new().

Parameters

The $state is set to the $state slot of the Learner object, together with the $state elements inherited from mlr3pipelines::PipeOpTaskPreproc. It is a named list with the inherited members, as well as:

- model :: any
  Model created by the Learner's $.train() function.
- train_log :: data.table with columns class (character), msg (character)
  Errors logged during training.
- train_time :: numeric(1)
  Training time, in seconds.
- predict_log :: NULL | data.table with columns class (character), msg (character)
  Errors logged during prediction.
- predict_time :: NULL | numeric(1) Prediction time, in seconds.
- max_iter :: integer
  A integer specifying the number of multi-calibration rounds. Defaults to 5.

Fields

Fields inherited from PipeOp, as well as:

- learner :: Learner
  Learner that is being wrapped. Read-only.
- learner_model :: Learner
  Learner that is being wrapped. This learner contains the model if the PipeOp is trained. Read-only.

Only fields inherited from mlr3pipelines::PipeOp.

Methods

Methods inherited from mlr3pipelines::PipeOpTaskPreproc/mlr3pipelines::PipeOp.

Only methods inherited from mlr3pipelines::PipeOp.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `PipeOpLearnerPred`

Active bindings

`learner` The wrapped learner.

`learner_model` The wrapped learner's model(s).

Methods

Public methods:

- `PipeOpLearnerPred$new()`
- `PipeOpLearnerPred$clone()`

Method `new()`**:** Initialize a Learner Predictor PipeOp. Can be used to wrap trained or untrainted mlr3 learners.

Usage:

```
PipeOpLearnerPred$new(learner, id = NULL, param_vals = list())
```

Arguments:

`learner` `Learner`
The learner that should be wrapped.

`id` `character`
The `PipeOp`'s id. Defaults to "mcboost".

`param_vals` `list`
List of hyperparameters for the `PipeOp`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpMCBoost`

Active bindings

`predict_type` Predict type of the PipeOp.

Methods

Public methods:

- `PipeOpMCBoost$new()`
- `PipeOpMCBoost$clone()`

Method `new()`**:** Initialize a Multi-Calibration PipeOp.

Usage:

```
PipeOpMCBoost$new(id = "mcboost", param_vals = list())
```

Arguments:

`id` `character`
: The `PipeOp`'s id. Defaults to "mcboost".

`param_vals` `list`
: List of hyperparameters for the `PipeOp`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

https://mlr3book.mlr-org.com/list-pipeops.html

https://mlr3book.mlr-org.com/list-pipeops.html

Other PipeOps: `mlr_pipeops_mcboostsurv`

Other PipeOps: `mlr_pipeops_mcboostsurv`

Examples

```
## Not run:
gr = gunion(list(
  "data" = po("nop"),
  "prediction" = po("learner_cv", lrn("classif.rpart"))
)) %>>%
  PipeOpMCBoost$new()
tsk = tsk("sonar")
tid = sample(1:208, 108)
gr$train(tsk$clone()$filter(tid))
gr$predict(tsk$clone()$filter(setdiff(1:208, tid)))

## End(Not run)
```

---

mlr_pipeops_mcboostsurv

Multi-Calibrate a Learner's Prediction (Survival Model)

Description

Post-process a survival learner prediction using multi-calibration. For more details, please refer to https://arxiv.org/pdf/1805.12317.pdf (Kim et al. 2018) or the help for `MCBoostSurv`. If no `init_predictor` is provided, the preceding learner's predictions corresponding to the `prediction` slot are used as an initial predictor for `MCBoostSurv`.

Format

R6Class inheriting from mlr3pipelines::PipeOp.

Construction

```
PipeOpMCBoostSurv$new(id = "mcboostsurv", param_vals = list())
```

- id :: character(1) Identifier of the resulting object, default "threshold".
- param_vals :: named list
  List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. See MCBoostSurv for a comprehensive description of all hyperparameters.

Input and Output Channels

During training, the input and output are "data" and "prediction", two TaskSurv. A PredictionSurv is required as input and returned as output during prediction.

State

The $state is a MCBoostSurv Object as obtained from MCBoostSurv$new().

Parameters

- max_iter :: integer
  A integer specifying the number of multi-calibration rounds. Defaults to 5.

Fields

Only fields inherited from mlr3pipelines::PipeOp.

Methods

Only methods inherited from mlr3pipelines::PipeOp.

Super class

mlr3pipelines::PipeOp -> PipeOpMCBoostSurv

Active bindings

predict_type Predict type of the PipeOp.

Methods

Public methods:

- PipeOpMCBoostSurv$new()
- PipeOpMCBoostSurv$clone()

Method new(): Initialize a Multi-Calibration PipeOp (for Survival).

Usage:

```
PipeOpMCBoostSurv$new(id = "mcboostsurv", param_vals = list())
```

Arguments:

`id` `character`
: The `PipeOp`'s id. Defaults to "mcboostsurv".

`param_vals` `list`
: List of hyperparameters for the `PipeOp`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

https://mlr3book.mlr-org.com/list-pipeops.html

Other PipeOps: `mlr_pipeops_mcboost`

Examples

```
library(mlr3)
library(mlr3pipelines)
# Attention: gunion inputs have to be in the correct order for now.
## Not run:
gr = gunion(list(
  "data" = po("nop"),
  "prediction" = po("learner_pred", lrn("surv.ranger"))
)) %>>%
  PipeOpMCBoostSurv$new()
tsk = tsk("rats")
tid = sample(1:300, 100)
gr$train(tsk$clone()$filter(tid))
gr$predict(tsk$clone()$filter(setdiff(1:300, tid)))

## End(Not run)
```

`one_hot` *One-hot encode a factor variable*

Description

One-hot encode a factor variable

Usage

```
one_hot(labels)
```

Arguments

labels	factor Factor to encode.

Value

integer
Integer vector of encoded labels.

Examples

```
## Not run:
one_hot(factor(c("a", "b", "a")))

## End(Not run)
```

---

ppl_mcboost *Multi-calibration pipeline*

Description

Wraps MCBoost in a Pipeline to be used with `mlr3pipelines`. For now this assumes training on the same dataset that is later used for multi-calibration.

Usage

```
ppl_mcboost(learner = lrn("classif.featureless"), param_vals = list())
```

Arguments

learner	(mlr3)`mlr3::Learner` Initial learner. Internally wrapped into a `PipeOpLearnerCV` with `resampling.method = "insample"` as a default. All parameters can be adjusted through the resulting Graph's `param_set`. Defaults to `lrn("classif.featureless")`. Note: An initial predictor can also be supplied via the `init_predictor` parameter.
param_vals	`list` List of parameter values passed on to `MCBoost$new`.

Value

(mlr3pipelines) `Graph`

Examples

```
## Not run:
library("mlr3pipelines")
gr = ppl_mcboost()

## End(Not run)
```

ppl_mcboostsurv *Multi-calibration pipeline (for survival models)*

Description

Wraps MCBoostSurv in a Pipeline to be used with `mlr3pipelines`. For now this assumes training on the same dataset that is later used for multi-calibration.

Usage

```
ppl_mcboostsurv(learner = lrn("surv.kaplan"), param_vals = list())
```

Arguments

`learner`	(mlr3)`mlr3::Learner` Initial learner. Defaults to `lrn("surv.kaplan")`. Note: An initial predictor can also be supplied via the `init_predictor` parameter. The learner is internally wrapped into a `PipeOpLearnerCV` with `resampling.method = "insample"` as a default. All parameters can be adjusted through the resulting Graph's `param_set`.
`param_vals`	`list` List of parameter values passed on to `MCBoostSurv$new`

Value

(mlr3pipelines) `Graph`

Examples

```
library("mlr3pipelines")
gr = ppl_mcboostsurv()
```

SubgroupAuditorFitter *Static AuditorFitter based on Subgroups*

Description

Used to assess multi-calibration based on a list of binary `subgroup_masks` passed during initialization.

Value

`AuditorFitter`

`list` with items

- `corr`: pseudo-correlation between residuals and learner prediction.
- `l`: the trained learner.

Super class

`mcboost::AuditorFitter` -> `SubgroupAuditorFitter`

Public fields

`subgroup_masks` `list`
: List of subgroup masks. Initialize a SubgroupAuditorFitter

Methods

Public methods:

- `SubgroupAuditorFitter$new()`
- `SubgroupAuditorFitter$fit()`
- `SubgroupAuditorFitter$clone()`

Method `new()`**:** Initializes a `SubgroupAuditorFitter` that assesses multi-calibration within each group defined by the 'subpops'.

Usage:

```
SubgroupAuditorFitter$new(subgroup_masks)
```

Arguments:

`subgroup_masks` `list`
: List of subgroup masks. Subgroup masks are list(s) of integer masks, each with the same length as data to be fitted on. They allow defining subgroups of the data.

Method `fit()`**:** Fit the learner and compute correlation

Usage:

```
SubgroupAuditorFitter$fit(data, resid, mask)
```

Arguments:

`data` `data.table`
: Features.

`resid` `numeric`
: Residuals (of same length as data).

`mask` `integer`
: Mask applied to the data. Only used for `SubgroupAuditorFitter`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other AuditorFitter: `CVLearnerAuditorFitter`, `LearnerAuditorFitter`, `SubpopAuditorFitter`

Examples

```
## Not run:
library("data.table")
data = data.table(
  "AGE_0_10" =  c(1, 1, 0, 0, 0),
  "AGE_11_20" = c(0, 0, 1, 0, 0),
  "AGE_21_31" = c(0, 0, 0, 1, 1),
  "X1" = runif(5),
  "X2" = runif(5)
)
label = c(1,0,0,1,1)
masks = list(
  "M1" = c(1L, 0L, 1L, 1L, 0L),
  "M2" = c(1L, 0L, 0L, 0L, 1L)
)
sg = SubgroupAuditorFitter$new(masks)

## End(Not run)
```

SubpopAuditorFitter *Static AuditorFitter based on Subpopulations*

Description

Used to assess multi-calibration based on a list of binary valued columns: subpops passed during initialization.

Value

AuditorFitter

list with items

- corr: pseudo-correlation between residuals and learner prediction.
- l: the trained learner.

Super class

mcboost::AuditorFitter -> SubpopAuditorFitter

Public fields

subpops list
: List of subpopulation indicators. Initialize a SubpopAuditorFitter

Methods

Public methods:

- SubpopAuditorFitter$new()
- SubpopAuditorFitter$fit()
- SubpopAuditorFitter$clone()

Method `new()`**:** Initializes a SubpopAuditorFitter that assesses multi-calibration within each group defined by the subpops'. Names in subpops‘ must correspond to columns in the data.

Usage:

```
SubpopAuditorFitter$new(subpops)
```

Arguments:

`subpops` list
: Specifies a collection of characteristic attributes and the values they take to define subpopulations e.g. list(age = c('20-29','30-39','40+'), nJobs = c(0,1,2,'3+'), ,..).

Method `fit()`**:** Fit the learner and compute correlation

Usage:

```
SubpopAuditorFitter$fit(data, resid, mask)
```

Arguments:

`data` data.table
: Features.

`resid` numeric
: Residuals (of same length as data).

`mask` integer
: Mask applied to the data. Only used for `SubgroupAuditorFitter`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other AuditorFitter: CVLearnerAuditorFitter, LearnerAuditorFitter, SubgroupAuditorFitter

Examples

```
## Not run:
library("data.table")
data = data.table(
  "AGE_NA" = c(0, 0, 0, 0, 0),
  "AGE_0_10" =  c(1, 1, 0, 0, 0),
  "AGE_11_20" = c(0, 0, 1, 0, 0),
  "AGE_21_31" = c(0, 0, 0, 1, 1),
```

```
    "X1" = runif(5),
    "X2" = runif(5)
  )
  label = c(1,0,0,1,1)
 pops = list("AGE_NA", "AGE_0_10", "AGE_11_20", "AGE_21_31", function(x) {x[["X1" > 0.5]]})
  sf = SubpopAuditorFitter$new(subpops = pops)
  sf$fit(data, label - 0.5)

## End(Not run)
```

Index