Package: mlr3proba (via r-universe)

November 21, 2024

Title Probabilistic Supervised Learning for 'mlr3'

Version 0.7.0

Description Provides extensions for probabilistic supervised learning for 'mlr3'. This includes extending the regression task to probabilistic and interval regression, adding a survival task, and other specialized models, predictions, and measures.

License LGPL-3

URL `https://mlr3proba.mlr-org.com`, `https://github.com/mlr-org/mlr3proba`

BugReports `https://github.com/mlr-org/mlr3proba/issues`

Depends mlr3 (>= 0.14.1), R (>= 3.5.0)

Imports checkmate, data.table, distr6 (>= 1.8.4), ggplot2, mlr3misc (>= 0.7.0), mlr3pipelines (>= 0.7.0), mlr3viz, paradox (>= 1.0.0), R6, Rcpp (>= 1.0.4), survival

Suggests bujar, GGally, knitr, lgr, lifecycle, param6 (>= 0.2.4), pracma, rpart, set6 (>= 0.2.6), simsurv, survAUC, testthat (>= 3.0.0), vdiffr, abind, Ecdat, coxed, mlr3learners, pammtools

LinkingTo Rcpp

Remotes xoopR/distr6, xoopR/param6, xoopR/set6

Config/testthat/edition 3

ByteCompile true

Encoding UTF-8

LazyData true

NeedsCompilation no

Roxygen list(markdown = TRUE, r6 = TRUE)

RoxygenNote 7.3.2

Collate 'LearnerDens.R' 'aaa.R' 'LearnerDensHistogram.R' 'LearnerDensKDE.R' 'LearnerSurv.R' 'LearnerSurvCoxPH.R' 'LearnerSurvKaplan.R' 'LearnerSurvRpart.R' 'MeasureDens.R'

'MeasureDensLogloss.R' 'MeasureRegrLogloss.R' 'MeasureSurv.R'
'MeasureSurvAUC.R' 'MeasureSurvCalibrationAlpha.R'
'MeasureSurvCalibrationBeta.R' 'MeasureSurvChamblessAUC.R'
'MeasureSurvCindex.R' 'MeasureSurvDCalibration.R'
'MeasureSurvGraf.R' 'MeasureSurvHungAUC.R'
'MeasureSurvIntLogloss.R' 'MeasureSurvLogloss.R'
'MeasureSurvMAE.R' 'MeasureSurvMSE.R' 'MeasureSurvNagelkR2.R'
'MeasureSurvOQuigleyR2.R' 'MeasureSurvRCLL.R'
'MeasureSurvRMSE.R' 'MeasureSurvSchmid.R'
'MeasureSurvSongAUC.R' 'MeasureSurvSongTNR.R'
'MeasureSurvSongTPR.R' 'MeasureSurvUnoAUC.R'
'MeasureSurvUnoTNR.R' 'MeasureSurvUnoTPR.R' 'MeasureSurvXuR2.R'
'PipeOpBreslow.R' 'PipeOpCrankCompositor.R'
'PipeOpDistrCompositor.R' 'PipeOpPredClassifSurvDiscTime.R'
'PipeOpPredClassifSurvIPCW.R' 'PipeOpTransformer.R'
'PipeOpPredTransformer.R' 'PipeOpPredRegrSurv.R'
'PipeOpPredSurvRegr.R' 'PipeOpProbregrCompositor.R'
'PipeOpResponseCompositor.R' 'PipeOpSurvAvg.R'
'PipeOpTaskRegrSurv.R' 'PipeOpTaskSurvClassifDiscTime.R'
'PipeOpTaskSurvClassifIPCW.R' 'PipeOpTaskSurvRegr.R'
'PipeOpTaskTransformer.R' 'PredictionDataDens.R'
'PredictionDataSurv.R' 'PredictionDens.R' 'PredictionSurv.R'
'RcppExports.R' 'TaskDens.R' 'TaskDens_zzz.R'
'TaskGeneratorCoxed.R' 'TaskGeneratorSimdens.R'
'TaskGeneratorSimsurv.R' 'TaskSurv.R' 'TaskSurv_zzz.R'
'as_prediction_dens.R' 'as_prediction_surv.R' 'as_task_dens.R'
'as_task_surv.R' 'assertions.R' 'autoplot.R' 'bibentries.R'
'breslow.R' 'cindex.R' 'data.R' 'helpers.R' 'histogram.R'
'integrated_scores.R' 'mlr3proba-package.R' 'pecs.R'
'pipelines.R' 'plot.R' 'plot_probregr.R' 'scoring_rule_erv.R'
'surv_measures.R' 'surv_return.R' 'zzz.R'

Repository https://mlr-org.r-universe.dev

RemoteUrl https://github.com/mlr-org/mlr3proba

RemoteRef v0.7.0

RemoteSha 44fd50c3313b0a14b023a4d4a707c16f52cc3bda

Contents

mlr3proba-package *mlr3proba: Probabilistic Supervised Learning for 'mlr3'*

Description

Provides extensions for probabilistic supervised learning for 'mlr3'. This includes extending the regression task to probabilistic and interval regression, adding a survival task, and other specialized models, predictions, and measures.

Author(s)

Maintainer: John Zobolas <bblodfon@gmail.com> (ORCID)

Authors:

- Raphael Sonabend <raphaelsonabend@gmail.com> (ORCID)
- Franz Kiraly <f.kiraly@ucl.ac.uk>
- Michel Lang <michellang@gmail.com> (ORCID)
- Philip Studener <philip.studener@gmx.de>

Other contributors:

- Nurul Ain Toha <nurul.toha.15@ucl.ac.uk> [contributor]
- Andreas Bender <bender.at.R@gmail.com> (ORCID) [contributor]
- Lukas Burk <github@quantenbrot.de> (ORCID) [contributor]
- Maximilian Muecke <muecke.maximilian@gmail.com> (ORCID) [contributor]

See Also

Useful links:

- https://mlr3proba.mlr-org.com
- https://github.com/mlr-org/mlr3proba
- Report bugs at https://github.com/mlr-org/mlr3proba/issues

`.surv_return` *Get Survival Predict Types*

Description

Internal helper function to easily return the correct survival predict types.

Usage

```
.surv_return(
  times = NULL,
  surv = NULL,
  crank = NULL,
  lp = NULL,
  response = NULL,
  which.curve = NULL
)
```

Arguments

`times` (`numeric()`)
Vector of survival times.

`surv` (`matrix()|array()`)
Matrix or array of predicted survival probabilities, rows (1st dimension) are observations, columns (2nd dimension) are times and in the case of an array there should be one more dimension. Number of columns should be equal to length of `times`. In case a `numeric()` vector is provided, it is converted to a single row (one observation) matrix.

`crank` (`numeric()`)
Relative risk/continuous ranking. Higher value is associated with higher risk. If `NULL` then either set as `-response` if available or `lp` if available (this assumes that the `lp` prediction comes from a PH type model - in case of an AFT model the user should provide `-lp`). In case neither `response` or `lp` are provided, then `crank` is calculated as the sum of the cumulative hazard function (**expected mortality**) derived from the predicted survival function (`surv`), see get_mortality. In case `surv` is a 3d array, we use the `which.curve` parameter to decide which survival matrix (index in the 3rd dimension) will be chosen for the calculation of `crank`.

`lp` (`numeric()`)
Predicted linear predictor, used to impute `crank` if `NULL`.

`response` (`numeric()`)
Predicted survival time, passed through function without modification.

`which.curve` Which curve (3rd dimension) should the `crank` be calculated for, in case `surv` is an `array`? If between (0,1) it is taken as the quantile of the curves otherwise if greater than 1 it is taken as the curve index. It can also be 'mean' and the survival probabilities are averaged across the 3rd dimension. Default value (`NULL`) is the **0.5 quantile** which is the median across the 3rd dimension of the survival array.

References

Sonabend, Raphael, Bender, Andreas, Vollmer, Sebastian (2022). "Avoiding C-hacking when evaluating survival distribution predictions with discrimination measures." *Bioinformatics*. ISSN 1367-4803, doi:10.1093/BIOINFORMATICS/BTAC451, `https://academic.oup.com/bioinformatics/advance-article/doi/10.1093/bioinformatics/btac451/6640155`.

Examples

```
n = 10 # number of observations
k = 50 # time points

# Create the matrix with random values between 0 and 1
mat = matrix(runif(n * k, min = 0, max = 1), nrow = n, ncol = k)

# transform it to a survival matrix
surv_mat = t(apply(mat, 1L, function(row) sort(row, decreasing = TRUE)))

# crank is expected mortality, distr is the survival matrix
.surv_return(times = 1:k, surv = surv_mat)

# if crank is set, it's not overwritten
.surv_return(times = 1:k, surv = surv_mat, crank = rnorm(n))

# lp = crank
.surv_return(lp = rnorm(n))

# if response is set and no crank, crank = -response
.surv_return(response = sample(1:100, n))

# if both are set, they are not overwritten
.surv_return(crank = rnorm(n), response = sample(1:100, n))
```

---

actg *ACTG 320 Clinical Trial Dataset*

Description

actg dataset from Hosmer et al. (2008)

Usage

```
actg
```

Format

id Identification Code

time Time to AIDS diagnosis or death (days).

censor Event indicator. 1 = AIDS defining diagnosis, 0 = Otherwise.

time_d Time to death (days)

censor_d Event indicator for death (only). 1 = Death, 0 = Otherwise.

tx Treatment indicator. 1 = Treatment includes IDV, 0 = Control group.

txgrp Treatment group indicator. 1 = ZDV + 3TC. 2 = ZDV + 3TC + IDV. 3 = d4T + 3TC. 4 = d4T + 3TC + IDV.

strat2 CD4 stratum at screening. 0 = CD4 <= 50. 1 = CD4 > 50.

sexF 0 = Male. 1 = Female.

raceth Race/Ethnicity. 1 = White Non-Hispanic. 2 = Black Non-Hispanic. 3 = Hispanic. 4 = Asian, Pacific Islander. 5 = American Indian, Alaskan Native. 6 = Other/unknown.

ivdrug IV drug use history. 1 = Never. 2 = Currently. 3 = Previously.

hemophil Hemophiliac. 1 = Yes. 0 = No.

karnof Karnofsky Performance Scale. 100 = Normal; no complaint no evidence of disease. 90 = Normal activity possible; minor signs/symptoms of disease. 80 = Normal activity with effort; some signs/symptoms of disease. 70 = Cares for self; normal activity/active work not possible.

cd4 Baseline CD4 count (Cells/Milliliter).

priorzdv Months of prior ZDV use (months).

age Age at Enrollment (years).

Source

https://onlinelibrary.wiley.com/doi/book/10.1002/9780470258019

References

Hosmer, D.W. and Lemeshow, S. and May, S. (2008) Applied Survival Analysis: Regression Modeling of Time to Event Data: Second Edition, John Wiley and Sons Inc., New York, NY

---

assert_surv *Assert survival object*

Description

Asserts x is a survival::Surv object with added checks

Usage

```
assert_surv(
  x,
  len = NULL,
  any.missing = TRUE,
  null.ok = FALSE,
  .var.name = vname(x)
)
```

Arguments

`x`	Object to check
`len`	If non-NULL checks object is length `len`
`any.missing`	If `FALSE` then errors if there are any NAs in `x`
`null.ok`	If `FALSE` then errors if `x` is NULL, otherwise passes
`.var.name`	Optional variable name to return if assertion fails

`assert_surv_matrix` *Assert survival matrix*

Description

Asserts if the given input matrix is a (discrete) survival probabilities matrix using Rcpp code. The following checks are performed:

1. All values are probabilities, i.e. $S(t) \in [0, 1]$
2. Column names correspond to time-points and should therefore be coercable to `numeric` and increasing
3. Per row/observation, the survival probabilities decrease non-strictly, i.e. $S(t) \geq S(t + 1)$

Usage

```
assert_surv_matrix(x)
```

Arguments

`x`	`(matrix())` A matrix of (predicted) survival probabilities. Rows are observations, columns are (increasing) time points.

Value

if the assertion fails an error occurs, otherwise `NULL` is returned invisibly.

Examples

```
x = matrix(data = c(1,0.6,0.4,0.8,0.8,0.7), nrow = 2, ncol = 3, byrow = TRUE)
colnames(x) = c(12, 34, 42)
x

assert_surv_matrix(x)
```

as_prediction_dens *Convert to a Density Prediction*

Description

Convert object to a PredictionDens.

Usage

```
as_prediction_dens(x, ...)

## S3 method for class 'PredictionDens'
as_prediction_dens(x, ...)

## S3 method for class 'data.frame'
as_prediction_dens(x, ...)
```

Arguments

`x`	(any) Object to convert.
`...`	(any) Additional arguments.

Value

PredictionDens.

Examples

```
library(mlr3)
task = tsk("precip")
learner = lrn("dens.hist")
learner$train(task)
p = learner$predict(task)

# convert to a data.table
tab = as.data.table(p)

# convert back to a Prediction
as_prediction_dens(tab)
```

as_prediction_surv *Convert to a Survival Prediction*

Description

Convert object to a PredictionSurv.

Usage

```
as_prediction_surv(x, ...)

## S3 method for class 'PredictionSurv'
as_prediction_surv(x, ...)

## S3 method for class 'data.frame'
as_prediction_surv(x, ...)
```

Arguments

`x`	(any) Object to convert.
`...`	(any) Additional arguments.

Value

PredictionSurv.

Examples

```
library(mlr3)
task = tsk("rats")
learner = lrn("surv.coxph")
learner$train(task)
p = learner$predict(task)

# convert to a data.table
tab = as.data.table(p)

# convert back to a Prediction
as_prediction_surv(tab)
```

as_task_dens *Convert to a Density Task*

Description

Convert object to a density task (TaskDens).

Usage

```
as_task_dens(x, ...)

## S3 method for class 'TaskDens'
as_task_dens(x, clone = FALSE, ...)

## S3 method for class 'data.frame'
as_task_dens(x, id = deparse(substitute(x)), ...)

## S3 method for class 'DataBackend'
as_task_dens(x, id = deparse(substitute(x)), ...)
```

Arguments

`x`	(`any`) Object to convert, e.g. a `data.frame()`.
`...`	(`any`) Additional arguments.
`clone`	(`logical(1)`) If `TRUE`, ensures that the returned object is not the same as the input `x`.
`id`	(`character(1)`) Id for the new task. Defaults to the (deparsed and substituted) name of `x`.

as_task_surv *Convert to a Survival Task*

Description

Convert object to a survival task (TaskSurv).

Usage

```
as_task_surv(x, ...)

## S3 method for class 'TaskSurv'
as_task_surv(x, clone = FALSE, ...)

## S3 method for class 'data.frame'
as_task_surv(
  x,
  time = "time",
  event = "event",
  time2,
  type = "right",
  id = deparse(substitute(x)),
  ...
)

## S3 method for class 'DataBackend'
as_task_surv(
  x,
  time = "time",
  event = "event",
  time2,
  type = "right",
  id = deparse(substitute(x)),
  ...
)

## S3 method for class 'formula'
as_task_surv(x, data, id = deparse(substitute(data)), ...)
```

Arguments

`x`
(any)
Object to convert, e.g. a `data.frame()`.

`...`
(any)
Additional arguments.

`clone`
(`logical(1)`)
If `TRUE`, ensures that the returned object is not the same as the input `x`.

`time`
(`character(1)`)
Name of the column for event time if data is right censored, otherwise starting time if interval censored.

`event`
(`character(1)`)
Name of the column giving the event indicator. If data is right censored then "0"/`FALSE` means alive (no event), "1"/`TRUE` means dead (event). If `type` is `"interval"` then "0" means right censored, "1" means dead (event), "2" means

left censored, and "3" means interval censored. If type is "interval2" then event is ignored.

time2 (character(1))
Name of the column for ending time of the interval for interval censored or counting process data, otherwise ignored.

type (character(1))
Name of the column giving the type of censoring. Default is 'right' censoring.

id (character(1))
Id for the new task. Defaults to the (deparsed and substituted) name of x.

data (data.frame())
Data frame containing all columns referenced in formula x.

autoplot.PredictionSurv

Plot for PredictionSurv

Description

Generates plots for PredictionSurv, depending on argument type:

- "calib" (default): Calibration plot comparing the average predicted survival distribution to a Kaplan-Meier prediction, this is *not* a comparison of a stratified crank or lp prediction. object must have distr prediction. geom_line() is used for comparison split between the prediction (Pred) and Kaplan-Meier estimate (KM). In addition labels are added for the x (T) and y (S(T)) axes.
- "dcalib": Distribution calibration plot. A model is D-calibrated if X% of deaths occur before the X/100 quantile of the predicted distribution, e.g. if 50% of observations die before their predicted median survival time. A model is D-calibrated if the resulting plot lies on x = y.
- "preds": Matplots the survival curves for all predictions

Usage

```
## S3 method for class 'PredictionSurv'
autoplot(
  object,
  type = "calib",
  task = NULL,
  row_ids = NULL,
  times = NULL,
  xyline = TRUE,
  cuts = 11L,
  theme = theme_minimal(),
  extend_quantile = FALSE,
  ...
)
```

Arguments

`object` (PredictionSurv).

`type` (`character(1)`)
Name of the column giving the type of censoring. Default is 'right' censoring.

`task` (TaskSurv)
If `type = "calib"` then `task` is passed to `$predict` in the Kaplan-Meier learner.

`row_ids` (`integer()`)
If `type = "calib"` then `row_ids` is passed to `$predict` in the Kaplan-Meier learner.

`times` (`numeric()`)
If `type = "calib"` then `times` is the values on the x-axis to plot over, if `NULL` uses all times from `task`.

`xyline` (`logical(1)`)
If `TRUE` (default) plots the x-y line for `type = "dcalib"`.

`cuts` (`integer(1)`)
Number of cuts in (0,1) to plot `dcalib` over, default is `11`.

`theme` (`ggplot2::theme()`)
The `ggplot2::theme_minimal()` is applied by default to all plots.

`extend_quantile`
(`logical(1)`)
If `TRUE` then `dcalib` will impute NAs from predicted quantile function with the maximum observed outcome time, e.g. if the last predicted survival probability is greater than 0.1, then the last predicted cdf is smaller than 0.9 so F^1(0.9) = NA, this would be imputed with max(times). Default is `FALSE`.

`...` (any): Additional arguments, currently unused.

References

Haider H, Hoehn B, Davis S, Greiner R (2020). "Effective Ways to Build and Evaluate Individual Survival Distributions." *Journal of Machine Learning Research*, **21**(85), 1-63. `https://jmlr.org/papers/v21/18-772.html`.

Examples

```
library(mlr3)
library(mlr3proba)
library(mlr3viz)

learn = lrn("surv.coxph")
task = tsk("unemployment")
p = learn$train(task, row_ids = 1:300)$predict(task, row_ids = 301:400)

# calibration by comparison of average prediction to Kaplan-Meier
autoplot(p, type = "calib", task = task, row_ids = 301:400)

# Distribution-calibration (D-Calibration)
autoplot(p, type = "dcalib")
```

```
# Predictions
autoplot(p, type = "preds")
```

autoplot.TaskDens *Plot for Density Tasks*

Description

Generates plots for TaskDens.

Usage

```
## S3 method for class 'TaskDens'
autoplot(object, type = "dens", theme = theme_minimal(), ...)
```

Arguments

object	(TaskDens).
type	(character(1)): Type of the plot. Available choices: • "dens": histogram density estimator (default) with ggplot2::geom_histogram(). • "freq": histogram frequency plot with ggplot2::geom_histogram(). • "overlay": histogram with overlaid density plot with ggplot2::geom_histogram() and ggplot2::geom_density(). • "freqpoly": frequency polygon plot with ggplot2::geom_freqpoly.
theme	(ggplot2::theme()) The ggplot2::theme_minimal() is applied by default to all plots.
...	(any): Additional arguments, possibly passed down to the underlying plot functions.

Value

ggplot2::ggplot() object.

Examples

```
library(mlr3)
library(mlr3proba)
library(mlr3viz)
library(ggplot2)
task = tsk("precip")

head(fortify(task))
autoplot(task, bins = 15)
autoplot(task, type = "freq", bins = 15)
autoplot(task, type = "overlay", bins = 15)
autoplot(task, type = "freqpoly", bins = 15)
```

autoplot.TaskSurv *Plot for Survival Tasks*

Description

Generates plots for TaskSurv, depending on argument type:

- "target": Calls GGally::ggsurv() on a survival::survfit() object. This computes the **Kaplan-Meier survival curve** for the observations if this task.
- "duo": Passes data and additional arguments down to GGally::ggduo(). columnsX is target, columnsY is features.
- "pairs": Passes data and additional arguments down to GGally::ggpairs(). Color is set to target column.

Usage

```
## S3 method for class 'TaskSurv'
autoplot(
  object,
  type = "target",
  theme = theme_minimal(),
  reverse = FALSE,
  ...
)
```

Arguments

object	(TaskSurv).
type	(character(1)): Type of the plot. See above for available choices.
theme	(ggplot2::theme()) The ggplot2::theme_minimal() is applied by default to all plots.
reverse	(logical()) If TRUE and type = 'target', it plots the Kaplan-Meier curve of the censoring distribution. Default is FALSE.
...	(any): Additional arguments. rhs is passed down to $formula of TaskSurv for stratification for type "target". Other arguments are passed to the respective underlying plot functions.

Value

ggplot2::ggplot() object.

Examples

```
library(mlr3)
library(mlr3viz)
library(mlr3proba)
library(ggplot2)

task = tsk("lung")

head(fortify(task))
autoplot(task) # KM
autoplot(task) # KM of the censoring distribution
autoplot(task, rhs = "sex")
autoplot(task, type = "duo")
```

breslow *Survival probabilities using Breslow's estimator*

Description

Helper function to compose a survival distribution (or cumulative hazard) from the relative risk predictions (linear predictors, `lp`) of a **proportional hazards** model (e.g. a Cox-type model).

Usage

```
breslow(times, status, lp_train, lp_test, eval_times = NULL, type = "surv")
```

Arguments

`times` (`numeric()`)
Vector of times (train set).

`status` (`numeric()`)
Vector of status indicators (train set). For each observation in the train set, this should be 0 (alive/censored) or 1 (dead).

`lp_train` (`numeric()`)
Vector of linear predictors (train set). These are the relative score predictions ($lp = \hat{\beta} X_{train}$) from a proportional hazards model on the train set.

`lp_test` (`numeric()`)
Vector of linear predictors (test set). These are the relative score predictions ($lp = \hat{\beta} X_{test}$) from a proportional hazards model on the test set.

`eval_times` (`numeric()`)
Vector of times to compute survival probabilities. If `NULL` (default), the unique and sorted `times` from the train set will be used, otherwise the unique and sorted `eval_times`.

`type` (`character()`)
Type of prediction estimates. Default is `surv` which returns the survival probabilities $S_i(t)$ for each test observation i. If `cumhaz`, the function returns the estimated cumulative hazards $H_i(t)$.

Details

We estimate the survival probability of individual i (from the test set), at time point t as follows:

$$S_i(t) = e^{-H_i(t)} = e^{-\hat{H}_0(t) \times e^{lp_i}}$$

where:

- $H_i(t)$ is the cumulative hazard function for individual i
- $\hat{H}_0(t)$ is Breslow's estimator for the **cumulative baseline hazard**. Estimation requires the training set's `times` and `status` as well the risk predictions (`lp_train`).
- lp_i is the risk prediction (linear predictor) of individual i on the test set.

Breslow's approach uses a non-parametric maximum likelihood estimation of the cumulative baseline hazard function:

$$\hat{H}_0(t) = \sum_{i=1}^{n} \frac{I(T_i \le t)\delta_i}{\sum_{j \in R_i} e^{lp_j}}$$

where:

- t is the vector of time points (unique and sorted, from the train set)
- n is number of events (train set)
- T is the vector of event times (train set)
- δ is the status indicator (1 = event or 0 = censored)
- R_i is the risk set (number of individuals at risk just before event i)
- lp_j is the risk prediction (linear predictor) of individual j (who is part of the risk set R_i) on the train set.

We employ **constant interpolation** to estimate the cumulative baseline hazards, extending from the observed unique event times to the specified evaluation times (`eval_times`). Any values falling outside the range of the estimated times are assigned as follows:

$$\hat{H}_0(eval\_times < min(t)) = 0$$

and

$$\hat{H}_0(eval\_times > max(t)) = \hat{H}_0(max(t))$$

Note that in the rare event of `lp` predictions being `Inf` or `-Inf`, the resulting cumulative hazard values become `NaN`, which we substitute with `Inf` (and corresponding survival probabilities take the value of 0).

For similar implementations, see `gbm::basehaz.gbm()`, `C060::basesurv()` and `xgboost.surv::sgb_bhaz()`.

Value

a `matrix` (obs x times). Number of columns is equal to `eval_times` and number of rows is equal to the number of test observations (i.e. the length of the `lp_test` vector). Depending on the `type` argument, the matrix can have either survival probabilities (0-1) or cumulative hazard estimates (0-`Inf`).

References

Cox DR (1972). “Regression Models and Life-Tables.” *Journal of the Royal Statistical Society: Series B (Methodological)*, **34**(2), 187–202. doi:10.1111/j.25176161.1972.tb00899.x.

Lin, Y. D (2007). “On the Breslow estimator.” *Lifetime Data Analysis*, **13**(4), 471-480. doi:10.1007/s109850079048y.

Examples

```
task = tsk("rats")
part = partition(task, ratio = 0.8)

learner = lrn("surv.coxph")
learner$train(task, part$train)
p_train = learner$predict(task, part$train)
p_test = learner$predict(task, part$test)

surv = breslow(times = task$times(part$train), status = task$status(part$train),
               lp_train = p_train$lp, lp_test = p_test$lp)
head(surv)
```

gbcs *German Breast Cancer Study (GBCS) Dataset*

Description

gbcs dataset from Hosmer et al. (2008)

Usage

```
gbcs
```

Format

id Identification Code

diagdate Date of diagnosis.

recdate Date of recurrence free survival.

deathdate Date of death.

age Age at diagnosis (years).

menopause Menopausal status. 1 = Yes, 0 = No.

hormone Hormone therapy. 1 = Yes. 0 = No.

size Tumor size (mm).

grade Tumor grade (1-3).

nodes Number of lymph nodes.

prog_recp Number of progesterone receptors.

estrg_recp Number of estrogen receptors.

rectime Time to recurrence (days).

censrec Recurrence status. 1 = Recurrence. 0 = Censored.

survtime Time to death (days).

censdead Censoring status. 1 = Death. 0 = Censored.

Source

`https://onlinelibrary.wiley.com/doi/book/10.1002/9780470258019`

References

Hosmer, D.W. and Lemeshow, S. and May, S. (2008) Applied Survival Analysis: Regression Modeling of Time to Event Data: Second Edition, John Wiley and Sons Inc., New York, NY

`get_mortality` *Calculate the expected mortality risks from a survival matrix*

Description

Many methods can be used to reduce a discrete survival distribution prediction (i.e. matrix) to a relative risk / ranking prediction, see Sonabend et al. (2022).

This function calculates a relative risk score as the sum of the predicted cumulative hazard function, also called **ensemble/expected mortality**. This risk score can be loosely interpreted as the expected number of deaths for patients with similar characteristics, see Ishwaran et al. (2008) and has no model or survival distribution assumptions.

Usage

```
get_mortality(x)
```

Arguments

`x`	`(matrix())` A survival matrix where rows are the (predicted) observations and columns the time-points. For more details, see assert_surv_matrix.

Value

a `numeric` vector of the mortality risk scores, one per row of the input survival matrix.

References

Sonabend, Raphael, Bender, Andreas, Vollmer, Sebastian (2022). "Avoiding C-hacking when evaluating survival distribution predictions with discrimination measures." *Bioinformatics*. ISSN 1367-4803, doi:10.1093/BIOINFORMATICS/BTAC451, `https://academic.oup.com/bioinformatics/advance-article/doi/10.1093/bioinformatics/btac451/6640155`.

Ishwaran, Hemant, Kogalur, B U, Blackstone, H E, Lauer, S M, others (2008). "Random survival forests." *The Annals of applied statistics*, **2**(3), 841–860.

Examples

```
n = 10 # number of observations
k = 50 # time points

# Create the matrix with random values between 0 and 1
mat = matrix(runif(n * k, min = 0, max = 1), nrow = n, ncol = k)

# transform it to a survival matrix
surv_mat = t(apply(mat, 1L, function(row) sort(row, decreasing = TRUE)))
colnames(surv_mat) = 1:k # time points

# get mortality scores (the larger, the more risk)
mort = get_mortality(surv_mat)
mort
```

`grace` *GRACE 1000 Dataset*

Description

grace dataset from Hosmer et al. (2008)

Usage

```
grace
```

Format

id Identification Code

days Follow up time.

death Censoring indicator. 1 = Death. 0 = Censored.

revasc Revascularization Performed. 1 = Yes. 0 = No.

revascdays Days to revascularization after admission.

los Length of hospital stay (days).

age Age at admission (years).

sysbp Systolic blood pressure on admission (mm Hg).

stchange ST-segment deviation on index ECG. 1 = Yes. 0 = No.

Source

https://onlinelibrary.wiley.com/doi/book/10.1002/9780470258019

References

Hosmer, D.W. and Lemeshow, S. and May, S. (2008) Applied Survival Analysis: Regression Modeling of Time to Event Data: Second Edition, John Wiley and Sons Inc., New York, NY

LearnerDens *Density Learner*

Description

This Learner specializes Learner for density estimation problems:

- `task_type` is set to `"dens"`
- Creates Predictions of class PredictionDens.
- Possible values for `predict_types` are:
  - `"pdf"`: Evaluates estimated probability density function for each value in the test set.
  - `"cdf"`: Evaluates estimated cumulative distribution function for each value in the test set.

Super class

`mlr3::Learner` -> `LearnerDens`

Methods

Public methods:

- `LearnerDens$new()`
- `LearnerDens$clone()`

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

Usage:

```
LearnerDens$new(
  id,
  param_set = ps(),
  predict_types = "cdf",
  feature_types = character(),
  properties = character(),
  packages = character(),
  label = NA_character_,
  man = NA_character_
)
```

Arguments:

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

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`predict_types` (`character()`)
Supported predict types. Must be a subset of `mlr_reflections$learner_predict_types`.

`feature_types` (`character()`)
Feature types the learner operates on. Must be a subset of `mlr_reflections$task_feature_types`.

`properties` (`character()`)
Set of properties of the Learner. Must be a subset of `mlr_reflections$learner_properties`. The following properties are currently standardized and understood by learners in **mlr3**:

- `"missings"`: The learner can handle missing values in the data.
- `"weights"`: The learner supports observation weights.
- `"importance"`: The learner supports extraction of importance scores, i.e. comes with an `$importance()` extractor function (see section on optional extractors in Learner).
- `"selected_features"`: The learner supports extraction of the set of selected features, i.e. comes with a `$selected_features()` extractor function (see section on optional extractors in Learner).
- `"oob_error"`: The learner supports extraction of estimated out of bag error, i.e. comes with a `oob_error()` extractor function (see section on optional extractors in Learner).

`packages` (`character()`)
Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.

`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()`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Learner: `LearnerSurv`

Examples

```
library(mlr3)
# get all density learners from mlr_learners:
lrns = mlr_learners$mget(mlr_learners$keys("^dens"))
names(lrns)

# get a specific learner from mlr_learners:
```

```
mlr_learners$get("dens.hist")
lrn("dens.hist")
```

`LearnerSurv` *Survival Learner*

Description

This Learner specializes Learner for survival problems:

- `task_type` is set to `"surv"`
- Creates Predictions of class PredictionSurv.
- Possible values for `predict_types` are:
  - `"distr"`: Predicts a probability distribution for each observation in the test set, uses distr6.
  - `"lp"`: Predicts a linear predictor for each observation in the test set.
  - `"crank"`: Predicts a continuous ranking for each observation in the test set.
  - `"response"`: Predicts a survival time for each observation in the test set.

Super class

`mlr3::Learner` -> `LearnerSurv`

Methods

Public methods:

- `LearnerSurv$new()`
- `LearnerSurv$clone()`

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

Usage:

```
LearnerSurv$new(
  id,
  param_set = ps(),
  predict_types = "distr",
  feature_types = character(),
  properties = character(),
  packages = character(),
  label = NA_character_,
  man = NA_character_
)
```

Arguments:

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

param_set (paradox::ParamSet)
Set of hyperparameters.

predict_types (character())
Supported predict types. Must be a subset of mlr_reflections$learner_predict_types.

feature_types (character())
Feature types the learner operates on. Must be a subset of mlr_reflections$task_feature_types.

properties (character())
Set of properties of the Learner. Must be a subset of mlr_reflections$learner_properties. The following properties are currently standardized and understood by learners in **mlr3**:

- "missings": The learner can handle missing values in the data.
- "weights": The learner supports observation weights.
- "importance": The learner supports extraction of importance scores, i.e. comes with an $importance() extractor function (see section on optional extractors in Learner).
- "selected_features": The learner supports extraction of the set of selected features, i.e. comes with a $selected_features() extractor function (see section on optional extractors in Learner).
- "oob_error": The learner supports extraction of estimated out of bag error, i.e. comes with a oob_error() extractor function (see section on optional extractors in Learner).

packages (character())
Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via requireNamespace().

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().

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

Other Learner: LearnerDens

Examples

```
library(mlr3)
# get all survival learners from mlr_learners:
lrns = mlr_learners$mget(mlr_learners$keys("^surv"))
names(lrns)

# get a specific learner from mlr_learners:
mlr_learners$get("surv.coxph")
lrn("surv.coxph")
```

MeasureDens *Density Measure*

Description

This measure specializes Measure for survival problems.

- `task_type` is set to `"dens"`.
- Possible values for `predict_type` are `"pdf"` and `"cdf"`.

Predefined measures can be found in the dictionary mlr3::mlr_measures.

Super class

`mlr3::Measure` -> `MeasureDens`

Methods

Public methods:

- `MeasureDens$new()`

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

Usage:

```
MeasureDens$new(
  id,
  param_set = ps(),
  range,
  minimize = NA,
  aggregator = NULL,
  properties = character(),
  predict_type = "pdf",
  task_properties = character(),
  packages = character(),
  label = NA_character_,
  man = NA_character_
)
```

Arguments:

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

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`range` (`numeric(2)`)
Feasible range for this measure as `c(lower_bound, upper_bound)`. Both bounds may be infinite.

`minimize` (`logical(1)`)
: Set to `TRUE` if good predictions correspond to small values, and to `FALSE` if good predictions correspond to large values. If set to `NA` (default), tuning this measure is not possible.

`aggregator` (`function(x)`)
: Function to aggregate individual performance scores `x` where `x` is a numeric vector. If `NULL`, defaults to `mean()`.

`properties` (`character()`)
: Properties of the measure. Must be a subset of mlr_reflections$measure_properties. Supported by `mlr3`:

  - `"requires_task"` (requires the complete Task),
  - `"requires_learner"` (requires the trained Learner),
  - `"requires_train_set"` (requires the training indices from the Resampling), and
  - `"na_score"` (the measure is expected to occasionally return `NA` or `NaN`).

`predict_type` (`character(1)`)
: Required predict type of the Learner. Possible values are stored in mlr_reflections$learner_predict_types.

`task_properties` (`character()`)
: Required task properties, see Task.

`packages` (`character()`)
: Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.

`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()`.

See Also

Default density measures: `dens.logloss`

Other Measure: `MeasureSurv`

`MeasureSurv` *Survival Measure*

Description

This measure specializes Measure for survival problems.

- `task_type` is set to `"surv"`.
- Possible values for `predict_type` are `"distr"`, `"lp"`, `"crank"`, and `"response"`.

Predefined measures can be found in the dictionary mlr3::mlr_measures.

Super class

`mlr3::Measure` -> `MeasureSurv`

Methods

Public methods:

- `MeasureSurv$new()`
- `MeasureSurv$print()`

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

Usage:

```
MeasureSurv$new(
  id,
  param_set = ps(),
  range,
  minimize = NA,
  aggregator = NULL,
  properties = character(),
  predict_type = "distr",
  task_properties = character(),
  packages = character(),
  label = NA_character_,
  man = NA_character_,
  se = FALSE
)
```

Arguments:

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

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`range` (`numeric(2)`)
: Feasible range for this measure as `c(lower_bound, upper_bound)`. Both bounds may be infinite.

`minimize` (`logical(1)`)
: Set to `TRUE` if good predictions correspond to small values, and to `FALSE` if good predictions correspond to large values. If set to `NA` (default), tuning this measure is not possible.

`aggregator` (`function(x)`)
: Function to aggregate individual performance scores `x` where `x` is a numeric vector. If `NULL`, defaults to `mean()`.

`properties` (`character()`)
: Properties of the measure. Must be a subset of mlr_reflections$measure_properties. Supported by `mlr3`:

- `"requires_task"` (requires the complete Task),
- `"requires_learner"` (requires the trained Learner),
- `"requires_train_set"` (requires the training indices from the Resampling), and
- `"na_score"` (the measure is expected to occasionally return `NA` or `NaN`).

`predict_type` (`character(1)`)
: Required predict type of the Learner. Possible values are stored in mlr_reflections$learner_predict_types.

`task_properties` (`character()`)
: Required task properties, see Task.

`packages` (`character()`)
Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.

`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()`.

`se` If `TRUE` then returns standard error of the measure otherwise returns the mean (default).

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

Usage:

```
MeasureSurv$print()
```

See Also

Default survival measures: `surv.cindex`

Other Measure: `MeasureDens`

`MeasureSurvAUC` *Abstract Class for survAUC Measures*

Description

This is an abstract class that should not be constructed directly.

Parameter details

- `integrated` (`logical(1)`)
If `TRUE` (default), returns the integrated score (eg across time points); otherwise, not integrated (eg at a single time point).

- `times` (`numeric()`)
If `integrated` == `TRUE` then a vector of time-points over which to integrate the score. If `integrated` == `FALSE` then a single time point at which to return the score.

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvAUC`

Methods

Public methods:

- `MeasureSurvAUC$new()`
- `MeasureSurvAUC$clone()`

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

Usage:

```
MeasureSurvAUC$new(
  id,
  properties = character(),
  label = NA_character_,
  man = NA_character_,
  param_set = ps()
)
```

Arguments:

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

`properties` (`character()`)
Properties of the measure. Must be a subset of mlr_reflections$measure_properties. Supported by `mlr3`:

- `"requires_task"` (requires the complete Task),
- `"requires_learner"` (requires the trained Learner),
- `"requires_train_set"` (requires the training indices from the Resampling), and
- `"na_score"` (the measure is expected to occasionally return `NA` or `NaN`).

`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()`.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

mlr_graphs_crankcompositor

Estimate Survival crank Predict Type Pipeline

Description

Wrapper around PipeOpCrankCompositor to simplify Graph creation.

Usage

```
pipeline_crankcompositor(
  learner,
  method = c("mort"),
  overwrite = FALSE,
  graph_learner = FALSE
)
```

Arguments

learner
: [mlr3::Learner]|[mlr3pipelines::PipeOp]|[mlr3pipelines::Graph]
Either a Learner which will be wrapped in mlr3pipelines::PipeOpLearner, a PipeOp which will be wrapped in mlr3pipelines::Graph or a Graph itself. Underlying Learner should be LearnerSurv.

method
: (character(1))
Determines what method should be used to produce a continuous ranking from the distribution. Currently only mort is supported, which is the sum of the cumulative hazard, also called *expected/ensemble mortality*, see Ishwaran et al. (2008). For more details, see get_mortality().

overwrite
: (logical(1))
If FALSE (default) and the prediction already has a crank prediction, then the compositor returns the input prediction unchanged. If TRUE, then the crank will be overwritten.

graph_learner
: (logical(1))
If TRUE returns wraps the Graph as a GraphLearner otherwise (default) returns as a Graph.

Value

mlr3pipelines::Graph or mlr3pipelines::GraphLearner

Dictionary

This Graph can be instantiated via the dictionary mlr_graphs or with the associated sugar function ppl():

```
mlr_graphs$get("crankcompositor")
ppl("crankcompositor")
```

See Also

Other pipelines: mlr_graphs_distrcompositor, mlr_graphs_probregr, mlr_graphs_responsecompositor, mlr_graphs_survaverager, mlr_graphs_survbagging, mlr_graphs_survtoclassif_IPCW, mlr_graphs_survtoclass

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  task = tsk("lung")
  part = partition(task)

  # change the crank prediction type of a Cox's model predictions
  grlrn = ppl(
    "crankcompositor",
    learner = lrn("surv.coxph"),
    method = "mort",
    overwrite = TRUE,
    graph_learner = TRUE
  )
  grlrn$train(task, part$train)
  grlrn$predict(task, part$test)

## End(Not run)
```

mlr_graphs_distrcompositor

Estimate Survival distr Predict Type Pipeline

Description

Wrapper around PipeOpDistrCompositor or PipeOpBreslow to simplify Graph creation.

[Experimental]

Usage

```
pipeline_distrcompositor(
  learner,
  estimator = "kaplan",
  form = "aft",
  overwrite = FALSE,
  scale_lp = FALSE,
  graph_learner = FALSE
)
```

Arguments

`learner` [mlr3::Learner]|[mlr3pipelines::PipeOp]|[mlr3pipelines::Graph]
Either a `Learner` which will be wrapped in mlr3pipelines::PipeOpLearner, a `PipeOp` which will be wrapped in mlr3pipelines::Graph or a `Graph` itself. Underlying `Learner` should be LearnerSurv.

`estimator` (character(1))
One of `kaplan` (default), `nelson` or `breslow`, corresponding to the Kaplan-Meier, Nelson-Aalen and Breslow estimators respectively. Used to estimate the baseline survival distribution.

`form` (character(1))
One of `aft` (default), `ph`, or `po`, corresponding to accelerated failure time, proportional hazards, and proportional odds respectively. Used to determine the form of the composed survival distribution. Ignored if estimator is `breslow`.

`overwrite` (logical(1))
If `FALSE` (default) then if the `learner` already has a `distr`, the compositor does nothing. If `TRUE` then the `distr` is overwritten by the compositor if already present, which may be required for changing the prediction `distr` from one model form to another.

`scale_lp` (logical(1))
If `TRUE` and `form` is `"aft"`, the linear predictor scores are scaled before the composition. Experimental option, see more details on PipeOpDistrCompositor. Default is `FALSE`.

`graph_learner` (logical(1))
If `TRUE` returns wraps the Graph as a GraphLearner otherwise (default) returns as a `Graph`.

Value

mlr3pipelines::Graph or mlr3pipelines::GraphLearner

Dictionary

This Graph can be instantiated via the dictionary mlr_graphs or with the associated sugar function ppl():

```
mlr_graphs$get("distrcompositor")
ppl("distrcompositor")
```

See Also

Other pipelines: `mlr_graphs_crankcompositor`, `mlr_graphs_probregr`, `mlr_graphs_responsecompositor`, `mlr_graphs_survaverager`, `mlr_graphs_survbagging`, `mlr_graphs_survtoclassif_IPCW`, `mlr_graphs_survtoclassi`

Examples

```
## Not run:
  library(mlr3pipelines)
  # let's change the distribution prediction of Cox (Breslow-based) to an AFT form:
  task = tsk("rats")
  grlrn = ppl(
    "distrcompositor",
    learner = lrn("surv.coxph"),
    estimator = "kaplan",
    form = "aft",
    overwrite = TRUE,
    graph_learner = TRUE
  )
  grlrn$train(task)
  grlrn$predict(task)

## End(Not run)
```

`mlr_graphs_probregr` *Estimate Regression distr Predict Type Pipeline*

Description

Wrapper around PipeOpProbregr to simplify Graph creation.

[Experimental]

Usage

```
pipeline_probregr(
  learner,
  learner_se = NULL,
  dist = "Uniform",
  graph_learner = FALSE
)
```

Arguments

`learner`
: `[mlr3::Learner]|[mlr3pipelines::PipeOp]|[mlr3pipelines::Graph]`
Either a `Learner` which will be wrapped in mlr3pipelines::PipeOpLearner, a `PipeOp` which will be wrapped in mlr3pipelines::Graph or a `Graph` itself. Underlying `Learner` should be LearnerRegr.

`learner_se`
: `[mlr3::Learner]|[mlr3pipelines::PipeOp]`
Optional LearnerRegr with predict_type se to estimate the standard error. If left `NULL` then `learner` must have se in predict_types.

`dist`
: `(character(1))`
Location-scale distribution to use for composition. Current possibilities are' `"Cauchy"`, `"Gumbel"`, `"Laplace"`, `"Logistic"`, `"Normal"`, `"Uniform"`. Default is `"Uniform"`.

`graph_learner`
: `(logical(1))`
If `TRUE` returns wraps the Graph as a GraphLearner otherwise (default) returns as a `Graph`.

Value

mlr3pipelines::Graph or mlr3pipelines::GraphLearner

Dictionary

This Graph can be instantiated via the dictionary mlr_graphs or with the associated sugar function ppl():

```
mlr_graphs$get("probregr")
ppl("probregr")
```

See Also

Other pipelines: `mlr_graphs_crankcompositor`, `mlr_graphs_distrcompositor`, `mlr_graphs_responsecompositor`, `mlr_graphs_survaverager`, `mlr_graphs_survbagging`, `mlr_graphs_survtoclassif_IPCW`, `mlr_graphs_survtoclassi`

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  task = tsk("boston_housing")

  # method 1 - same learner for response and se
  pipe = ppl(
    "probregr",
    learner = lrn("regr.featureless", predict_type = "se"),
    dist = "Uniform"
  )
  pipe$train(task)
  pipe$predict(task)

  # method 2 - different learners for response and se
  pipe = ppl(
    "probregr",
    learner = lrn("regr.rpart"),
    learner_se = lrn("regr.featureless", predict_type = "se"),
    dist = "Normal"
  )
  pipe$train(task)
  pipe$predict(task)

## End(Not run)
```

mlr_graphs_responsecompositor

Estimate Survival Time/Response Predict Type Pipeline

Description

Wrapper around PipeOpResponseCompositor to simplify Graph creation.

Usage

```
pipeline_responsecompositor(
  learner,
  method = "rmst",
  tau = NULL,
  add_crank = FALSE,
  overwrite = FALSE,
  graph_learner = FALSE
)
```

Arguments

learner
: [mlr3::Learner]|[mlr3pipelines::PipeOp]|[mlr3pipelines::Graph]
Either a Learner which will be wrapped in mlr3pipelines::PipeOpLearner, a PipeOp which will be wrapped in mlr3pipelines::Graph or a Graph itself. Underlying Learner should be LearnerSurv.

method
: (character(1))
Determines what method should be used to produce a survival time (response) from the survival distribution. Available methods are "rmst" and "median", corresponding to the *restricted mean survival time* and the *median survival time* respectively.

tau
: (numeric(1))
Determines the time point up to which we calculate the restricted mean survival time (works only for the "rmst" method). If NULL (default), all the available time points in the predicted survival distribution will be used.

add_crank
: (logical(1))
If TRUE then crank predict type will be set as -response (as higher survival times correspond to lower risk). Works only if overwrite is TRUE.

overwrite
: (logical(1))
If FALSE (default) and the prediction already has a response prediction, then the compositor returns the input prediction unchanged. If TRUE, then the response (and the crank, if add_crank is TRUE) will be overwritten.

graph_learner
: (logical(1))
If TRUE returns wraps the Graph as a GraphLearner otherwise (default) returns as a Graph.

Value

mlr3pipelines::Graph or mlr3pipelines::GraphLearner

Dictionary

This Graph can be instantiated via the dictionary mlr_graphs or with the associated sugar function ppl():

```
mlr_graphs$get("responsecompositor")
ppl("responsecompositor")
```

See Also

Other pipelines: mlr_graphs_crankcompositor, mlr_graphs_distrcompositor, mlr_graphs_probregr, mlr_graphs_survaverager, mlr_graphs_survbagging, mlr_graphs_survtoclassif_IPCW, mlr_graphs_survtoclassi

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  task = tsk("lung")
  part = partition(task)

  # add survival time prediction type to the predictions of a Cox model
  grlrn = ppl(
    "responsecompositor",
    learner = lrn("surv.coxph"),
    method = "rmst",
    overwrite = TRUE,
    graph_learner = TRUE
  )
  grlrn$train(task, part$train)
  grlrn$predict(task, part$test)

## End(Not run)
```

`mlr_graphs_survaverager`

Survival Prediction Averaging Pipeline

Description

Wrapper around PipeOpSurvAvg to simplify Graph creation.

Usage

```
pipeline_survaverager(learners, param_vals = list(), graph_learner = FALSE)
```

Arguments

learners (list())
List of LearnerSurvs to average.

param_vals (list())
Parameters, including weights, to pass to PipeOpSurvAvg.

graph_learner (logical(1))
If TRUE returns wraps the Graph as a GraphLearner otherwise (default) returns as a Graph.

Value

mlr3pipelines::Graph or mlr3pipelines::GraphLearner

Dictionary

This Graph can be instantiated via the dictionary mlr_graphs or with the associated sugar function ppl():

```
mlr_graphs$get("survaverager")
ppl("survaverager")
```

See Also

Other pipelines: mlr_graphs_crankcompositor, mlr_graphs_distrcompositor, mlr_graphs_probregr, mlr_graphs_responsecompositor, mlr_graphs_survbagging, mlr_graphs_survtoclassif_IPCW, mlr_graphs_survtoclassif_disctime

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  task = tsk("rats")
  pipe = ppl(
    "survaverager",
    learners = lrns(c("surv.kaplan", "surv.coxph")),
    param_vals = list(weights = c(0.1, 0.9)),
    graph_learner = FALSE
  )
  pipe$train(task)
  pipe$predict(task)

## End(Not run)
```

mlr_graphs_survbagging

Survival Prediction Averaging Pipeline

Description

Wrapper around PipeOpSubsample and PipeOpSurvAvg to simplify Graph creation.

Usage

```
pipeline_survbagging(
  learner,
  iterations = 10,
  frac = 0.7,
  avg = TRUE,
  weights = 1,
  graph_learner = FALSE
)
```

Arguments

learner	[mlr3::Learner]|[mlr3pipelines::PipeOp]|[mlr3pipelines::Graph] Either a Learner which will be wrapped in mlr3pipelines::PipeOpLearner, a PipeOp which will be wrapped in mlr3pipelines::Graph or a Graph itself. Underlying Learner should be LearnerSurv.
iterations	(integer(1)) Number of bagging iterations. Defaults to 10.
frac	(numeric(1)) Percentage of rows to keep during subsampling. See PipeOpSubsample for more information. Defaults to 0.7.
avg	(logical(1)) If TRUE (default) predictions are aggregated with PipeOpSurvAvg, otherwise returned as multiple predictions. Can only be FALSE if graph_learner = FALSE.
weights	(numeric()) Weights for model avering, ignored if avg = FALSE. Default is uniform weighting, see PipeOpSurvAvg.
graph_learner	(logical(1)) If TRUE returns wraps the Graph as a GraphLearner otherwise (default) returns as a Graph.

Details

Bagging (Bootstrap AGGregatING) is the process of bootstrapping data and aggregating the final predictions. Bootstrapping splits the data into B smaller datasets of a given size and is performed with PipeOpSubsample. Aggregation is the sample mean of deterministic predictions and a MixtureDistribution of distribution predictions. This can be further enhanced by using a weighted average by supplying weights.

Value

mlr3pipelines::Graph or mlr3pipelines::GraphLearner

Dictionary

This Graph can be instantiated via the dictionary mlr_graphs or with the associated sugar function ppl():

```
mlr_graphs$get("survbagging")
ppl("survbagging")
```

See Also

Other pipelines: `mlr_graphs_crankcompositor`, `mlr_graphs_distrcompositor`, `mlr_graphs_probregr`, `mlr_graphs_responsecompositor`, `mlr_graphs_survaverager`, `mlr_graphs_survtoclassif_IPCW`, `mlr_graphs_survtoclassif_disctime`

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  task = tsk("rats")
  pipe = ppl(
    "survbagging",
    learner = lrn("surv.coxph"),
    iterations = 5,
    graph_learner = FALSE
  )
  pipe$train(task)
  pipe$predict(task)

## End(Not run)
```

`mlr_graphs_survtoclassif_disctime` *Survival to Classification Reduction using Discrete Time Pipeline*

Description

Wrapper around PipeOpTaskSurvClassifDiscTime and PipeOpPredClassifSurvDiscTime to simplify Graph creation.

Usage

```
pipeline_survtoclassif_disctime(
  learner,
  cut = NULL,
  max_time = NULL,
  rhs = NULL,
  graph_learner = FALSE
)
```

Arguments

`learner` LearnerClassif
Classification learner to fit the transformed TaskClassif. `learner` must have `predict_type` of type `"prob"`.

`cut` (`numeric()`)
Split points, used to partition the data into intervals. If unspecified, all unique event times will be used. If `cut` is a single integer, it will be interpreted as the number of equidistant intervals from 0 until the maximum event time.

`max_time` (`numeric(1)`)
If cut is unspecified, this will be the last possible event time. All event times after max_time will be administratively censored at max_time.

`rhs` (`character(1)`)
Right-hand side of the formula to use with the learner. All features of the task are available as well as `tend` the upper bounds of the intervals created by `cut`. If `rhs` is unspecified, the formula of the task will be used.

`graph_learner` (`logical(1)`)
If `TRUE` returns wraps the Graph as a GraphLearner otherwise (default) returns as a `Graph`.

Details

The pipeline consists of the following steps:

1. PipeOpTaskSurvClassifDiscTime Converts TaskSurv to a TaskClassif.
2. A LearnerClassif is fit and predicted on the new `TaskClassif`.
3. PipeOpPredClassifSurvDiscTime transforms the resulting PredictionClassif to PredictionSurv.
4. Optionally: PipeOpModelMatrix is used to transform the formula of the task before fitting the learner.

Value

mlr3pipelines::Graph or mlr3pipelines::GraphLearner

Dictionary

This Graph can be instantiated via the dictionary mlr_graphs or with the associated sugar function ppl():

```
mlr_graphs$get("survtoclassif_disctime")
ppl("survtoclassif_disctime")
```

References

Tutz, Gerhard, Schmid, Matthias (2016). *Modeling Discrete Time-to-Event Data*, series Springer Series in Statistics. Springer International Publishing. ISBN 978-3-319-28156-8 978-3-319-28158-2, http://link.springer.com/10.1007/978-3-319-28158-2.

See Also

Other pipelines: mlr_graphs_crankcompositor, mlr_graphs_distrcompositor, mlr_graphs_probregr, mlr_graphs_responsecompositor, mlr_graphs_survaverager, mlr_graphs_survbagging, mlr_graphs_survtoclass

Examples

```
## Not run:
  library(mlr3)
  library(mlr3learners)
  library(mlr3pipelines)

  task = tsk("lung")
  part = partition(task)

  grlrn = ppl(
    "survtoclassif_disctime",
    learner = lrn("classif.log_reg"),
    cut = 4, # 4 equidistant time intervals
    graph_learner = TRUE
  )
  grlrn$train(task, row_ids = part$train)
  grlrn$predict(task, row_ids = part$test)

## End(Not run)
```

mlr_graphs_survtoclassif_IPCW

Survival to Classification Reduction using IPCW Pipeline

Description

Wrapper around PipeOpTaskSurvClassifIPCW and PipeOpPredClassifSurvIPCW to simplify Graph creation.

Usage

```
pipeline_survtoclassif_IPCW(
  learner,
  tau = NULL,
  eps = 0.001,
  graph_learner = FALSE
)
```

Arguments

`learner` LearnerClassif
Classification learner to fit the transformed TaskClassif.

`tau` (`numeric()`)
Predefined time point for IPCW. Observations with time larger than τ are censored. Must be less or equal to the maximum event time.

`eps` (`numeric()`)
Small value to replace $G(t) = 0$ censoring probabilities to prevent infinite weights (a warning is triggered if this happens).

`graph_learner` (`logical(1)`)
If `TRUE` returns wraps the Graph as a GraphLearner otherwise (default) returns as a `Graph`.

Details

The pipeline consists of the following steps:

1. PipeOpTaskSurvClassifIPCW Converts TaskSurv to a TaskClassif.
2. A LearnerClassif is fit and predicted on the new `TaskClassif`.
3. PipeOpPredClassifSurvIPCW transforms the resulting PredictionClassif to PredictionSurv.

Value

mlr3pipelines::Graph or mlr3pipelines::GraphLearner

Dictionary

This Graph can be instantiated via the dictionary mlr_graphs or with the associated sugar function ppl():

```
mlr_graphs$get("survtoclassif_IPCW")
ppl("survtoclassif_IPCW")
```

Additional alias id for pipeline construction:

```
ppl("survtoclassif_vock")
```

References

Vock, M D, Wolfson, Julian, Bandyopadhyay, Sunayan, Adomavicius, Gediminas, Johnson, E P, Vazquez-Benitez, Gabriela, O'Connor, J P (2016). "Adapting machine learning techniques to censored time-to-event health record data: A general-purpose approach using inverse probability of censoring weighting." *Journal of Biomedical Informatics*, **61**, 119–131. doi:10.1016/j.jbi.2016.03.009, `https://www.sciencedirect.com/science/article/pii/S1532046416000496`.

See Also

Other pipelines: `mlr_graphs_crankcompositor`, `mlr_graphs_distrcompositor`, `mlr_graphs_probregr`, `mlr_graphs_responsecompositor`, `mlr_graphs_survaverager`, `mlr_graphs_survbagging`, `mlr_graphs_survtoclassi`

Examples

```
## Not run:
  library(mlr3)
  library(mlr3learners)
  library(mlr3pipelines)

  task = tsk("lung")
  part = partition(task)

  grlrn = ppl(
    "survtoclassif_IPCW",
    learner = lrn("classif.rpart"),
    tau = 500, # Observations after 500 days are censored
    graph_learner = TRUE
  )
  grlrn$train(task, row_ids = part$train)
  pred = grlrn$predict(task, row_ids = part$test)
  pred # crank and distr at the cutoff time point included

  # score predictions
  pred$score() # C-index
  pred$score(msr("surv.brier", times = 500, integrated = FALSE)) # Brier score at tau

## End(Not run)
```

`mlr_graphs_survtoregr` *Survival to Regression Reduction Pipeline*

Description

Wrapper around multiple PipeOps to help in creation of complex survival reduction methods. Three reductions are currently implemented, see details. **[Experimental]**

Usage

```
pipeline_survtoregr(
  method = 1,
  regr_learner = lrn("regr.featureless"),
  distrcompose = TRUE,
  distr_estimator = lrn("surv.kaplan"),
  regr_se_learner = NULL,
  surv_learner = lrn("surv.coxph"),
  survregr_params = list(method = "ipcw", estimator = "kaplan", alpha = 1),
  distrcompose_params = list(form = "aft"),
  probregr_params = list(dist = "Uniform"),
  learnercv_params = list(resampling.method = "insample"),
  graph_learner = FALSE
)
```

Arguments

`method` (integer(1))
Reduction method to use, corresponds to those in details. Default is 1.

`regr_learner` LearnerRegr
Regression learner to fit to the transformed TaskRegr. If regr_se_learner is NULL in method 2, then regr_learner must have se predict_type.

`distrcompose` (logical(1))
For method 3 if TRUE (default) then PipeOpDistrCompositor is utilised to transform the deterministic predictions to a survival distribution.

`distr_estimator`
LearnerSurv
For methods 1 and 3 if distrcompose = TRUE then specifies the learner to estimate the baseline hazard, must have predict_type distr.

`regr_se_learner`
LearnerRegr
For method 2 if regr_learner is not used to predict the se then a LearnerRegr with se predict_type must be provided.

`surv_learner` LearnerSurv
For method 3, a LearnerSurv with lp predict type to estimate linear predictors.

`survregr_params`
(list())
Parameters passed to PipeOpTaskSurvRegr, default are survival to regression transformation via ipcw, with weighting determined by Kaplan-Meier and no additional penalty for censoring.

`distrcompose_params`
(list())
Parameters passed to PipeOpDistrCompositor, default is accelerated failure time model form.

`probregr_params`
(list())
Parameters passed to PipeOpProbregr, default is Uniform distribution for composition.

`learnercv_params`
: `(list())`
Parameters passed to PipeOpLearnerCV, default is to use insampling.

`graph_learner`
: `(logical(1))`
If `TRUE` returns wraps the Graph as a GraphLearner otherwise (default) returns as a `Graph`.

Details

Three reduction strategies are implemented, these are:

1. Survival to Deterministic Regression A
   (a) PipeOpTaskSurvRegr Converts TaskSurv to TaskRegr.
   (b) A LearnerRegr is fit and predicted on the new `TaskRegr`.
   (c) PipeOpPredRegrSurv transforms the resulting PredictionRegr to PredictionSurv.
2. Survival to Probabilistic Regression
   (a) PipeOpTaskSurvRegr Converts TaskSurv to TaskRegr.
   (b) A LearnerRegr is fit on the new `TaskRegr` to predict `response`, optionally a second `LearnerRegr` can be fit to predict `se`.
   (c) PipeOpProbregr composes a `distr` prediction from the learner(s).
   (d) PipeOpPredRegrSurv transforms the resulting PredictionRegr to PredictionSurv.
3. Survival to Deterministic Regression B
   (a) PipeOpLearnerCV cross-validates and makes predictions from a linear LearnerSurv with `lp` predict type on the original TaskSurv.
   (b) PipeOpTaskSurvRegr transforms the `lp` predictions into the target of a TaskRegr with the same features as the original TaskSurv.
   (c) A LearnerRegr is fit and predicted on the new `TaskRegr`.
   (d) PipeOpPredRegrSurv transforms the resulting PredictionRegr to PredictionSurv.
   (e) Optionally: PipeOpDistrCompositor is used to compose a `distr` predict_type from the predicted `lp` predict_type.

Interpretation:

1. Once a dataset has censoring removed (by a given method) then a regression learner can predict the survival time as the `response`.
2. This is a very similar reduction to the first method with the main difference being the distribution composition. In the first case this is composed in a survival framework by assuming a linear model form and baseline hazard estimator, in the second case the composition is in a regression framework. The latter case could result in problematic negative predictions and should therefore be interpreted with caution, however a wider choice of distributions makes it a more flexible composition.
3. This is a rarer use-case that bypasses censoring not be removing it but instead by first predicting the linear predictor from a survival model and fitting a regression model on these predictions. The resulting regression predictions can then be viewed as the linear predictors of the new data, which can ultimately be composed to a distribution.

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  task = tsk("rats")

  # method 1 with censoring deletion, compose to distribution
  pipe = ppl(
    "survtoregr",
    method = 1,
    regr_learner = lrn("regr.featureless"),
    survregr_params = list(method = "delete")
  )
  pipe$train(task)
  pipe$predict(task)

  # method 2 with censoring imputation (mrl), one regr learner
  pipe = ppl(
    "survtoregr",
    method = 2,
    regr_learner = lrn("regr.featureless", predict_type = "se"),
    survregr_params = list(method = "mrl")
  )
  pipe$train(task)
  pipe$predict(task)

  # method 3 with censoring omission and no composition, insample resampling
  pipe = ppl(
    "survtoregr",
    method = 3,
    regr_learner = lrn("regr.featureless"),
    distrcompose = FALSE,
    surv_learner = lrn("surv.coxph"),
    survregr_params = list(method = "omission")
  )
  pipe$train(task)
  pipe$predict(task)

## End(Not run)
```

`mlr_learners_dens.hist`

Histogram Density Estimator

Description

Calls `graphics::hist()` and the result is coerced to a distr6::Distribution.

Dictionary

This Learner can be instantiated via the dictionary mlr_learners or with the associated sugar function lrn():

```
LearnerDensHistogram$new()
mlr_learners$get("dens.hist")
lrn("dens.hist")
```

Meta Information

- Type: "dens"
- Predict Types: `pdf, cdf, distr`
- Feature Types: `integer, numeric`
- Properties: `-`
- Packages: **mlr3 mlr3proba distr6**

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerDens` -> `LearnerDensHistogram`

Methods

Public methods:

- `LearnerDensHistogram$new()`
- `LearnerDensHistogram$clone()`

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

Usage:

```
LearnerDensHistogram$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

Other density estimators: `mlr_learners_dens.kde`

mlr_learners_dens.kde *Kernel Density Estimator*

Description

Calls kernels implemented in distr6 and the result is coerced to a distr6::Distribution.

Details

The default bandwidth uses Silverman's rule-of-thumb for Gaussian kernels, however for non-Gaussian kernels it is recommended to use **mlr3tuning** to tune the bandwidth with cross-validation. Other density learners can be used for automated bandwidth selection. The default kernel is Epanechnikov (chosen to reduce dependencies).

Dictionary

This Learner can be instantiated via the dictionary mlr_learners or with the associated sugar function lrn():

```
LearnerDensKDE$new()
mlr_learners$get("dens.kde")
lrn("dens.kde")
```

Meta Information

- Type: "dens"
- Predict Types: `pdf, distr`
- Feature Types: `integer, numeric`
- Properties: `missings`
- Packages: **mlr3 mlr3proba distr6**

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerDens` -> `LearnerDensKDE`

Methods

Public methods:

- `LearnerDensKDE$new()`
- `LearnerDensKDE$clone()`

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

Usage:

```
LearnerDensKDE$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Silverman, W. B (1986). *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London.

See Also

Other density estimators: `mlr_learners_dens.hist`

---

mlr_learners_surv.coxph

Cox Proportional Hazards Survival Learner

Description

Calls `survival::coxph()`.

- lp is predicted by `survival::predict.coxph()`
- distr is predicted by `survival::survfit.coxph()`
- `crank` is identical to `lp`

Dictionary

This Learner can be instantiated via the dictionary mlr_learners or with the associated sugar function lrn():

```
LearnerSurvCoxPH$new()
mlr_learners$get("surv.coxph")
lrn("surv.coxph")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”, “lp”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3proba**, **survival**, **distr6**

Parameters

Id	Type	Default	Levels	Range
ties	character	efron	efron, breslow, exact	-
singular.ok	logical	TRUE	TRUE, FALSE	-
type	character	efron	efron, aalen, kalbfleisch-prentice	-
stype	integer	2		$[1, 2]$

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvCoxPH`

Methods

Public methods:

- `LearnerSurvCoxPH$new()`
- `LearnerSurvCoxPH$clone()`

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

Usage:

```
LearnerSurvCoxPH$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Cox DR (1972). “Regression Models and Life-Tables.” *Journal of the Royal Statistical Society: Series B (Methodological)*, **34**(2), 187–202. doi:10.1111/j.25176161.1972.tb00899.x.

See Also

Other survival learners: `mlr_learners_surv.kaplan`, `mlr_learners_surv.rpart`

mlr_learners_surv.kaplan

Kaplan-Meier Estimator Survival Learner

Description

Calls `survival::survfit()`.

- distr is predicted by estimating the survival function with `survival::survfit()`
- `crank` is predicted as the sum of the cumulative hazard function (expected mortality) derived from the survival distribution, `distr`

Dictionary

This Learner can be instantiated via the dictionary mlr_learners or with the associated sugar function lrn():

```
LearnerSurvKaplan$new()
mlr_learners$get("surv.kaplan")
lrn("surv.kaplan")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3proba**, **survival**, **distr6**

Parameters

Empty ParamSet

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvKaplan`

Methods

Public methods:

- `LearnerSurvKaplan$new()`
- `LearnerSurvKaplan$clone()`

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

Usage:

```
LearnerSurvKaplan$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Kaplan EL, Meier P (1958). “Nonparametric Estimation from Incomplete Observations.” *Journal of the American Statistical Association*, **53**(282), 457–481. doi:10.1080/01621459.1958.10501452.

See Also

Other survival learners: `mlr_learners_surv.coxph`, `mlr_learners_surv.rpart`

`mlr_learners_surv.rpart`

Rpart Survival Trees Survival Learner

Description

Calls `rpart::rpart()`.

- crank is predicted using `rpart::predict.rpart()`

Dictionary

This Learner can be instantiated via the dictionary mlr_learners or with the associated sugar function lrn():

```
LearnerSurvRpart$new()
mlr_learners$get("surv.rpart")
lrn("surv.rpart")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3proba**, **rpart**, **distr6**, **survival**

Parameters

Id	Type	Default	Levels	Range
parms	numeric	1		$(-\infty, \infty)$
minbucket	integer	-		$[1, \infty)$
minsplit	integer	20		$[1, \infty)$
cp	numeric	0.01		$[0, 1]$
maxcompete	integer	4		$[0, \infty)$
maxsurrogate	integer	5		$[0, \infty)$
maxdepth	integer	30		$[1, 30]$
usesurrogate	integer	2		$[0, 2]$
surrogatestyle	integer	0		$[0, 1]$
xval	integer	10		$[0, \infty)$
cost	untyped	-		-
keep_model	logical	FALSE	TRUE, FALSE	-

Initial parameter values

- `xval` is set to 0 in order to save some computation time.
- `model` has been renamed to `keep_model`.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvRpart`

Methods

Public methods:

- `LearnerSurvRpart$new()`
- `LearnerSurvRpart$importance()`
- `LearnerSurvRpart$selected_features()`
- `LearnerSurvRpart$clone()`

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

Usage:

```
LearnerSurvRpart$new()
```

Method `importance()`**:** The importance scores are extracted from the model slot `variable.importance`.

Usage:

```
LearnerSurvRpart$importance()
```

Returns: Named `numeric()`.

Method `selected_features()`**:** Selected features are extracted from the model slot `frame$var`.

Usage:

```
LearnerSurvRpart$selected_features()
```

Returns: `character()`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984). *Classification And Regression Trees*. Routledge. doi:10.1201/9781315139470.

See Also

Other survival learners: `mlr_learners_surv.coxph`, `mlr_learners_surv.kaplan`

`mlr_measures_dens.logloss` *Log Loss Density Measure*

Description

Calculates the cross-entropy, or logarithmic (log), loss.

Details

The Log Loss, in the context of probabilistic predictions, is defined as the negative log probability density function, f, evaluated at the observed value, y,

$$L(f, y) = -\log(f(y))$$

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureDensLogloss$new()
mlr_measures$get("dens.logloss")
msr("dens.logloss")
```

Parameters

Id	Type	Default	Range
eps	numeric	1e-15	$[0, 1]$

Meta Information

- Type: `"density"`
- Range: $[0, \infty)$
- Minimize: `TRUE`
- Required prediction: `pdf`

Parameter details

- `eps` (`numeric(1)`)
  Very small number to substitute zero values in order to prevent errors in e.g. log(0) and/or division-by-zero calculations. Default value is 1e-15.

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureDens` -> `MeasureDensLogloss`

Methods

Public methods:

- `MeasureDensLogloss$new()`
- `MeasureDensLogloss$clone()`

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

Usage:

```
MeasureDensLogloss$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

mlr_measures_regr.logloss

Log Loss Regression Measure

Description

Calculates the cross-entropy, or logarithmic (log), loss.

Details

The Log Loss, in the context of probabilistic predictions, is defined as the negative log probability density function, f, evaluated at the observed value, y,

$$L(f, y) = -\log(f(y))$$

Parameters

Id	Type	Default	Range
eps	numeric	1e-15	$[0, 1]$

Meta Information

- Type: `"regr"`
- Range: $[0, \infty)$
- Minimize: `TRUE`
- Required prediction: `distr`

Parameter details

- `eps` (`numeric(1)`)
  Very small number to substitute zero values in order to prevent errors in e.g. log(0) and/or division-by-zero calculations. Default value is 1e-15.

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureRegrLogloss`

Methods

Public methods:

- `MeasureRegrLogloss$new()`
- `MeasureRegrLogloss$clone()`

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

Usage:

```
MeasureRegrLogloss$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`mlr_measures_surv.calib_alpha`

Van Houwelingen's Calibration Alpha Survival Measure

Description

This calibration method is defined by estimating

$$\hat{\alpha} = \sum \delta_i / \sum H_i(T_i)$$

where δ is the observed censoring indicator from the test data, H_i is the predicted cumulative hazard, and T_i is the observed survival time (event or censoring).

The standard error is given by

$$\hat{\alpha_{se}} = exp(1/\sqrt{\sum \delta_i})$$

The model is well calibrated if the estimated $\hat{\alpha}$ coefficient (returned score) is equal to 1.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvCalibrationAlpha$new()
mlr_measures$get("surv.calib_alpha")
msr("surv.calib_alpha")
```

Parameters

Id	Type	Default	Levels	Range
eps	numeric	0.001		$[0, 1]$
se	logical	FALSE	TRUE, FALSE	-
method	character	ratio	ratio, diff	-
truncate	numeric	Inf		$(-\infty, \infty)$

Meta Information

- Type: `"surv"`
- Range: $(-\infty, \infty)$
- Minimize: `FALSE`
- Required prediction: `distr`

Parameter details

- `eps` (`numeric(1)`)
  Very small number to substitute zero values in order to prevent errors in e.g. log(0) and/or division-by-zero calculations. Default value is 0.001.

- `se` (`logical(1)`)
  If `TRUE` then return standard error of the measure, otherwise the score itself (default).
- `method` (`character(1)`)
  Returns $\hat{\alpha}$ if equal to `ratio` (default) and $|1 - \hat{\alpha}|$ if equal to `diff`. With `diff`, the output score can be minimized and for example be used for tuning purposes. This parameter takes effect only if `se` is `FALSE`.
- `truncate` (`double(1)`)
  This parameter controls the upper bound of the output score. We use `truncate = Inf` by default (so no truncation) and it's up to the user **to set this up reasonably** given the chosen `method`. Note that truncation may severely limit automated tuning with this measure using `method = diff`.

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvCalibrationAlpha`

Methods

Public methods:

- `MeasureSurvCalibrationAlpha$new()`
- `MeasureSurvCalibrationAlpha$clone()`

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

Usage:

```
MeasureSurvCalibrationAlpha$new(method = "ratio")
```

Arguments:

`method` defines which output score to return, see "Parameter details" section.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Van Houwelingen, C. H (2000). "Validation, calibration, revision and combination of prognostic survival models." *Statistics in Medicine*, **19**(24), 3401–3415. doi:10.1002/10970258(20001230)19:24<3401::AID-SIM554>3.0.CO;22.

See Also

Other survival measures: `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_t` `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other calibration survival measures: `mlr_measures_surv.calib_beta`, `mlr_measures_surv.dcalib`

Other distr survival measures: `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.intloglos` `mlr_measures_surv.logloss`, `mlr_measures_surv.rcll`, `mlr_measures_surv.schmid`

`mlr_measures_surv.calib_beta` *Van Houwelingen's Calibration Beta Survival Measure*

Description

This calibration method fits the predicted linear predictor from a Cox PH model as the only predictor in a new Cox PH model with the test data as the response.

$$h(t|x) = h_0(t)exp(\beta \times lp)$$

where lp is the predicted linear predictor on the test data.

The model is well calibrated if the estimated $\hat{\beta}$ coefficient (returned score) is equal to 1.

Note: Assumes fitted model is Cox PH (i.e. has an `lp` prediction type).

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvCalibrationBeta$new()
mlr_measures$get("surv.calib_beta")
msr("surv.calib_beta")
```

Parameters

Id	Type	Default	Levels
se	logical	FALSE	TRUE, FALSE
method	character	ratio	ratio, diff

Meta Information

- Type: "surv"
- Range: $(-\infty, \infty)$
- Minimize: FALSE
- Required prediction: lp

Parameter details

- se (logical(1))
  If TRUE then return standard error of the measure which is the standard error of the estimated coefficient $se_{\hat{\beta}}$ from the Cox PH model. If FALSE (default) then returns the estimated coefficient $\hat{\beta}$.
- method (character(1))
  Returns $\hat{\beta}$ if equal to ratio (default) and $|1 - \hat{\beta}|$ if diff. With diff, the output score can be minimized and for example be used for tuning purposes. This parameter takes effect only if se is FALSE.

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> MeasureSurvCalibrationBeta

Methods

Public methods:

- MeasureSurvCalibrationBeta$new()
- MeasureSurvCalibrationBeta$clone()

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

Usage:

```
MeasureSurvCalibrationBeta$new(method = "ratio")
```

Arguments:

`method` defines which output score to return, see "Parameter details" section.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Van Houwelingen, C. H (2000). "Validation, calibration, revision and combination of prognostic survival models." *Statistics in Medicine*, **19**(24), 3401–3415. doi:10.1002/10970258(20001230)19:24<3401::AID-SIM554>3.0.CO;22.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_t` `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other calibration survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.dcalib`

Other lp survival measures: `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

`mlr_measures_surv.chambless_auc`

Chambless and Diao's AUC Survival Measure

Description

Calls `survAUC::AUC.cd()`.

Assumes Cox PH model specification.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in `mlr3proba`.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvChamblessAUC$new()
mlr_measures$get("surv.chambless_auc")
msr("surv.chambless_auc")
```

Parameters

Id	Type	Default	Levels
integrated	logical	TRUE	TRUE, FALSE
times	untyped	-	

Meta Information

- Type: "surv"
- Range: $[0, 1]$
- Minimize: FALSE
- Required prediction: lp

Parameter details

- integrated (logical(1))
  If TRUE (default), returns the integrated score (eg across time points); otherwise, not integrated (eg at a single time point).

- times (numeric())
  If integrated == TRUE then a vector of time-points over which to integrate the score. If integrated == FALSE then a single time point at which to return the score.

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> mlr3proba::MeasureSurvAUC -> MeasureSurvChamblessAUC

Methods

Public methods:

- MeasureSurvChamblessAUC$new()
- MeasureSurvChamblessAUC$clone()

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

Usage:

```
MeasureSurvChamblessAUC$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Chambless LE, Diao G (2006). "Estimation of time-dependent area under the ROC curve for long-term risk prediction." *Statistics in Medicine*, **25**(20), 3474–3486. doi:10.1002/sim.2299.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tr` `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other AUC survival measures: `mlr_measures_surv.hung_auc`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`

Other lp survival measures: `mlr_measures_surv.calib_beta`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

`mlr_measures_surv.cindex` *Concordance Statistics Survival Measure*

Description

Calculates weighted concordance statistics, which, depending on the chosen weighting method (`weight_meth`) and tied times parameter (`tiex`), are equivalent to several proposed methods. By default, no weighting is applied and this is equivalent to Harrell's C-index.

Details

For the Kaplan-Meier estimate of the **training survival** distribution (S), and the Kaplan-Meier estimate of the **training censoring** distribution (G), we have the following options for time-independent concordance statistics (C-indexes) given the weighted method:

`weight_meth`:

- `"I"` = No weighting. (Harrell)

- "GH" = Gonen and Heller's Concordance Index
- "G" = Weights concordance by $1/G$.
- "G2" = Weights concordance by $1/G^2$. (Uno et al.)
- "SG" = Weights concordance by S/G (Shemper et al.)
- "S" = Weights concordance by S (Peto and Peto)

The last three require training data. "GH" is only applicable to LearnerSurvCoxPH.

The implementation is slightly different from survival::concordance. Firstly this implementation is faster, and secondly the weights are computed on the training dataset whereas in survival::concordance the weights are computed on the same testing data.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvCindex$new()
mlr_measures$get("surv.cindex")
msr("surv.cindex")
```

Parameters

Id	Type	Default	Levels	Range
t_max	numeric	-		$[0, \infty)$
p_max	numeric	-		$[0, 1]$
weight_meth	character	I	I, G, G2, SG, S, GH	-
tiex	numeric	0.5		$[0, 1]$
eps	numeric	0.001		$[0, 1]$

Meta Information

- Type: "surv"
- Range: $[0, 1]$
- Minimize: FALSE
- Required prediction: crank

Parameter details

- eps (numeric(1))
  Very small number to substitute zero values in order to prevent errors in e.g. log(0) and/or division-by-zero calculations. Default value is 0.001.

- t_max (numeric(1))
  Cutoff time (i.e. time horizon) to evaluate concordance up to.

- `p_max` (`numeric(1)`)
  The proportion of censoring to evaluate concordance up to in the given dataset. When `t_max` is specified, this parameter is ignored.
- `weight_meth` (`character(1)`)
  Method for weighting concordance. Default `"I"` is Harrell's C. See details.
- `tiex` (`numeric(1)`)
  Weighting applied to tied rankings, default is to give them half (0.5) weighting.

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvCindex`

Methods

Public methods:

- `MeasureSurvCindex$new()`
- `MeasureSurvCindex$clone()`

Method `new()`**:** This is an abstract class that should not be constructed directly.

Usage:

```
MeasureSurvCindex$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Peto, Richard, Peto, Julian (1972). “Asymptotically efficient rank invariant test procedures.” *Journal of the Royal Statistical Society: Series A (General)*, **135**(2), 185–198.

Harrell, E F, Califf, M R, Pryor, B D, Lee, L K, Rosati, A R (1982). “Evaluating the yield of medical tests.” *Jama*, **247**(18), 2543–2546.

Goenen M, Heller G (2005). “Concordance probability and discriminatory power in proportional hazards regression.” *Biometrika*, **92**(4), 965–970. doi:10.1093/biomet/92.4.965.

Schemper, Michael, Wakounig, Samo, Heinze, Georg (2009). “The estimation of average hazard ratios by weighted Cox regression.” *Statistics in Medicine*, **28**(19), 2473–2489. doi:10.1002/sim.3623.

Uno H, Cai T, Pencina MJ, D'Agostino RB, Wei LJ (2011). “On the C-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data.” *Statistics in Medicine*, n/a–n/a. doi:10.1002/sim.4154.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.dcalib, mlr_measures_surv.graf, mlr_measures_surv.hung_auc, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mae, mlr_measures_surv.mse, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_ mlr_measures_surv.rcll, mlr_measures_surv.rmse, mlr_measures_surv.schmid, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Examples

```
library(mlr3)
task = tsk("rats")
learner = lrn("surv.coxph")
part = partition(task) # train/test split
learner$train(task, part$train)
p = learner$predict(task, part$test)

# Harrell's C-index
p$score(msr("surv.cindex")) # same as `p$score()`

# Uno's C-index
p$score(msr("surv.cindex", weight_meth = "G2"),
        task = task, train_set = part$train)

# Harrell's C-index evaluated up to a specific time horizon
p$score(msr("surv.cindex", t_max = 97))
# Harrell's C-index evaluated up to the time corresponding to 30% of censoring
p$score(msr("surv.cindex", p_max = 0.3))
```

---

mlr_measures_surv.dcalib

D-Calibration Survival Measure

Description

This calibration method is defined by calculating the following statistic:

$$s = B/n \sum_i (P_i - n/B)^2$$

where B is number of 'buckets' (that equally divide $[0,1]$ into intervals), n is the number of predictions, and P_i is the observed proportion of observations in the ith interval. An observation is assigned to the ith bucket, if its predicted survival probability at the time of event falls within the corresponding interval. This statistic assumes that censoring time is independent of death time.

A model is well-calibrated if $s \sim Unif(B)$, tested with chisq.test ($p > 0.05$ if well-calibrated). Model i is better calibrated than model j if $s(i) < s(j)$, meaning that *lower values* of this measure are preferred.

Details

This measure can either return the test statistic or the p-value from the chisq.test. The former is useful for model comparison whereas the latter is useful for determining if a model is well-calibrated. If chisq = FALSE and s is the predicted value then you can manually compute the p.value with pchisq(s, B - 1, lower.tail = FALSE).

NOTE: This measure is still experimental both theoretically and in implementation. Results should therefore only be taken as an indicator of performance and not for conclusive judgements about model calibration.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvDCalibration$new()
mlr_measures$get("surv.dcalib")
msr("surv.dcalib")
```

Parameters

Id	Type	Default	Levels	Range
B	integer	10		$[1, \infty)$
chisq	logical	FALSE	TRUE, FALSE	-
truncate	numeric	Inf		$[0, \infty)$

Meta Information

- Type: "surv"
- Range: $[0, \infty)$
- Minimize: TRUE
- Required prediction: distr

Parameter details

- B (integer(1))
  Number of buckets to test for uniform predictions over. Default of 10 is recommended by Haider et al. (2020). Changing this parameter affects truncate.
- chisq (logical(1))
  If TRUE returns the p-value of the corresponding chisq.test instead of the measure. Default is FALSE and returns the statistic s. You can manually get the p-value by executing pchisq(s, B - 1, lower.tail = FALSE). The null hypothesis is that the model is D-calibrated.

- truncate (double(1))
  This parameter controls the upper bound of the output statistic, when chisq is FALSE. We use truncate = Inf by default but 10 may be sufficient for most purposes, which corresponds to a p-value of 0.35 for the chisq.test using $B = 10$ buckets. Values > 10 translate to even lower p-values and thus less calibrated models. If the number of buckets B changes, you probably will want to change the truncate value as well to correspond to the same p-value significance. Note that truncation may severely limit automated tuning with this measure.

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> MeasureSurvDCalibration

Methods

Public methods:

- MeasureSurvDCalibration$new()
- MeasureSurvDCalibration$clone()

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

Usage:

```
MeasureSurvDCalibration$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Haider, Humza, Hoehn, Bret, Davis, Sarah, Greiner, Russell (2020). “Effective Ways to Build and Evaluate Individual Survival Distributions.” *Journal of Machine Learning Research*, **21**(85), 1–63. https://jmlr.org/papers/v21/18-772.html.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.cindex, mlr_measures_surv.graf, mlr_measures_surv.hung_auc, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mae, mlr_measures_surv.mse, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_r mlr_measures_surv.rcll, mlr_measures_surv.rmse, mlr_measures_surv.schmid, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Other calibration survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta

Other distr survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.graf, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.rcll, mlr_measures_surv.schr `mlr_measures_surv.graf`

Integrated Brier Score Survival Measure

Description

Calculates the **Integrated Survival Brier Score** (ISBS), Integrated Graf Score or squared survival loss.

Details

This measure has two dimensions: (test set) observations and time points. For a specific individual i from the test set, with observed survival outcome (t_i, δ_i) (time and censoring indicator) and predicted survival function $S_i(t)$, the *observation-wise* loss integrated across the time dimension up to the time cutoff τ^*, is:

$$L_{ISBS}(S_i, t_i, \delta_i) = \text{I}(t_i \leq \tau^*) \int_0^{\tau^*} \frac{S_i^2(\tau)\text{I}(t_i \leq \tau, \delta = 1)}{G(t_i)} + \frac{(1 - S_i(\tau))^2 \text{I}(t_i > \tau)}{G(\tau)} \, d\tau$$

where G is the Kaplan-Meier estimate of the censoring distribution.

The **re-weighted ISBS** (RISBS) is:

$$L_{RISBS}(S_i, t_i, \delta_i) = \delta_i \text{I}(t_i \leq \tau^*) \int_0^{\tau^*} \frac{S_i^2(\tau)\text{I}(t_i \leq \tau) + (1 - S_i(\tau))^2 \text{I}(t_i > \tau)}{G(t_i)} \, d\tau$$

which is always weighted by $G(t_i)$ and is equal to zero for a censored subject.

To get a single score across all N observations of the test set, we return the average of the time-integrated observation-wise scores:

$$\sum_{i=1}^{N} L(S_i, t_i, \delta_i)/N$$

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvGraf$new()
mlr_measures$get("surv.graf")
msr("surv.graf")
```

Parameters

Id	Type	Default	Levels	Range
integrated	logical	TRUE	TRUE, FALSE	-
times	untyped	-		-
t_max	numeric	-		$[0, \infty)$
p_max	numeric	-		$[0, 1]$
method	integer	2		$[1, 2]$
se	logical	FALSE	TRUE, FALSE	-
proper	logical	FALSE	TRUE, FALSE	-
eps	numeric	0.001		$[0, 1]$
ERV	logical	FALSE	TRUE, FALSE	-

Meta Information

- Type: "surv"
- Range: $[0, \infty)$
- Minimize: TRUE
- Required prediction: distr

Parameter details

- integrated (logical(1))
  If TRUE (default), returns the integrated score (eg across time points); otherwise, not integrated (eg at a single time point).

- times (numeric())
  If integrated == TRUE then a vector of time-points over which to integrate the score. If integrated == FALSE then a single time point at which to return the score.

- t_max (numeric(1))
  Cutoff time τ^* (i.e. time horizon) to evaluate the measure up to. Mutually exclusive with p_max or times. This will effectively remove test observations for which the observed time (event or censoring) is strictly more than t_max. It's recommended to set t_max to avoid division by eps, see Details. If t_max is not specified, an Inf time horizon is assumed.

- p_max (numeric(1))
  The proportion of censoring to integrate up to in the given dataset. Mutually exclusive with times or t_max.

- method (integer(1))
  If integrate == TRUE, this selects the integration weighting method. method == 1 corresponds to weighting each time-point equally and taking the mean score over discrete time-points. method == 2 corresponds to calculating a mean weighted by the difference between time-points. method == 2 is the default value, to be in line with other packages.

- se (logical(1))
  If TRUE then returns standard error of the measure otherwise returns the mean across all individual scores, e.g. the mean of the per observation scores. Default is FALSE (returns the mean).

- proper (logical(1))
  If TRUE then weights scores by the censoring distribution at the observed event time, which results in a strictly proper scoring rule if censoring and survival time distributions are independent and a sufficiently large dataset is used. If FALSE then weights scores by the Graf method which is the more common usage but the loss is not proper.

- eps (numeric(1))
  Very small number to substitute zero values in order to prevent errors in e.g. log(0) and/or division-by-zero calculations. Default value is 0.001.

- ERV (logical(1))
  If TRUE then the Explained Residual Variation method is applied, which means the score is standardized against a Kaplan-Meier baseline. Default is FALSE.

Properness

RISBS is strictly proper when the censoring distribution is independent of the survival distribution and when $G(t)$ is fit on a sufficiently large dataset. ISBS is never proper. Use proper = FALSE for ISBS and proper = TRUE for RISBS. Results may be very different if many observations are censored at the last observed time due to division by $1/eps$ in proper = TRUE.

Time points used for evaluation

If the times argument is not specified (NULL), then the unique (and sorted) time points from the **test set** are used for evaluation of the time-integrated score. This was a design decision due to the fact that different predicted survival distributions $S(t)$ usually have a **discretized time domain** which may differ, i.e. in the case the survival predictions come from different survival learners. Essentially, using the same set of time points for the calculation of the score minimizes the bias that would come from using different time points. We note that $S(t)$ is by default constantly interpolated for time points that fall outside its discretized time domain.

Naturally, if the times argument is specified, then exactly these time points are used for evaluation. A warning is given to the user in case some of the specified times fall outside of the time point range of the test set. The assumption here is that if the test set is large enough, it should have a time domain/range similar to the one from the train set, and therefore time points outside that domain might lead to interpolation or extrapolation of $S(t)$.

Implementation differences

If comparing the integrated graf score to other packages, e.g. **pec**, then method = 2 should be used. However the results may still be very slightly different as this package uses survfit to estimate the censoring distribution, in line with the Graf 1999 paper; whereas some other packages use prodlim with reverse = TRUE (meaning Kaplan-Meier is not used).

Data used for Estimating Censoring Distribution

If `task` and `train_set` are passed to `$score` then $G(t)$ is fit on training data, otherwise testing data. The first is likely to reduce any bias caused by calculating parts of the measure on the test data it is evaluating. The training data is automatically used in scoring resamplings.

Time Cutoff Details

If `t_max` or `p_max` is given, then $G(t)$ will be fitted using **all observations** from the train set (or test set) and only then the cutoff time will be applied. This is to ensure that more data is used for fitting the censoring distribution via the Kaplan-Meier. Setting the `t_max` can help alleviate inflation of the score when `proper` is `TRUE`, in cases where an observation is censored at the last observed time point. This results in $G(t_{max}) = 0$ and the use of `eps` instead (when `t_max` is `NULL`).

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvGraf`

Methods

Public methods:

- `MeasureSurvGraf$new()`
- `MeasureSurvGraf$clone()`

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

Usage:

```
MeasureSurvGraf$new(ERV = FALSE)
```

Arguments:

`ERV (logical(1))`
Standardize measure against a Kaplan-Meier baseline (Explained Residual Variation)

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Graf E, Schmoor C, Sauerbrei W, Schumacher M (1999). “Assessment and comparison of prognostic classification schemes for survival data.” *Statistics in Medicine*, **18**(17-18), 2529–2545. doi:10.1002/(sici)10970258(19990915/30)18:17/18<2529::aidsim274>3.0.co;25.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.cindex, mlr_measures_surv.dcalib, mlr_measures_surv.hung_auc, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mae, mlr_measures_surv.mse, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_r mlr_measures_surv.rcll, mlr_measures_surv.rmse, mlr_measures_surv.schmid, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Other Probabilistic survival measures: mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.rcll, mlr_measures_surv.schmid

Other distr survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.dcalib, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.rcll, mlr_measures_surv.schn

---

mlr_measures_surv.hung_auc

Hung and Chiang's AUC Survival Measure

Description

Calls `survAUC::AUC.hc()`.

Assumes random censoring.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in `mlr3proba`.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvHungAUC$new()
mlr_measures$get("surv.hung_auc")
msr("surv.hung_auc")
```

Parameters

Id	Type	Default	Levels
integrated	logical	TRUE	TRUE, FALSE
times	untyped	-	

Meta Information

- Type: "surv"
- Range: $[0, 1]$
- Minimize: FALSE
- Required prediction: lp

Parameter details

- integrated (logical(1))
  If TRUE (default), returns the integrated score (eg across time points); otherwise, not integrated (eg at a single time point).

- times (numeric())
  If integrated == TRUE then a vector of time-points over which to integrate the score. If integrated == FALSE then a single time point at which to return the score.

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> mlr3proba::MeasureSurvAUC -> MeasureSurvHungAUC

Methods

Public methods:

- MeasureSurvHungAUC$new()
- MeasureSurvHungAUC$clone()

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

Usage:

```
MeasureSurvHungAUC$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Hung H, Chiang C (2010). “Estimation methods for time-dependent AUC models with survival data.” *The Canadian Journal of Statistics / La Revue Canadienne de Statistique*, **38**(1), 8–26. https://www.jstor.org/stable/27805213.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.cindex, mlr_measures_surv.dcalib, mlr_measures_surv.graf, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mae, mlr_measures_surv.mse, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_r2, mlr_measures_surv.rcll, mlr_measures_surv.rmse, mlr_measures_surv.schmid, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Other AUC survival measures: mlr_measures_surv.chambless_auc, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr

Other lp survival measures: mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_r2, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

mlr_measures_surv.intlogloss *Integrated Log-Likelihood Survival Measure*

Description

Calculates the **Integrated Survival Log-Likelihood** (ISLL) or Integrated Logarithmic (log) Loss, aka integrated cross entropy.

Details

This measure has two dimensions: (test set) observations and time points. For a specific individual i from the test set, with observed survival outcome (t_i, δ_i) (time and censoring indicator) and predicted survival function $S_i(t)$, the *observation-wise* loss integrated across the time dimension up to the time cutoff τ^*, is:

$$L_{ISLL}(S_i, t_i, \delta_i) = -\text{I}(t_i \leq \tau^*) \int^{\tau^*}_0 \frac{log[1 - S_i(\tau)]\text{I}(t_i \leq \tau, \delta = 1)}{G(t_i)} + \frac{\log[S_i(\tau)]\text{I}(t_i > \tau)}{G(\tau)} \, d\tau$$

where G is the Kaplan-Meier estimate of the censoring distribution.

The **re-weighted ISLL** (RISLL) is:

$$L_{RISLL}(S_i, t_i, \delta_i) = -\delta_i \text{I}(t_i \leq \tau^*) \int^{\tau^*}_0 \frac{\log[1 - S_i(\tau)])\text{I}(t_i \leq \tau) + \log[S_i(\tau)]\text{I}(t_i > \tau)}{G(t_i)} \, d\tau$$

which is always weighted by $G(t_i)$ and is equal to zero for a censored subject.

To get a single score across all N observations of the test set, we return the average of the time-integrated observation-wise scores:

$$\sum_{i=1}^{N} L(S_i, t_i, \delta_i)/N$$

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvIntLogloss$new()
mlr_measures$get("surv.intlogloss")
msr("surv.intlogloss")
```

Parameters

Id	Type	Default	Levels	Range
integrated	logical	TRUE	TRUE, FALSE	-
times	untyped	-		-
t_max	numeric	-		$[0, \infty)$
p_max	numeric	-		$[0, 1]$
method	integer	2		$[1, 2]$
se	logical	FALSE	TRUE, FALSE	-
proper	logical	FALSE	TRUE, FALSE	-
eps	numeric	0.001		$[0, 1]$
ERV	logical	FALSE	TRUE, FALSE	-

Meta Information

- Type: "surv"
- Range: $[0, \infty)$
- Minimize: TRUE
- Required prediction: distr

Parameter details

- integrated (logical(1))
  If TRUE (default), returns the integrated score (eg across time points); otherwise, not integrated (eg at a single time point).

- times (numeric())
  If integrated == TRUE then a vector of time-points over which to integrate the score. If integrated == FALSE then a single time point at which to return the score.

- t_max (numeric(1))
  Cutoff time τ^* (i.e. time horizon) to evaluate the measure up to. Mutually exclusive with p_max or times. This will effectively remove test observations for which the observed time (event or censoring) is strictly more than t_max. It's recommended to set t_max to avoid division by eps, see Details. If t_max is not specified, an Inf time horizon is assumed.

- p_max (numeric(1))
  The proportion of censoring to integrate up to in the given dataset. Mutually exclusive with times or t_max.

- method (integer(1))
  If integrate == TRUE, this selects the integration weighting method. method == 1 corresponds to weighting each time-point equally and taking the mean score over discrete time-points. method == 2 corresponds to calculating a mean weighted by the difference between time-points. method == 2 is the default value, to be in line with other packages.

- se (logical(1))
  If TRUE then returns standard error of the measure otherwise returns the mean across all individual scores, e.g. the mean of the per observation scores. Default is FALSE (returns the mean).

- proper (logical(1))
  If TRUE then weights scores by the censoring distribution at the observed event time, which results in a strictly proper scoring rule if censoring and survival time distributions are independent and a sufficiently large dataset is used. If FALSE then weights scores by the Graf method which is the more common usage but the loss is not proper.

- eps (numeric(1))
  Very small number to substitute zero values in order to prevent errors in e.g. log(0) and/or division-by-zero calculations. Default value is 0.001.

- ERV (logical(1))
  If TRUE then the Explained Residual Variation method is applied, which means the score is standardized against a Kaplan-Meier baseline. Default is FALSE.

Properness

RISLL is strictly proper when the censoring distribution is independent of the survival distribution and when $G(t)$ is fit on a sufficiently large dataset. ISLL is never proper. Use proper = FALSE for ISLL and proper = TRUE for RISLL. Results may be very different if many observations are censored at the last observed time due to division by $1/eps$ in proper = TRUE.

Time points used for evaluation

If the times argument is not specified (NULL), then the unique (and sorted) time points from the **test set** are used for evaluation of the time-integrated score. This was a design decision due to the fact that different predicted survival distributions $S(t)$ usually have a **discretized time domain** which may differ, i.e. in the case the survival predictions come from different survival learners. Essentially, using the same set of time points for the calculation of the score minimizes the bias that would come from using different time points. We note that $S(t)$ is by default constantly interpolated for time points that fall outside its discretized time domain.

Naturally, if the `times` argument is specified, then exactly these time points are used for evaluation. A warning is given to the user in case some of the specified `times` fall outside of the time point range of the test set. The assumption here is that if the test set is large enough, it should have a time domain/range similar to the one from the train set, and therefore time points outside that domain might lead to interpolation or extrapolation of $S(t)$.

Implementation differences

If comparing the integrated graf score to other packages, e.g. **pec**, then `method = 2` should be used. However the results may still be very slightly different as this package uses `survfit` to estimate the censoring distribution, in line with the Graf 1999 paper; whereas some other packages use `prodlim` with `reverse = TRUE` (meaning Kaplan-Meier is not used).

Data used for Estimating Censoring Distribution

If `task` and `train_set` are passed to `$score` then $G(t)$ is fit on training data, otherwise testing data. The first is likely to reduce any bias caused by calculating parts of the measure on the test data it is evaluating. The training data is automatically used in scoring resamplings.

Time Cutoff Details

If `t_max` or `p_max` is given, then $G(t)$ will be fitted using **all observations** from the train set (or test set) and only then the cutoff time will be applied. This is to ensure that more data is used for fitting the censoring distribution via the Kaplan-Meier. Setting the `t_max` can help alleviate inflation of the score when `proper` is `TRUE`, in cases where an observation is censored at the last observed time point. This results in $G(t_{max}) = 0$ and the use of `eps` instead (when `t_max` is `NULL`).

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvIntLogloss`

Methods

Public methods:

- `MeasureSurvIntLogloss$new()`
- `MeasureSurvIntLogloss$clone()`

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

Usage:

```
MeasureSurvIntLogloss$new(ERV = FALSE)
```

Arguments:

`ERV (logical(1))`
Standardize measure against a Kaplan-Meier baseline (Explained Residual Variation)

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Graf E, Schmoor C, Sauerbrei W, Schumacher M (1999). "Assessment and comparison of prognostic classification schemes for survival data." *Statistics in Medicine*, **18**(17-18), 2529–2545. doi:10.1002/(sici)10970258(19990915/30)18:17/18<2529::aidsim274>3.0.co;25.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other Probabilistic survival measures: `mlr_measures_surv.graf`, `mlr_measures_surv.logloss`, `mlr_measures_surv.rcll`, `mlr_measures_surv.schmid`

Other distr survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.logloss`, `mlr_measures_surv.rcll`, `mlr_measures_surv.schmid`

`mlr_measures_surv.logloss`

Negative Log-Likelihood Survival Measure

Description

Calculates the cross-entropy, or negative log-likelihood (NLL) or logarithmic (log), loss.

Details

The Log Loss, in the context of probabilistic predictions, is defined as the negative log probability density function, f, evaluated at the observation time (event or censoring), t,

$$L_{NLL}(f, t) = -\log[f(t)]$$

The standard error of the Log Loss, L, is approximated via,

$$se(L) = sd(L)/\sqrt{N}$$

where N are the number of observations in the test set, and sd is the standard deviation.

The **Re-weighted Negative Log-Likelihood** (RNLL) or IPCW (Inverse Probability Censoring Weighted) Log Loss is defined by

$$L_{RNLL}(f, t, \delta) = -\frac{\delta \log[f(t)]}{G(t)}$$

where δ is the censoring indicator and $G(t)$ is the Kaplan-Meier estimator of the censoring distribution. So only observations that have experienced the event are taking into account for RNLL (i.e. $\delta = 1$) and both $f(t), G(t)$ are calculated only at the event times. If only censored observations exist in the test set, `NaN` is returned.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvLogloss$new()
mlr_measures$get("surv.logloss")
msr("surv.logloss")
```

Parameters

Id	Type	Default	Levels	Range
eps	numeric	1e-15		$[0, 1]$
se	logical	FALSE	TRUE, FALSE	-
IPCW	logical	TRUE	TRUE, FALSE	-
ERV	logical	FALSE	TRUE, FALSE	-

Meta Information

- Type: `"surv"`
- Range: $[0, \infty)$
- Minimize: `TRUE`
- Required prediction: `distr`

Parameter details

- `eps` (`numeric(1)`)
  Very small number to substitute zero values in order to prevent errors in e.g. log(0) and/or division-by-zero calculations. Default value is 1e-15.

- `se` (`logical(1)`)
  If `TRUE` then returns standard error of the measure otherwise returns the mean across all individual scores, e.g. the mean of the per observation scores. Default is `FALSE` (returns the mean).

- `ERV` (`logical(1)`)
  If `TRUE` then the Explained Residual Variation method is applied, which means the score is standardized against a Kaplan-Meier baseline. Default is `FALSE`.

- `IPCW` (`logical(1)`)
  If `TRUE` (default) then returns the L_{RNLL} score (which is proper), otherwise the L_{NLL} score (improper).

Data used for Estimating Censoring Distribution

If `task` and `train_set` are passed to `$score` then $G(t)$ is fit on training data, otherwise testing data. The first is likely to reduce any bias caused by calculating parts of the measure on the test data it is evaluating. The training data is automatically used in scoring resamplings.

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvLogloss`

Methods

Public methods:

- `MeasureSurvLogloss$new()`
- `MeasureSurvLogloss$clone()`

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

Usage:

```
MeasureSurvLogloss$new(ERV = FALSE)
```

Arguments:

`ERV` (`logical(1)`)
Standardize measure against a Kaplan-Meier baseline (Explained Residual Variation)

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r` `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other Probabilistic survival measures: `mlr_measures_surv.graf`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.rcll`, `mlr_measures_surv.schmid`

Other distr survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.rcll`, `mlr_measures_surv.schmid`

`mlr_measures_surv.mae` *Mean Absolute Error Survival Measure*

Description

Calculates the mean absolute error (MAE).

The MAE is defined by

$$\frac{1}{n}\sum|t - \hat{t}|$$

where t is the true value and $\hat{t}$ is the prediction.

Censored observations in the test set are ignored.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvMAE$new()
mlr_measures$get("surv.mae")
msr("surv.mae")
```

Parameters

Id	Type	Default	Levels
se	logical	FALSE	TRUE, FALSE

Meta Information

- Type: `"surv"`
- Range: $[0, \infty)$
- Minimize: `TRUE`
- Required prediction: `response`

Parameter details

- se (logical(1))
  If TRUE then returns standard error of the measure otherwise returns the mean across all individual scores, e.g. the mean of the per observation scores. Default is FALSE (returns the mean).

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> MeasureSurvMAE

Methods

Public methods:

- MeasureSurvMAE$new()
- MeasureSurvMAE$clone()

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

Usage:

```
MeasureSurvMAE$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.cindex, mlr_measures_surv.dcalib, mlr_measures_surv.graf, mlr_measures_surv.hung_auc, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mse, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquig mlr_measures_surv.rcll, mlr_measures_surv.rmse, mlr_measures_surv.schmid, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Other response survival measures: mlr_measures_surv.mse, mlr_measures_surv.rmse

mlr_measures_surv.mse *Mean Squared Error Survival Measure*

Description

Calculates the mean squared error (MSE).

The MSE is defined by

$$\frac{1}{n} \sum ((t - \hat{t})^2)$$

where t is the true value and $\hat{t}$ is the prediction.

Censored observations in the test set are ignored.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvMSE$new()
mlr_measures$get("surv.mse")
msr("surv.mse")
```

Parameters

Id	Type	Default	Levels
se	logical	FALSE	TRUE, FALSE

Meta Information

- Type: "surv"
- Range: $[0, \infty)$
- Minimize: TRUE
- Required prediction: response

Parameter details

- se (logical(1))
  If TRUE then returns standard error of the measure otherwise returns the mean across all individual scores, e.g. the mean of the per observation scores. Default is FALSE (returns the mean).

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> MeasureSurvMSE

Methods

Public methods:

- MeasureSurvMSE$new()
- MeasureSurvMSE$clone()

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

Usage:

```
MeasureSurvMSE$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquig` `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other response survival measures: `mlr_measures_surv.mae`, `mlr_measures_surv.rmse`

`mlr_measures_surv.nagelk_r2` *Nagelkerke's R2 Survival Measure*

Description

Calls `survAUC::Nagelk()`.

Assumes Cox PH model specification.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in `mlr3proba`.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvNagelkR2$new()
mlr_measures$get("surv.nagelk_r2")
msr("surv.nagelk_r2")
```

Parameters

Empty ParamSet

Meta Information

- Type: `"surv"`
- Range: $[0, 1]$
- Minimize: `FALSE`
- Required prediction: `lp`

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvNagelkR2`

Methods

Public methods:

- `MeasureSurvNagelkR2$new()`
- `MeasureSurvNagelkR2$clone()`

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

Usage:

```
MeasureSurvNagelkR2$new()
```

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

Usage:

```
MeasureSurvNagelkR2$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Nagelkerke, JD N, others (1991). “A note on a general definition of the coefficient of determination.” *Biometrika*, **78**(3), 691–692.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.cindex, mlr_measures_surv.dcalib, mlr_measures_surv.graf, mlr_measures_surv.hung_auc, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mae, mlr_measures_surv.mse, mlr_measures_surv.oquigley_r2, mlr_measures_surv.rcll, mlr_measures_surv.rmse, mlr_measures_surv.schmid, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Other R2 survival measures: mlr_measures_surv.oquigley_r2, mlr_measures_surv.xu_r2

Other lp survival measures: mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.hung_auc, mlr_measures_surv.oquigley_r2, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

---

mlr_measures_surv.oquigley_r2

O'Quigley, Xu, and Stare's R2 Survival Measure

Description

Calls survAUC::OXS().

Assumes Cox PH model specification.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in mlr3proba.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvOQuigleyR2$new()
mlr_measures$get("surv.oquigley_r2")
msr("surv.oquigley_r2")
```

Parameters

Empty ParamSet

Meta Information

- Type: "surv"
- Range: $[0, 1]$
- Minimize: FALSE
- Required prediction: lp

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> MeasureSurvOQuigleyR2

Methods

Public methods:

- MeasureSurvOQuigleyR2$new()
- MeasureSurvOQuigleyR2$clone()

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

Usage:

```
MeasureSurvOQuigleyR2$new()
```

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

Usage:

```
MeasureSurvOQuigleyR2$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

O'Quigley J, Xu R, Stare J (2005). "Explained randomness in proportional hazards models." *Statistics in Medicine*, **24**(3), 479–489. doi:10.1002/sim.1946.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.cindex, mlr_measures_surv.dcalib, mlr_measures_surv.graf, mlr_measures_surv.hung_auc, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mae, mlr_measures_surv.mse, mlr_measures_surv.nagelk_r2, mlr_measures_surv.rcll, mlr_measures_surv.rmse, mlr_measures_surv.schmid, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Other R2 survival measures: mlr_measures_surv.nagelk_r2, mlr_measures_surv.xu_r2

Other lp survival measures: mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.hung_auc, mlr_measures_surv.nagelk_r2, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

mlr_measures_surv.rcll *Right-Censored Log Loss Survival Measure*

Description

Calculates the right-censored logarithmic (log), loss.

Details

The RCLL, in the context of probabilistic predictions, is defined by

$$L(f, t, \Delta) = -log(\Delta f(t) + (1 - \Delta)S(t))$$

where Δ is the censoring indicator, f the probability density function and S the survival function. RCLL is proper given that censoring and survival distribution are independent, see Rindt et al. (2022).

Note: Even though RCLL is a proper scoring rule, the calculation of $f(t)$ (which in our case is discrete, i.e. it is a *probability mass function*) for time points in the test set that don't exist in the predicted survival matrix (`distr`), results in 0 values, which are substituted by `"eps"` in our implementation, therefore skewing the result towards $-log(eps)$. This problem is also discussed in Rindt et al. (2022), where the authors perform interpolation to get non-zero values for the $f(t)$. Until this is handled in `mlr3proba` some way, we advise against using this measure for model evaluation.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvRCLL$new()
mlr_measures$get("surv.rcll")
msr("surv.rcll")
```

Parameters

Id	Type	Default	Levels	Range
eps	numeric	1e-15		$[0, 1]$
se	logical	FALSE	TRUE, FALSE	-
ERV	logical	FALSE	TRUE, FALSE	-
na.rm	logical	TRUE	TRUE, FALSE	-

Meta Information

- Type: "surv"
- Range: $[0, \infty)$
- Minimize: TRUE
- Required prediction: distr

Parameter details

- eps (numeric(1))
  Very small number to substitute zero values in order to prevent errors in e.g. log(0) and/or division-by-zero calculations. Default value is 1e-15.

- se (logical(1))
  If TRUE then returns standard error of the measure otherwise returns the mean across all individual scores, e.g. the mean of the per observation scores. Default is FALSE (returns the mean).

- ERV (logical(1))
  If TRUE then the Explained Residual Variation method is applied, which means the score is standardized against a Kaplan-Meier baseline. Default is FALSE.

- na.rm (logical(1))
  If TRUE (default) then removes any NAs in individual score calculations.

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> MeasureSurvRCLL

Methods

Public methods:

- MeasureSurvRCLL$new()
- MeasureSurvRCLL$clone()

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

Usage:

```
MeasureSurvRCLL$new(ERV = FALSE)
```

Arguments:

ERV (logical(1))
Standardize measure against a Kaplan-Meier baseline (Explained Residual Variation)

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Avati, Anand, Duan, Tony, Zhou, Sharon, Jung, Kenneth, Shah, H N, Ng, Y A (2020). “Countdown Regression: Sharp and Calibrated Survival Predictions.” *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, **115**(4), 145–155. `https://proceedings.mlr.press/v115/avati20a.html`.

Rindt, David, Hu, Robert, Steinsaltz, David, Sejdinovic, Dino (2022). “Survival regression with proper scoring rules and monotonic neural networks.” *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, **151**(4), 1190–1205. `https://proceedings.mlr.press/v151/rindt22a.html`.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song`
`mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other Probabilistic survival measures: `mlr_measures_surv.graf`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.schmid`

Other distr survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.schm`

`mlr_measures_surv.rmse` *Root Mean Squared Error Survival Measure*

Description

Calculates the root mean squared error (RMSE).

The RMSE is defined by

$$\sqrt{\frac{1}{n}\sum((t - \hat{t})^2)}$$

where t is the true value and $\hat{t}$ is the prediction.

Censored observations in the test set are ignored.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvRMSE$new()
mlr_measures$get("surv.rmse")
msr("surv.rmse")
```

Parameters

Id	Type	Default	Levels
se	logical	FALSE	TRUE, FALSE

Meta Information

- Type: `"surv"`
- Range: $[0, \infty)$
- Minimize: `TRUE`
- Required prediction: `response`

Parameter details

- `se` (`logical(1)`)
  If `TRUE` then returns standard error of the measure otherwise returns the mean across all individual scores, e.g. the mean of the per observation scores. Default is `FALSE` (returns the mean).

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvRMSE`

Methods

Public methods:

- `MeasureSurvRMSE$new()`
- `MeasureSurvRMSE$clone()`

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

Usage:

```
MeasureSurvRMSE$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.cindex, mlr_measures_surv.dcalib, mlr_measures_surv.graf, mlr_measures_surv.hung_auc, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mae, mlr_measures_surv.mse, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_r2, mlr_measures_surv.rcll, mlr_measures_surv.schmid, mlr_measures_surv.song mlr_measures_surv.song_tnr, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Other response survival measures: mlr_measures_surv.mae, mlr_measures_surv.mse

mlr_measures_surv.schmid *Integrated Schmid Score Survival Measure*

Description

Calculates the **Integrated Schmid Score** (ISS), aka integrated absolute loss.

Details

This measure has two dimensions: (test set) observations and time points. For a specific individual i from the test set, with observed survival outcome (t_i, δ_i) (time and censoring indicator) and predicted survival function $S_i(t)$, the *observation-wise* loss integrated across the time dimension up to the time cutoff τ^*, is:

$$L_{ISS}(S_i, t_i, \delta_i) = \text{I}(t_i \leq \tau^*) \int_0^{\tau^*} \frac{S_i(\tau)\text{I}(t_i \leq \tau, \delta = 1)}{G(t_i)} + \frac{(1 - S_i(\tau))\text{I}(t_i > \tau)}{G(\tau)} \, d\tau$$

where G is the Kaplan-Meier estimate of the censoring distribution.

The **re-weighted ISS** (RISS) is:

$$L_{RISS}(S_i, t_i, \delta_i) = \delta_i \text{I}(t_i \leq \tau^*) \int_0^{\tau^*} \frac{S_i(\tau)\text{I}(t_i \leq \tau) + (1 - S_i(\tau))\text{I}(t_i > \tau)}{G(t_i)} \, d\tau$$

which is always weighted by $G(t_i)$ and is equal to zero for a censored subject.

To get a single score across all N observations of the test set, we return the average of the time-integrated observation-wise scores:

$$\sum_{i=1}^{N} L(S_i, t_i, \delta_i)/N$$

$$L_{ISS}(S, t|t^*) = [(S(t^*))I(t \leq t^*, \delta = 1)(1/G(t))] + [((1 - S(t^*)))I(t > t^*)(1/G(t^*))]$$

where G is the Kaplan-Meier estimate of the censoring distribution.

The re-weighted ISS, RISS is given by

$$L_{RISS}(S, t|t^*) = [(S(t^*))I(t \leq t^*, \delta = 1)(1/G(t))] + [((1 - S(t^*)))I(t > t^*)(1/G(t))]$$

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvSchmid$new()
mlr_measures$get("surv.schmid")
msr("surv.schmid")
```

Parameters

Id	Type	Default	Levels	Range
integrated	logical	TRUE	TRUE, FALSE	-
times	untyped	-		-
t_max	numeric	-		$[0, \infty)$
p_max	numeric	-		$[0, 1]$
method	integer	2		$[1, 2]$
se	logical	FALSE	TRUE, FALSE	-
proper	logical	FALSE	TRUE, FALSE	-
eps	numeric	0.001		$[0, 1]$
ERV	logical	FALSE	TRUE, FALSE	-

Meta Information

- Type: "surv"
- Range: $[0, \infty)$
- Minimize: TRUE
- Required prediction: distr

Parameter details

- integrated (logical(1))
  If TRUE (default), returns the integrated score (eg across time points); otherwise, not integrated (eg at a single time point).

- times (numeric())
  If integrated == TRUE then a vector of time-points over which to integrate the score. If integrated == FALSE then a single time point at which to return the score.

- t_max (numeric(1))
  Cutoff time τ^* (i.e. time horizon) to evaluate the measure up to. Mutually exclusive with p_max or times. This will effectively remove test observations for which the observed time (event or censoring) is strictly more than t_max. It's recommended to set t_max to avoid division by eps, see Details. If t_max is not specified, an Inf time horizon is assumed.

- p_max (numeric(1))
  The proportion of censoring to integrate up to in the given dataset. Mutually exclusive with times or t_max.
- method (integer(1))
  If integrate == TRUE, this selects the integration weighting method. method == 1 corresponds to weighting each time-point equally and taking the mean score over discrete time-points. method == 2 corresponds to calculating a mean weighted by the difference between time-points. method == 2 is the default value, to be in line with other packages.
- se (logical(1))
  If TRUE then returns standard error of the measure otherwise returns the mean across all individual scores, e.g. the mean of the per observation scores. Default is FALSE (returns the mean).
- proper (logical(1))
  If TRUE then weights scores by the censoring distribution at the observed event time, which results in a strictly proper scoring rule if censoring and survival time distributions are independent and a sufficiently large dataset is used. If FALSE then weights scores by the Graf method which is the more common usage but the loss is not proper.
- eps (numeric(1))
  Very small number to substitute zero values in order to prevent errors in e.g. log(0) and/or division-by-zero calculations. Default value is 0.001.
- ERV (logical(1))
  If TRUE then the Explained Residual Variation method is applied, which means the score is standardized against a Kaplan-Meier baseline. Default is FALSE.

Properness

RISS is strictly proper when the censoring distribution is independent of the survival distribution and when $G(t)$ is fit on a sufficiently large dataset. ISS is never proper. Use proper = FALSE for ISS and proper = TRUE for RISS. Results may be very different if many observations are censored at the last observed time due to division by $1/eps$ in proper = TRUE.

Time points used for evaluation

If the times argument is not specified (NULL), then the unique (and sorted) time points from the **test set** are used for evaluation of the time-integrated score. This was a design decision due to the fact that different predicted survival distributions $S(t)$ usually have a **discretized time domain** which may differ, i.e. in the case the survival predictions come from different survival learners. Essentially, using the same set of time points for the calculation of the score minimizes the bias that would come from using different time points. We note that $S(t)$ is by default constantly interpolated for time points that fall outside its discretized time domain.

Naturally, if the times argument is specified, then exactly these time points are used for evaluation. A warning is given to the user in case some of the specified times fall outside of the time point range of the test set. The assumption here is that if the test set is large enough, it should have a time domain/range similar to the one from the train set, and therefore time points outside that domain might lead to interpolation or extrapolation of $S(t)$.

Implementation differences

If comparing the integrated graf score to other packages, e.g. **pec**, then `method = 2` should be used. However the results may still be very slightly different as this package uses `survfit` to estimate the censoring distribution, in line with the Graf 1999 paper; whereas some other packages use `prodlim` with `reverse = TRUE` (meaning Kaplan-Meier is not used).

Data used for Estimating Censoring Distribution

If `task` and `train_set` are passed to `$score` then $G(t)$ is fit on training data, otherwise testing data. The first is likely to reduce any bias caused by calculating parts of the measure on the test data it is evaluating. The training data is automatically used in scoring resamplings.

Time Cutoff Details

If `t_max` or `p_max` is given, then $G(t)$ will be fitted using **all observations** from the train set (or test set) and only then the cutoff time will be applied. This is to ensure that more data is used for fitting the censoring distribution via the Kaplan-Meier. Setting the `t_max` can help alleviate inflation of the score when `proper` is `TRUE`, in cases where an observation is censored at the last observed time point. This results in $G(t_{max}) = 0$ and the use of `eps` instead (when `t_max` is `NULL`).

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvSchmid`

Methods

Public methods:

- `MeasureSurvSchmid$new()`
- `MeasureSurvSchmid$clone()`

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

Usage:

```
MeasureSurvSchmid$new(ERV = FALSE)
```

Arguments:

`ERV (logical(1))`
Standardize measure against a Kaplan-Meier baseline (Explained Residual Variation)

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Schemper, Michael, Henderson, Robin (2000). “Predictive Accuracy and Explained Variation in Cox Regression.” *Biometrics*, **56**, 249–255. doi:10.1002/sim.1486.

Schmid, Matthias, Hielscher, Thomas, Augustin, Thomas, Gefeller, Olaf (2011). “A Robust Alternative to the Schemper-Henderson Estimator of Prediction Error.” *Biometrics*, **67**(2), 524–535. doi:10.1111/j.15410420.2010.01459.x.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.song_a` `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other Probabilistic survival measures: `mlr_measures_surv.graf`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.rcll`

Other distr survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.rcl`

`mlr_measures_surv.song_auc` *Song and Zhou’s AUC Survival Measure*

Description

Calls `survAUC::AUC.sh()`.

Assumes Cox PH model specification.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in `mlr3proba`.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvSongAUC$new()
mlr_measures$get("surv.song_auc")
msr("surv.song_auc")
```

Parameters

Id	Type	Default	Levels
times	untyped	-	
integrated	logical	TRUE	TRUE, FALSE
type	character	incident	incident, cumulative

Meta Information

- Type: "surv"
- Range: $[0, 1]$
- Minimize: FALSE
- Required prediction: lp

Parameter details

- times (numeric())
  If integrated == TRUE then a vector of time-points over which to integrate the score. If integrated == FALSE then a single time point at which to return the score.
- integrated (logical(1))
  If TRUE (default), returns the integrated score (eg across time points); otherwise, not integrated (eg at a single time point).
- type (character(1))
  A string defining the type of true positive rate (TPR): incident refers to incident TPR, cumulative refers to cumulative TPR.

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> mlr3proba::MeasureSurvAUC -> MeasureSurvSongAUC

Methods

Public methods:

- MeasureSurvSongAUC$new()
- MeasureSurvSongAUC$clone()

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

Usage:

```
MeasureSurvSongAUC$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Song, Xiao, Zhou, Xiao-Hua (2008). “A semiparametric approach for the covariate specific ROC curve with survival outcome.” *Statistica Sinica*, **18**(3), 947–65. https://www.jstor.org/stable/24308524.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmi`
`mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other AUC survival measures: `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`

Other lp survival measures: `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

`mlr_measures_surv.song_tnr` *Song and Zhou’s TNR Survival Measure*

Description

Calls `survAUC::spec.sh()`.

Assumes Cox PH model specification.

`times` and `lp_thresh` are arbitrarily set to `0` to prevent crashing, these should be further specified.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in `mlr3proba`.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvSongTNR$new()
mlr_measures$get("surv.song_tnr")
msr("surv.song_tnr")
```

Parameters

Id	Type	Default	Range
times	numeric	-	$[0, \infty)$
lp_thresh	numeric	0	$(-\infty, \infty)$

Meta Information

- Type: "surv"
- Range: $[0, 1]$
- Minimize: FALSE
- Required prediction: lp

Parameter details

- times (numeric())
  If integrated == TRUE then a vector of time-points over which to integrate the score. If integrated == FALSE then a single time point at which to return the score.
- lp_thresh (numeric(1))
  Determines the cutoff threshold of the linear predictor in the calculation of the TPR/TNR scores.

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> mlr3proba::MeasureSurvAUC -> MeasureSurvSongTNR

Methods

Public methods:

- MeasureSurvSongTNR$new()
- MeasureSurvSongTNR$clone()

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

Usage:

```
MeasureSurvSongTNR$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Song, Xiao, Zhou, Xiao-Hua (2008). "A semiparametric approach for the covariate specific ROC curve with survival outcome." *Statistica Sinica*, **18**(3), 947–65. https://www.jstor.org/stable/24308524.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.cindex, mlr_measures_surv.dcalib, mlr_measures_surv.graf, mlr_measures_surv.hung_auc, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mae, mlr_measures_surv.mse, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_r2, mlr_measures_surv.rcll, mlr_measures_surv.rmse, mlr_measures_surv.schmic mlr_measures_surv.song_auc, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Other AUC survival measures: mlr_measures_surv.chambless_auc, mlr_measures_surv.hung_auc, mlr_measures_surv.song_auc, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr

Other lp survival measures: mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.hung_auc, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_r2, mlr_measures_surv.song_auc, mlr_measures_surv.song_tpr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

---

mlr_measures_surv.song_tpr

Song and Zhou's TPR Survival Measure

Description

Calls survAUC::sens.sh().

Assumes Cox PH model specification.

times and lp_thresh are arbitrarily set to 0 to prevent crashing, these should be further specified.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in mlr3proba.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvSongTPR$new()
mlr_measures$get("surv.song_tpr")
msr("surv.song_tpr")
```

Parameters

Id	Type	Default	Levels	Range
times	numeric	-		$[0, \infty)$
lp_thresh	numeric	0		$(-\infty, \infty)$
type	character	incident	incident, cumulative	-

Meta Information

- Type: `"surv"`
- Range: $[0, 1]$
- Minimize: `FALSE`
- Required prediction: `lp`

Parameter details

- `times` (`numeric()`)
  If `integrated` == `TRUE` then a vector of time-points over which to integrate the score. If `integrated` == `FALSE` then a single time point at which to return the score.

- `lp_thresh` (`numeric(1)`)
  Determines the cutoff threshold of the linear predictor in the calculation of the TPR/TNR scores.

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `mlr3proba::MeasureSurvAUC` -> `MeasureSurvSongTPR`

Methods

Public methods:

- `MeasureSurvSongTPR$new()`
- `MeasureSurvSongTPR$clone()`

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

Usage:

```
MeasureSurvSongTPR$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Song, Xiao, Zhou, Xiao-Hua (2008). "A semiparametric approach for the covariate specific ROC curve with survival outcome." *Statistica Sinica*, **18**(3), 947–65. https://www.jstor.org/stable/24308524.

See Also

Other survival measures: mlr_measures_surv.calib_alpha, mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.cindex, mlr_measures_surv.dcalib, mlr_measures_surv.graf, mlr_measures_surv.hung_auc, mlr_measures_surv.intlogloss, mlr_measures_surv.logloss, mlr_measures_surv.mae, mlr_measures_surv.mse, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_r2, mlr_measures_surv.rcll, mlr_measures_surv.rmse, mlr_measures_surv.schmic
mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

Other AUC survival measures: mlr_measures_surv.chambless_auc, mlr_measures_surv.hung_auc, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr

Other lp survival measures: mlr_measures_surv.calib_beta, mlr_measures_surv.chambless_auc, mlr_measures_surv.hung_auc, mlr_measures_surv.nagelk_r2, mlr_measures_surv.oquigley_r2, mlr_measures_surv.song_auc, mlr_measures_surv.song_tnr, mlr_measures_surv.uno_auc, mlr_measures_surv.uno_tnr, mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

---

mlr_measures_surv.uno_auc

Uno's AUC Survival Measure

Description

Calls survAUC::AUC.uno().

Assumes random censoring.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in mlr3proba.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvUnoAUC$new()
mlr_measures$get("surv.uno_auc")
msr("surv.uno_auc")
```

Parameters

Id	Type	Default	Levels
integrated	logical	TRUE	TRUE, FALSE
times	untyped	-	

Meta Information

- Type: `"surv"`
- Range: $[0, 1]$
- Minimize: `FALSE`
- Required prediction: `lp`

Parameter details

- `integrated` (`logical(1)`)
  If `TRUE` (default), returns the integrated score (eg across time points); otherwise, not integrated (eg at a single time point).

- `times` (`numeric()`)
  If `integrated` == `TRUE` then a vector of time-points over which to integrate the score. If `integrated` == `FALSE` then a single time point at which to return the score.

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `mlr3proba::MeasureSurvAUC` -> `MeasureSurvUnoAUC`

Methods

Public methods:

- `MeasureSurvUnoAUC$new()`
- `MeasureSurvUnoAUC$clone()`

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

Usage:

```
MeasureSurvUnoAUC$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Uno H, Cai T, Tian L, Wei LJ (2007). "Evaluating Prediction Rules fort-Year Survivors With Censored Regression Models." *Journal of the American Statistical Association*, **102**(478), 527–537. doi:10.1198/016214507000000149.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmi` `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other AUC survival measures: `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`

Other lp survival measures: `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

`mlr_measures_surv.uno_tnr`

Uno's TNR Survival Measure

Description

Calls `survAUC::spec.uno()`.

Assumes random censoring.

`times` and `lp_thresh` are arbitrarily set to `0` to prevent crashing, these should be further specified.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in `mlr3proba`.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvUnoTNR$new()
mlr_measures$get("surv.uno_tnr")
msr("surv.uno_tnr")
```

Parameters

Id	Type	Default	Range
times	numeric	-	$[0, \infty)$
lp_thresh	numeric	0	$(-\infty, \infty)$

Meta Information

- Type: `"surv"`
- Range: $[0, 1]$
- Minimize: `FALSE`
- Required prediction: `lp`

Parameter details

- `times` (`numeric()`)
  A vector of time-points at which we calculate the TPR/TNR scores.

- `lp_thresh` (`numeric(1)`)
  Determines the cutoff threshold of the linear predictor in the calculation of the TPR/TNR scores.

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `mlr3proba::MeasureSurvAUC` -> `MeasureSurvUnoTNR`

Methods

Public methods:

- `MeasureSurvUnoTNR$new()`
- `MeasureSurvUnoTNR$clone()`

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

Usage:

```
MeasureSurvUnoTNR$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Uno H, Cai T, Tian L, Wei LJ (2007). "Evaluating Prediction Rules fort-Year Survivors With Censored Regression Models." *Journal of the American Statistical Association*, **102**(478), 527–537. doi:10.1198/016214507000000149.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid` `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other AUC survival measures: `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tpr`

Other lp survival measures: `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

`mlr_measures_surv.uno_tpr` *Uno's TPR Survival Measure*

Description

Calls `survAUC::sens.uno()`.

Assumes random censoring.

`times` and `lp_thresh` are arbitrarily set to `0` to prevent crashing, these should be further specified.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in `mlr3proba`.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvUnoTPR$new()
mlr_measures$get("surv.uno_tpr")
msr("surv.uno_tpr")
```

Parameters

Id	Type	Default	Range
times	numeric	-	$[0, \infty)$
lp_thresh	numeric	0	$(-\infty, \infty)$

Meta Information

- Type: "surv"
- Range: $[0, 1]$
- Minimize: FALSE
- Required prediction: lp

Parameter details

- times (numeric())
  A vector of time-points at which we calculate the TPR/TNR scores.
- lp_thresh (numeric(1))
  Determines the cutoff threshold of the linear predictor in the calculation of the TPR/TNR scores.

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> mlr3proba::MeasureSurvAUC -> MeasureSurvUnoTPR

Methods

Public methods:

- MeasureSurvUnoTPR$new()
- MeasureSurvUnoTPR$clone()

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

Usage:

```
MeasureSurvUnoTPR$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Uno H, Cai T, Tian L, Wei LJ (2007). "Evaluating Prediction Rules fort-Year Survivors With Censored Regression Models." *Journal of the American Statistical Association*, **102**(478), 527–537. doi:10.1198/016214507000000149.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`
`mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.xu_r2`

Other AUC survival measures: `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`

Other lp survival measures: `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.xu_r2`

`mlr_measures_surv.xu_r2` *Xu and O'Quigley's R2 Survival Measure*

Description

Calls `survAUC::XO()`.

Assumes Cox PH model specification.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates some of these measures may be re-written and implemented directly in `mlr3proba`.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
MeasureSurvXuR2$new()
mlr_measures$get("surv.xu_r2")
msr("surv.xu_r2")
```

Parameters

Empty ParamSet

Meta Information

- Type: `"surv"`
- Range: $[0, 1]$
- Minimize: `FALSE`
- Required prediction: `lp`

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvXuR2`

Methods

Public methods:

- `MeasureSurvXuR2$new()`
- `MeasureSurvXuR2$clone()`

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

Usage:

```
MeasureSurvXuR2$new()
```

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

Usage:

```
MeasureSurvXuR2$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Xu R, O'Quigley J (1999). "A R2 type measure of dependence for proportional hazards models." *Journal of Nonparametric Statistics*, **12**(1), 83–107. doi:10.1080/10485259908832799.

See Also

Other survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.dcalib`, `mlr_measures_surv.graf`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rcll`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmic` `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`

Other R2 survival measures: `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`

Other lp survival measures: `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`

`mlr_pipeops_compose_breslow_distr`

Wrap a learner into a PipeOp with survival predictions estimated by the Breslow estimator

---

Description

Composes a survival distribution (`distr`) using the linear predictor predictions (`lp`) from a given LearnerSurv during training and prediction, utilizing the breslow estimator. The specified `learner` must be capable of generating `lp`-type predictions (e.g., a Cox-type model).

Dictionary

This PipeOp can be instantiated via the Dictionary mlr_pipeops or with the associated sugar function po():

```
PipeOpBreslow$new(learner)
mlr_pipeops$get("breslowcompose", learner)
po("breslowcompose", learner, breslow.overwrite = TRUE)
```

Input and Output Channels

PipeOpBreslow is like a LearnerSurv. It has one input channel, named `input` that takes a TaskSurv during training and another TaskSurv during prediction. PipeOpBreslow has one output channel named `output`, producing `NULL` during training and a PredictionSurv during prediction.

State

The `$state` slot stores the `times` and `status` survival target variables of the train TaskSurv as well as the `lp` predictions on the train set.

Parameters

The parameters are:

- `breslow.overwrite` :: `logical(1)`
  If `FALSE` (default) then the compositor does nothing and returns the input `learner`'s PredictionSurv. If `TRUE` or in the case that the input `learner` doesn't have `distr` predictions, then the `distr` is overwritten with the `distr` composed from `lp` and the train set information using breslow. This is useful for changing the prediction `distr` from one model form to another.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpBreslow`

Active bindings

learner (mlr3::Learner)
The input survival learner.

Methods

Public methods:

- PipeOpBreslow$new()
- PipeOpBreslow$clone()

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

Usage:

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

Arguments:

learner (LearnerSurv)
Survival learner which must provide lp-type predictions

id (character(1))
Identifier of the resulting object. If NULL (default), it will be set as the id of the input learner.

param_vals (list())
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Cox DR (1972). "Regression Models and Life-Tables." *Journal of the Royal Statistical Society: Series B (Methodological)*, **34**(2), 187–202. doi:10.1111/j.25176161.1972.tb00899.x.

Lin, Y. D (2007). "On the Breslow estimator." *Lifetime Data Analysis*, **13**(4), 471-480. doi:10.1007/s109850079048y.

See Also

pipeline_distrcompositor

Other survival compositors: mlr_pipeops_crankcompose, mlr_pipeops_distrcompose, mlr_pipeops_responsecompose

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)
  task = tsk("rats")
  part = partition(task, ratio = 0.8)
  train_task = task$clone()$filter(part$train)
  test_task = task$clone()$filter(part$test)

  learner = lrn("surv.coxph") # learner with lp predictions
  b = po("breslowcompose", learner = learner, breslow.overwrite = TRUE)

  b$train(list(train_task))
  p = b$predict(list(test_task))[[1L]]

## End(Not run)
```

---

mlr_pipeops_compose_probregr

PipeOpProbregr

Description

[Experimental]

Combines a predicted `response` and `se` from PredictionRegr with a specified probability distribution to estimate (or 'compose') a `distr` prediction.

Dictionary

This PipeOp can be instantiated via the dictionary mlr3pipelines::mlr_pipeops or with the associated sugar function `mlr3pipelines::po()`:

```
PipeOpProbregr$new()
mlr_pipeops$get("compose_probregr")
po("compose_probregr")
```

Input and Output Channels

PipeOpProbregr has two input channels named `"input_response"` and `"input_se"`, which take `NULL` during training and two PredictionRegrs during prediction, these should respectively contain the `response` and `se` return type, the same object can be passed twice.

The output during prediction is a PredictionRegr with the "response" from `input_response`, the "se" from `input_se` and a "distr" created from combining the two.

State

The `$state` is left empty (`list()`).

Parameters

- dist :: character(1)
  Location-scale distribution to use for composition. Current choices are "Uniform" (default), "Normal", "Cauchy", "Gumbel", "Laplace", "Logistic". All implemented via distr6.

Internals

The composition is created by substituting the response and se predictions into the distribution location and scale parameters respectively.

Super class

mlr3pipelines::PipeOp -> PipeOpProbregr

Methods

Public methods:

- PipeOpProbregr$new()
- PipeOpProbregr$clone()

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

Usage:

```
PipeOpProbregr$new(id = "compose_probregr", param_vals = list())
```

Arguments:

id (character(1))
Identifier of the resulting object.

param_vals (list())
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)
  set.seed(1)
  task = tsk("boston_housing")

  # Option 1: Use a learner that can predict se
  learn = lrn("regr.featureless", predict_type = "se")
  pred = learn$train(task)$predict(task)
  poc = po("compose_probregr")
  poc$predict(list(pred, pred))[[1]]

  # Option 2: Use two learners, one for response and the other for se
  learn_response = lrn("regr.rpart")
  learn_se = lrn("regr.featureless", predict_type = "se")
  pred_response = learn_response$train(task)$predict(task)
  pred_se = learn_se$train(task)$predict(task)
  poc = po("compose_probregr")
  poc$predict(list(pred_response, pred_se))[[1]]

## End(Not run)
```

mlr_pipeops_crankcompose *PipeOpCrankCompositor*

Description

Uses a predicted distr in a PredictionSurv to estimate (or 'compose') a crank prediction.

Dictionary

This PipeOp can be instantiated via the dictionary mlr3pipelines::mlr_pipeops or with the associated sugar function `mlr3pipelines::po()`:

```
PipeOpCrankCompositor$new()
mlr_pipeops$get("crankcompose")
po("crankcompose")
```

Input and Output Channels

PipeOpCrankCompositor has one input channel named "input", which takes NULL during training and PredictionSurv during prediction.

PipeOpCrankCompositor has one output channel named "output", producing NULL during training and a PredictionSurv during prediction.

The output during prediction is the PredictionSurv from the input but with the crank predict type overwritten by the given estimation method.

State

The $state is left empty (list()).

Parameters

- `method` :: `character(1)`
  Determines what method should be used to produce a continuous ranking from the distribution. Currently only `mort` is supported, which is the sum of the cumulative hazard, also called *expected/ensemble mortality*, see Ishwaran et al. (2008). For more details, see `get_mortality()`.
- `overwrite` :: `logical(1)`
  If `FALSE` (default) and the prediction already has a `crank` prediction, then the compositor returns the input prediction unchanged. If `TRUE`, then the `crank` will be overwritten.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpCrankCompositor`

Methods

Public methods:

- `PipeOpCrankCompositor$new()`
- `PipeOpCrankCompositor$clone()`

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

Usage:

```
PipeOpCrankCompositor$new(id = "crankcompose", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Sonabend, Raphael, Bender, Andreas, Vollmer, Sebastian (2022). "Avoiding C-hacking when evaluating survival distribution predictions with discrimination measures." *Bioinformatics*. ISSN 1367-4803, doi:10.1093/BIOINFORMATICS/BTAC451, `https://academic.oup.com/bioinformatics/advance-article/doi/10.1093/bioinformatics/btac451/6640155`.

Ishwaran, Hemant, Kogalur, B U, Blackstone, H E, Lauer, S M, others (2008). "Random survival forests." *The Annals of applied statistics*, **2**(3), 841–860.

See Also

pipeline_crankcompositor

Other survival compositors: `mlr_pipeops_compose_breslow_distr`, `mlr_pipeops_distrcompose`, `mlr_pipeops_responsecompose`

Examples

```
## Not run:
  library(mlr3pipelines)
  task = tsk("rats")

  # change the crank prediction type of a Cox's model predictions
  pred = lrn("surv.coxph")$train(task)$predict(task)
  poc = po("crankcompose", param_vals = list(overwrite = TRUE))
  poc$predict(list(pred))[[1L]]

## End(Not run)
```

`mlr_pipeops_distrcompose` *PipeOpDistrCompositor*

Description

[Experimental]

Estimates (or 'composes') a survival distribution from a predicted baseline survival distribution (`distr`) and a linear predictor (`lp`) from two PredictionSurvs.

Compositor Assumptions:

- The baseline `distr` is a discrete estimator, e.g. surv.kaplan.
- The composed `distr` is of a linear form

Dictionary

This PipeOp can be instantiated via the dictionary mlr3pipelines::mlr_pipeops or with the associated sugar function `mlr3pipelines::po()`:

```
PipeOpDistrCompositor$new()
mlr_pipeops$get("distrcompose")
po("distrcompose")
```

Input and Output Channels

PipeOpDistrCompositor has two input channels, `"base"` and `"pred"`. Both input channels take `NULL` during training and PredictionSurv during prediction.

PipeOpDistrCompositor has one output channel named `"output"`, producing `NULL` during training and a PredictionSurv during prediction.

The output during prediction is the PredictionSurv from the `"pred"` input but with an extra (or overwritten) column for the `distr` predict type; which is composed from the `distr` of `"base"` and the `lp` of `"pred"`. If no `lp` predictions have been made or exist, then the `"pred"` is returned unchanged.

State

The `$state` is left empty (`list()`).

Parameters

The parameters are:

- `form` :: `character(1)`
  Determines the form that the predicted linear survival model should take. This is either, accelerated-failure time, `aft`, proportional hazards, `ph`, or proportional odds, `po`. Default `aft`.
- `overwrite` :: `logical(1)`
  If `FALSE` (default) then if the "pred" input already has a `distr`, the compositor does nothing and returns the given PredictionSurv. If `TRUE`, then the `distr` is overwritten with the `distr` composed from `lp` - this is useful for changing the prediction `distr` from one model form to another.
- `scale_lp` :: `logical(1)`
  This option is only applicable to `form` equal to `"aft"`. If `TRUE`, it min-max scales the linear prediction scores to be in the interval $[0,1]$, avoiding extrapolation of the baseline $S_0(t)$ on the transformed time points $\frac{t}{\exp(lp)}$, as these will be $\in [\frac{t}{e}, t]$, and so always smaller than the maximum time point for which we have estimated $S_0(t)$. Note that this is just a **heuristic** to get reasonable results in the case you observe survival predictions to be e.g. constant after the AFT composition and it definitely provides no guarantee for creating calibrated distribution predictions (as none of these methods do). Therefore, it is set to `FALSE` by default.

Internals

The respective `forms` above have respective survival distributions:

$$aft : S(t) = S_0(\frac{t}{\exp(lp)})$$

$$ph : S(t) = S_0(t)^{\exp(lp)}$$

$$po : S(t) = \frac{S_0(t)}{\exp(-lp) + (1 - \exp(-lp))S_0(t)}$$

where S_0 is the estimated baseline survival distribution, and lp is the predicted linear predictor.

For an example use of the `"aft"` composition using Kaplan-Meier as a baseline distribution, see Norman et al. (2024).

Super class

mlr3pipelines::PipeOp -> PipeOpDistrCompositor

Methods

Public methods:

- PipeOpDistrCompositor$new()
- PipeOpDistrCompositor$clone()

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

Usage:

```
PipeOpDistrCompositor$new(id = "distrcompose", param_vals = list())
```

Arguments:

id (character(1))
Identifier of the resulting object.

param_vals (list())
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Norman, A P, Li, Wanlu, Jiang, Wenyu, Chen, E B (2024). “deepAFT: A nonlinear accelerated failure time model with artificial neural network.” *Statistics in Medicine*. doi:10.1002/sim.10152.

See Also

pipeline_distrcompositor

Other survival compositors: mlr_pipeops_compose_breslow_distr, mlr_pipeops_crankcompose, mlr_pipeops_responsecompose

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)
  task = tsk("rats")

  base = lrn("surv.kaplan")$train(task)$predict(task)
  pred = lrn("surv.coxph")$train(task)$predict(task)
  # let's change the distribution prediction of Cox (Breslow-based) to an AFT form:
  pod = po("distrcompose", param_vals = list(form = "aft", overwrite = TRUE))
  pod$predict(list(base = base, pred = pred))[[1]]

## End(Not run)
```

mlr_pipeops_responsecompose *PipeOpResponseCompositor*

Description

Uses a predicted survival distribution (`distr`) in a PredictionSurv to estimate (or 'compose') an expected survival time (`response`) prediction. Practically, this `PipeOp` summarizes an observation's survival curve/distribution to a single number which can be either the restricted mean survival time or the median survival time.

Dictionary

This PipeOp can be instantiated via the dictionary mlr3pipelines::mlr_pipeops or with the associated sugar function `mlr3pipelines::po()`:

```
PipeOpResponseCompositor$new()
mlr_pipeops$get("responsecompose")
po("responsecompose")
```

Input and Output Channels

PipeOpResponseCompositor has one input channel named `"input"`, which takes `NULL` during training and PredictionSurv during prediction.

PipeOpResponseCompositor has one output channel named `"output"`, producing `NULL` during training and a PredictionSurv during prediction.

The output during prediction is the PredictionSurv from the input but with the `response` predict type overwritten by the given method.

State

The `$state` is left empty (`list()`).

Parameters

- `method` :: `character(1)`
  Determines what method should be used to produce a survival time (response) from the survival distribution. Available methods are `"rmst"` and `"median"`, corresponding to the *restricted mean survival time* and the *median survival time* respectively.
- `tau` :: `numeric(1)`
  Determines the time point up to which we calculate the restricted mean survival time (works only for the `"rmst"` method). If `NULL` (default), all the available time points in the predicted survival distribution will be used.

- `add_crank` :: `logical(1)`
  If `TRUE` then `crank` predict type will be set as `-response` (as higher survival times correspond to lower risk). Works only if `overwrite` is `TRUE`.
- `overwrite` :: `logical(1)`
  If `FALSE` (default) and the prediction already has a `response` prediction, then the compositor returns the input prediction unchanged. If `TRUE`, then the `response` (and the `crank`, if `add_crank` is `TRUE`) will be overwritten.

Internals

The restricted mean survival time is the default/preferred method and is calculated as follows:

$$T_{i,rmst} \approx \sum_{t_j \in [0,\tau]} (t_j - t_{j-1}) S_i(t_j)$$

where T is the expected survival time, τ is the time cutoff/horizon and $S_i(t_j)$ are the predicted survival probabilities of observation i for all the t_j time points.

The $T_{i,median}$ survival time is just the first time point for which the survival probability is less than 0.5. If no such time point exists (e.g. when the survival distribution is not proper due to high censoring) we return the last time point. This is **not a good estimate to use in general**, only a reasonable substitute in such cases.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpResponseCompositor`

Methods

Public methods:

- `PipeOpResponseCompositor$new()`
- `PipeOpResponseCompositor$clone()`

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

Usage:

```
PipeOpResponseCompositor$new(id = "responsecompose", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Zhao, Lihui, Claggett, Brian, Tian, Lu, Uno, Hajime, Pfeffer, A. M, Solomon, D. S, Trippa, Lorenzo, Wei, J. L (2016). “On the restricted mean survival time curve in survival analysis.” *Biometrics*, **72**(1), 215–221. ISSN 1541-0420, doi:10.1111/BIOM.12384, https://onlinelibrary.wiley.com/doi/full/10.1111/biom.12384.

See Also

pipeline_responsecompositor

Other survival compositors: mlr_pipeops_compose_breslow_distr, mlr_pipeops_crankcompose, mlr_pipeops_distrcompose

Examples

```
## Not run:
  library(mlr3pipelines)
  task = tsk("rats")

  # add survival time prediction type to the predictions of a Cox model
  # Median survival time as response
  pred = lrn("surv.coxph")$train(task)$predict(task)
  por = po("responsecompose", param_vals = list(method = "median", overwrite = TRUE))
  por$predict(list(pred))[[1L]]
  # mostly improper survival distributions, "median" sets the survival time
  # to the last time point

  # RMST (default) as response, while also changing the `crank` to `-response`
  por = po("responsecompose", param_vals = list(overwrite = TRUE, add_crank = TRUE))
  por$predict(list(pred))[[1L]]

## End(Not run)
```

mlr_pipeops_survavg *PipeOpSurvAvg*

Description

Perform (weighted) prediction averaging from survival PredictionSurvs by connecting `PipeOpSurvAvg` to multiple PipeOpLearner outputs.

The resulting prediction will aggregate any predict types that are contained within all inputs. Any predict types missing from at least one input will be set to `NULL`. These are aggregated as follows:

- `"response"`, `"crank"`, and `"lp"` are all a weighted average from the incoming predictions.
- `"distr"` is a distr6::VectorDistribution containing distr6::MixtureDistributions.

Weights can be set as a parameter; if none are provided, defaults to equal weights for each prediction.

Input and Output Channels

Input and output channels are inherited from PipeOpEnsemble with a PredictionSurv for inputs and outputs.

State

The `$state` is left empty (`list()`).

Parameters

The parameters are the parameters inherited from the PipeOpEnsemble.

Internals

Inherits from PipeOpEnsemble by implementing the `private$weighted_avg_predictions()` method.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpEnsemble` -> `PipeOpSurvAvg`

Methods

Public methods:

- `PipeOpSurvAvg$new()`
- `PipeOpSurvAvg$clone()`

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

Usage:

```
PipeOpSurvAvg$new(innum = 0, id = "survavg", param_vals = list(), ...)
```

Arguments:

`innum (numeric(1))`
Determines the number of input channels. If `innum` is 0 (default), a vararg input channel is created that can take an arbitrary number of inputs.

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

`... (ANY)`
Additional arguments passed to mlr3pipelines::PipeOpEnsemble.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: PipeOpPredTransformer, PipeOpTaskTransformer, PipeOpTransformer, mlr_pipeops_trafopred_cla
mlr_pipeops_trafopred_classifsurv_disctime, mlr_pipeops_trafopred_regrsurv, mlr_pipeops_trafopred_sur
mlr_pipeops_trafotask_regrsurv, mlr_pipeops_trafotask_survclassif_IPCW, mlr_pipeops_trafotask_survcla
mlr_pipeops_trafotask_survregr

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  task = tsk("rats")
  p1 = lrn("surv.coxph")$train(task)$predict(task)
  p2 = lrn("surv.kaplan")$train(task)$predict(task)
  poc = po("survavg", param_vals = list(weights = c(0.2, 0.8)))
  poc$predict(list(p1, p2))

## End(Not run)
```

---

mlr_pipeops_trafopred_classifsurv_disctime
PipeOpPredClassifSurvDiscTime

Description

Transform PredictionClassif to PredictionSurv by converting event probabilities of a pseudo status variable (discrete time hazards) to survival probabilities using the product rule (Tutz et al. 2016):

$$P_k = p_k \cdot ... \cdot p_1$$

Where:

- We assume that continuous time is divided into time intervals $[0, t_1), [t_1, t_2), ..., [t_n, \infty)$
- $P_k = P(T > t_k)$ is the survival probability at time t_k
- h_k is the discrete-time hazard (classifier prediction), i.e. the conditional probability for an event in the k-interval.
- $p_k = 1 - h_k = P(T \geq t_k | T \geq t_{k-1})$

Dictionary

This PipeOp can be instantiated via the dictionary mlr3pipelines::mlr_pipeops or with the associated sugar function `mlr3pipelines::po()`:

```
PipeOpPredClassifSurvDiscTime$new()
mlr_pipeops$get("trafopred_classifsurv_disctime")
po("trafopred_classifsurv_disctime")
```

Input and Output Channels

The input is a PredictionClassif and a data.table with the transformed data both generated by PipeOpTaskSurvClassifDiscTime. The output is the input PredictionClassif transformed to a PredictionSurv. Only works during prediction phase.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpPredClassifSurvDiscTime`

Methods

Public methods:

- `PipeOpPredClassifSurvDiscTime$new()`
- `PipeOpPredClassifSurvDiscTime$clone()`

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

Usage:

```
PipeOpPredClassifSurvDiscTime$new(id = "trafopred_classifsurv_disctime")
```

Arguments:

`id` (character(1))
Identifier of the resulting object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Tutz, Gerhard, Schmid, Matthias (2016). *Modeling Discrete Time-to-Event Data*, series Springer Series in Statistics. Springer International Publishing. ISBN 978-3-319-28156-8 978-3-319-28158-2, `http://link.springer.com/10.1007/978-3-319-28158-2`.

See Also

Other PipeOps: `PipeOpPredTransformer`, `PipeOpTaskTransformer`, `PipeOpTransformer`, `mlr_pipeops_survavg`, `mlr_pipeops_trafopred_classifsurv_IPCW`, `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_survregr` `mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_IPCW`, `mlr_pipeops_trafotask_survclas` `mlr_pipeops_trafotask_survregr`

Other Transformation PipeOps: `mlr_pipeops_trafopred_classifsurv_IPCW`, `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_survregr`, `mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_IPCW` `mlr_pipeops_trafotask_survclassif_disctime`, `mlr_pipeops_trafotask_survregr`

mlr_pipeops_trafopred_classifsurv_IPCW *PipeOpPredClassifSurvIPCW*

Description

Transform PredictionClassif to PredictionSurv using the **I**nverse **P**robability of **C**ensoring **W**eights (IPCW) method by Vock et al. (2016).

Dictionary

This PipeOp can be instantiated via the dictionary mlr3pipelines::mlr_pipeops or with the associated sugar function `mlr3pipelines::po()`:

```
PipeOpPredClassifSurvIPCW$new()
mlr_pipeops$get("trafopred_classifsurv_IPCW")
po("trafopred_classifsurv_IPCW")
```

Input and Output Channels

The input is a PredictionClassif and a data.table containing observed times, censoring indicators and row ids, all generated by PipeOpTaskSurvClassifIPCW during the prediction phase.

The output is the input PredictionClassif transformed to a PredictionSurv. Each input classification probability prediction corresponds to the probability of having the event up to the specified cutoff time $\hat{\pi}(\boldsymbol{X}_i) = P(T_i < \tau|\boldsymbol{X}_i)$, see Vock et al. (2016) and PipeOpTaskSurvClassifIPCW. Therefore, these predictions serve as **continuous risk scores** that can be directly interpreted as `crank` predictions in the right-censored survival setting. We also map them to the survival distribution prediction `distr`, at the specified cutoff time point τ, i.e. as $S_i(\tau) = 1 - \hat{\pi}(\boldsymbol{X}_i)$. Survival measures that use the survival distribution (eg ISBS) should be evaluated exactly at the cutoff time point τ, see example.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpPredClassifSurvIPCW`

Methods

Public methods:

- `PipeOpPredClassifSurvIPCW$new()`
- `PipeOpPredClassifSurvIPCW$clone()`

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

Usage:

```
PipeOpPredClassifSurvIPCW$new(id = "trafopred_classifsurv_IPCW")
```

Arguments:

`id` (character(1))
Identifier of the resulting object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Vock, M D, Wolfson, Julian, Bandyopadhyay, Sunayan, Adomavicius, Gediminas, Johnson, E P, Vazquez-Benitez, Gabriela, O'Connor, J P (2016). "Adapting machine learning techniques to censored time-to-event health record data: A general-purpose approach using inverse probability of censoring weighting." *Journal of Biomedical Informatics*, **61**, 119–131. doi:10.1016/j.jbi.2016.03.009, `https://www.sciencedirect.com/science/article/pii/S1532046416000496`.

See Also

Other PipeOps: `PipeOpPredTransformer`, `PipeOpTaskTransformer`, `PipeOpTransformer`, `mlr_pipeops_survavg`, `mlr_pipeops_trafopred_classifsurv_disctime`, `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_surv` `mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_IPCW`, `mlr_pipeops_trafotask_survclas` `mlr_pipeops_trafotask_survregr`

Other Transformation PipeOps: `mlr_pipeops_trafopred_classifsurv_disctime`, `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_survregr`, `mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_IPC` `mlr_pipeops_trafotask_survclassif_disctime`, `mlr_pipeops_trafotask_survregr`

`mlr_pipeops_trafopred_regrsurv`

PipeOpPredRegrSurv

Description

Transform PredictionRegr to PredictionSurv.

Input and Output Channels

Input and output channels are inherited from PipeOpPredTransformer.

The output is the input PredictionRegr transformed to a PredictionSurv. Censoring can be added with the `status` hyper-parameter. `se` is ignored.

State

The `$state` is a named `list` with the `$state` elements inherited from `PipeOpPredTransformer`.

Parameters

The parameters are

- status :: (numeric(1))
  If NULL then assumed no censoring in the dataset. Otherwise should be a vector of 0/1s of same length as the prediction object, where 1 is dead and 0 censored.

Super classes

mlr3pipelines::PipeOp -> mlr3proba::PipeOpTransformer -> mlr3proba::PipeOpPredTransformer -> PipeOpPredRegrSurv

Methods

Public methods:

- PipeOpPredRegrSurv$new()
- PipeOpPredRegrSurv$clone()

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

Usage:

```
PipeOpPredRegrSurv$new(id = "trafopred_regrsurv", param_vals = list())
```

Arguments:

id (character(1))
Identifier of the resulting object.

param_vals (list())
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: PipeOpPredTransformer, PipeOpTaskTransformer, PipeOpTransformer, mlr_pipeops_survavg, mlr_pipeops_trafopred_classifsurv_IPCW, mlr_pipeops_trafopred_classifsurv_disctime, mlr_pipeops_trafopred_survregr, mlr_pipeops_trafotask_regrsurv, mlr_pipeops_trafotask_survclassif_IPC
mlr_pipeops_trafotask_survclassif_disctime, mlr_pipeops_trafotask_survregr

Other Transformation PipeOps: mlr_pipeops_trafopred_classifsurv_IPCW, mlr_pipeops_trafopred_classifsurv_c
mlr_pipeops_trafopred_survregr, mlr_pipeops_trafotask_regrsurv, mlr_pipeops_trafotask_survclassif_IPC
mlr_pipeops_trafotask_survclassif_disctime, mlr_pipeops_trafotask_survregr

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  # simple example
  pred = PredictionRegr$new(row_ids = 1:10, truth = 1:10, response = 1:10)
  po = po("trafopred_regrsurv")

  # assume no censoring
  new_pred = po$predict(list(pred = pred, task = NULL))[[1]]
  po$train(list(NULL, NULL))
  print(new_pred)

  # add censoring
  task_surv = tsk("rats")
  task_regr = po("trafotask_survregr", method = "omit")$train(list(task_surv, NULL))[[1]]
  learn = lrn("regr.featureless")
  pred = learn$train(task_regr)$predict(task_regr)
  po = po("trafopred_regrsurv")
  new_pred = po$predict(list(pred = pred, task = task_surv))[[1]]
  all.equal(new_pred$truth, task_surv$truth())

## End(Not run)
```

`mlr_pipeops_trafopred_survregr`

PipeOpPredSurvRegr

Description

Transform PredictionSurv to PredictionRegr.

Input and Output Channels

Input and output channels are inherited from PipeOpPredTransformer.

The output is the input PredictionSurv transformed to a PredictionRegr. Censoring is ignored. `crank` and `lp` predictions are also ignored.

State

The `$state` is a named `list` with the `$state` elements inherited from PipeOpPredTransformer.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3proba::PipeOpTransformer` -> `mlr3proba::PipeOpPredTransformer` -> `PipeOpPredSurvRegr`

Methods

Public methods:

- PipeOpPredSurvRegr$new()
- PipeOpPredSurvRegr$clone()

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

Usage:

```
PipeOpPredSurvRegr$new(id = "trafopred_survregr")
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: PipeOpPredTransformer, PipeOpTaskTransformer, PipeOpTransformer, mlr_pipeops_survavg, mlr_pipeops_trafopred_classifsurv_IPCW, mlr_pipeops_trafopred_classifsurv_disctime, mlr_pipeops_trafopred_regrsurv, mlr_pipeops_trafotask_regrsurv, mlr_pipeops_trafotask_survclassif_IPC
mlr_pipeops_trafotask_survclassif_disctime, mlr_pipeops_trafotask_survregr

Other Transformation PipeOps: mlr_pipeops_trafopred_classifsurv_IPCW, mlr_pipeops_trafopred_classifsurv_d
mlr_pipeops_trafopred_regrsurv, mlr_pipeops_trafotask_regrsurv, mlr_pipeops_trafotask_survclassif_IPC
mlr_pipeops_trafotask_survclassif_disctime, mlr_pipeops_trafotask_survregr

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)
  library(survival)

  # simple example
  pred = PredictionSurv$new(row_ids = 1:10, truth = Surv(1:10, rbinom(10, 1, 0.5)),
    response = 1:10)
  po = po("trafopred_survregr")
  new_pred = po$predict(list(pred = pred))[[1]]
  print(new_pred)

## End(Not run)
```

mlr_pipeops_trafotask_regrsurv *PipeOpTaskRegrSurv*

Description

Transform TaskRegr to TaskSurv.

Input and Output Channels

Input and output channels are inherited from PipeOpTaskTransformer.

The output is the input TaskRegr transformed to a TaskSurv.

State

The `$state` is a named `list` with the `$state` elements inherited from PipeOpTaskTransformer.

Parameters

The parameters are

- `status :: (numeric(1))`
  If `NULL` then assumed no censoring in the dataset. Otherwise should be a vector of `0/1`s of same length as the prediction object, where `1` is dead and `0` censored.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3proba::PipeOpTransformer` -> `mlr3proba::PipeOpTaskTransformer` -> `PipeOpTaskRegrSurv`

Methods

Public methods:

- `PipeOpTaskRegrSurv$new()`
- `PipeOpTaskRegrSurv$clone()`

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

Usage:

```
PipeOpTaskRegrSurv$new(id = "trafotask_regrsurv")
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: PipeOpPredTransformer, PipeOpTaskTransformer, PipeOpTransformer, mlr_pipeops_survavg, mlr_pipeops_trafopred_classifsurv_IPCW, mlr_pipeops_trafopred_classifsurv_disctime, mlr_pipeops_trafopred_regrsurv, mlr_pipeops_trafopred_survregr, mlr_pipeops_trafotask_survclassif_IPCW mlr_pipeops_trafotask_survclassif_disctime, mlr_pipeops_trafotask_survregr

Other Transformation PipeOps: mlr_pipeops_trafopred_classifsurv_IPCW, mlr_pipeops_trafopred_classifsurv_c mlr_pipeops_trafopred_regrsurv, mlr_pipeops_trafopred_survregr, mlr_pipeops_trafotask_survclassif_IPCW mlr_pipeops_trafotask_survclassif_disctime, mlr_pipeops_trafotask_survregr

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  task = tsk("boston_housing")
  po = po("trafotask_regrsurv")

  # assume no censoring
  new_task = po$train(list(task_regr = task, task_surv = NULL))[[1]]
  print(new_task)

  # add censoring
  task_surv = tsk("rats")
  task_regr = po("trafotask_survregr", method = "omit")$train(list(task_surv, NULL))[[1]]
  print(task_regr)
  new_task = po$train(list(task_regr = task_regr, task_surv = task_surv))[[1]]
  new_task$truth()
  task_surv$truth()

## End(Not run)
```

mlr_pipeops_trafotask_survclassif_disctime

PipeOpTaskSurvClassifDiscTime

Description

Transform TaskSurv to TaskClassif by dividing continuous time into multiple time intervals for each observation. This transformation creates a new target variable `disc_status` that indicates whether an event occurred within each time interval. This approach facilitates survival analysis within a classification framework using discrete time intervals (Tutz et al. 2016).

Dictionary

This PipeOp can be instantiated via the dictionary mlr3pipelines::mlr_pipeops or with the associated sugar function `mlr3pipelines::po()`:

```
PipeOpTaskSurvClassifDiscTime$new()
mlr_pipeops$get("trafotask_survclassif_disctime")
po("trafotask_survclassif_disctime")
```

Input and Output Channels

PipeOpTaskSurvClassifDiscTime has one input channel named "input", and two output channels, one named "output" and the other "transformed_data".

During training, the "output" is the "input" TaskSurv transformed to a TaskClassif. The target column is named `"disc_status"` and indicates whether an event occurred in each time interval. An additional feature named `"tend"` contains the end time point of each interval. Lastly, the "output" task has a column with the original observation ids, under the role `"original_ids"`. The "transformed_data" is an empty data.table.

During prediction, the "input" TaskSurv is transformed to the "output" TaskClassif with `"disc_status"` as target and the `"tend"` feature included. The "transformed_data" is a data.table with columns the `"disc_status"` target of the "output" task, the `"id"` (original observation ids), `"obs_times"` (observed times per `"id"`) and `"tend"` (end time of each interval). This "transformed_data" is only meant to be used with the PipeOpPredClassifSurvDiscTime.

State

The `$state` contains information about the `cut` parameter used.

Parameters

The parameters are

- `cut :: numeric()`
  Split points, used to partition the data into intervals based on the `time` column. If unspecified, all unique event times will be used. If `cut` is a single integer, it will be interpreted as the number of equidistant intervals from 0 until the maximum event time.
- `max_time :: numeric(1)`
  If `cut` is unspecified, this will be the last possible event time. All event times after `max_time` will be administratively censored at `max_time.` Needs to be greater than the minimum event time in the given task.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpTaskSurvClassifDiscTime`

Methods

Public methods:

- `PipeOpTaskSurvClassifDiscTime$new()`
- `PipeOpTaskSurvClassifDiscTime$clone()`

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

Usage:

```
PipeOpTaskSurvClassifDiscTime$new(id = "trafotask_survclassif_disctime")
```

Arguments:

`id` (`character(1)`)
: Identifier of the resulting object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Tutz, Gerhard, Schmid, Matthias (2016). *Modeling Discrete Time-to-Event Data*, series Springer Series in Statistics. Springer International Publishing. ISBN 978-3-319-28156-8 978-3-319-28158-2, http://link.springer.com/10.1007/978-3-319-28158-2.

See Also

Other PipeOps: `PipeOpPredTransformer`, `PipeOpTaskTransformer`, `PipeOpTransformer`, `mlr_pipeops_survavg`, `mlr_pipeops_trafopred_classifsurv_IPCW`, `mlr_pipeops_trafopred_classifsurv_disctime`, `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_survregr`, `mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_IPCW`, `mlr_pipeops_trafotask_survregr`

Other Transformation PipeOps: `mlr_pipeops_trafopred_classifsurv_IPCW`, `mlr_pipeops_trafopred_classifsurv_d` `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_survregr`, `mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_IPCW`, `mlr_pipeops_trafotask_survregr`

Examples

```
## Not run:
  library(mlr3)
  library(mlr3learners)
  library(mlr3pipelines)

  task = tsk("lung")

  # transform the survival task to a classification task
  # all unique event times are used as cutpoints
  po_disc = po("trafotask_survclassif_disctime")
  task_classif = po_disc$train(list(task))[[1L]]

  # the end time points of the discrete time intervals
  unique(task_classif$data(cols = "tend"))[[1L]]

  # train a classification learner
  learner = lrn("classif.log_reg", predict_type = "prob")
  learner$train(task_classif)

## End(Not run)
```

`mlr_pipeops_trafotask_survclassif_IPCW` *PipeOpTaskSurvClassifIPCW*

Description

Transform TaskSurv to TaskClassif using the **I**nverse **P**robability of **C**ensoring **W**eights (IPCW) method by Vock et al. (2016).

Let T_i be the observed times (event or censoring) and δ_i the censoring indicators for each observation i in the training set. The IPCW technique consists of two steps: first we estimate the censoring distribution $\hat{G}(t)$ using the Kaplan-Meier estimator from the training data. Then we calculate the observation weights given a cutoff time τ as:

$$\omega_i = 1/\hat{G}(min(T_i, \tau))$$

Observations that are censored prior to τ are assigned zero weights, i.e. $\omega_i = 0$.

Dictionary

This PipeOp can be instantiated via the dictionary mlr3pipelines::mlr_pipeops or with the associated sugar function `mlr3pipelines::po()`:

```
PipeOpTaskSurvClassifIPCW$new()
mlr_pipeops$get("trafotask_survclassif_IPCW")
po("trafotask_survclassif_IPCW")
```

Input and Output Channels

PipeOpTaskSurvClassifIPCW has one input channel named "input", and two output channels, one named "output" and the other "data".

Training transforms the "input" TaskSurv to a TaskClassif, which is the "output". The target column is named `"status"` and indicates whether **an event occurred before the cutoff time** τ (`1` = yes, `0` = no). The observed times column is removed from the "output" task. The transformed task has the property `"weights"` (the ω_i). The "data" is `NULL`.

During prediction, the "input" TaskSurv is transformed to the "output" TaskClassif with `"status"` as target (again indicating if the event occurred before the cutoff time). The "data" is a data.table containing the observed `times` T_i and censoring indicators/`status` δ_i of each subject as well as the corresponding `row_ids`. This "data" is only meant to be used with the PipeOpPredClassifSurvIPCW.

Parameters

The parameters are

- `tau` :: `numeric()`
  Predefined time point for IPCW. Observations with time larger than τ are censored. Must be less or equal to the maximum event time.

- `eps` :: `numeric()`
  Small value to replace $G(t) = 0$ censoring probabilities to prevent infinite weights (a warning is triggered if this happens).

Super class

`mlr3pipelines::PipeOp` -> `PipeOpTaskSurvClassifIPCW`

Methods

Public methods:

- `PipeOpTaskSurvClassifIPCW$new()`
- `PipeOpTaskSurvClassifIPCW$clone()`

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

Usage:

```
PipeOpTaskSurvClassifIPCW$new(id = "trafotask_survclassif_IPCW")
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Vock, M D, Wolfson, Julian, Bandyopadhyay, Sunayan, Adomavicius, Gediminas, Johnson, E P, Vazquez-Benitez, Gabriela, O'Connor, J P (2016). "Adapting machine learning techniques to censored time-to-event health record data: A general-purpose approach using inverse probability of censoring weighting." *Journal of Biomedical Informatics*, **61**, 119–131. doi:10.1016/j.jbi.2016.03.009, `https://www.sciencedirect.com/science/article/pii/S1532046416000496`.

See Also

Other PipeOps: `PipeOpPredTransformer`, `PipeOpTaskTransformer`, `PipeOpTransformer`, `mlr_pipeops_survavg`, `mlr_pipeops_trafopred_classifsurv_IPCW`, `mlr_pipeops_trafopred_classifsurv_disctime`, `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_survregr`, `mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_disctime`, `mlr_pipeops_trafotask_survregr`

Other Transformation PipeOps: `mlr_pipeops_trafopred_classifsurv_IPCW`, `mlr_pipeops_trafopred_classifsurv_d` `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_survregr`, `mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_disctime`, `mlr_pipeops_trafotask_survregr`

Examples

```
## Not run:
  library(mlr3)
  library(mlr3learners)
  library(mlr3pipelines)

  task = tsk("lung")

  # split task to train and test subtasks
  part = partition(task)
  task_train = task$clone()$filter(part$train)
  task_test = task$clone()$filter(part$test)

  # define IPCW pipeop
  po_ipcw = po("trafotask_survclassif_IPCW", tau = 365)

  # during training, output is a classification task with weights
  task_classif_train = po_ipcw$train(list(task_train))[[1]]
  task_classif_train

  # during prediction, output is a classification task (no weights)
  task_classif_test = po_ipcw$predict(list(task_test))[[1]]
  task_classif_test

  # train classif learner on the train task with weights
  learner = lrn("classif.rpart", predict_type = "prob")
  learner$train(task_classif_train)

  # predict using the test output task
  p = learner$predict(task_classif_test)

  # use classif measures for evaluation
  p$confusion
  p$score()
  p$score(msr("classif.auc"))

## End(Not run)
```

mlr_pipeops_trafotask_survregr *PipeOpTaskSurvRegr*

Description

Transform TaskSurv to TaskRegr.

Input and Output Channels

Input and output channels are inherited from PipeOpTaskTransformer.

The output is the input TaskSurv transformed to a TaskRegr.

State

The `$state` is a named `list` with the `$state` elements

- `instatus`: Censoring status from input training task.
- `outstatus` : Censoring status from input prediction task.

Parameters

The parameters are

- `method` :: `character(1)`
  Method to use for dealing with censoring. Options are `"ipcw"` (Vock et al., 2016): censoring column is removed and a `weights` column is added, weights are inverse estimated survival probability of the censoring distribution evaluated at survival time; `"mrl"` (Klein and Moeschberger, 2003): survival time of censored observations is transformed to the observed time plus the mean residual life-time at the moment of censoring; `"bj"` (Buckley and James, 1979): Buckley-James imputation assuming an AFT model form, calls bujar::bujar; `"delete"`: censored observations are deleted from the data-set - should be used with caution if censoring is informative; `"omit"`: the censoring status column is deleted - again should be used with caution; `"reorder"`: selects features and targets and sets the target in the new task object. Note that `"mrl"` and `"ipcw"` will perform worse with Type I censoring.
- `estimator` :: `character(1)`
  Method for calculating censoring weights or mean residual lifetime in `"mrl"`, current options are: `"kaplan"`: unconditional Kaplan-Meier estimator; `"akritas"`: conditional non-parameteric nearest-neighbours estimator; `"cox"`.
- `alpha` :: `numeric(1)`
  When `ipcw` is used, optional hyper-parameter that adds an extra penalty to the weighting for censored observations. If set to `0` then censored observations are given zero weight and deleted, weighting only the non-censored observations. A weight for an observation is then $(\delta + \alpha(1 - \delta))/G(t)$ where δ is the censoring indicator.
- `eps` :: `numeric(1)`
  Small value to replace `0` survival probabilities with in IPCW to prevent infinite weights.
- `lambda` :: `numeric(1)`
  Nearest neighbours parameter for the `"akritas"` estimator in the mlr3extralearners package, default `0.5`.
- `features, target` :: `character()`
  For `"reorder"` method, specify which columns become features and targets.
- `learner cneter, mimpu, iter.bj, max.cycle, mstop, nu`
  Passed to bujar::bujar.

Super classes

mlr3pipelines::PipeOp -> mlr3proba::PipeOpTransformer -> mlr3proba::PipeOpTaskTransformer -> PipeOpTaskSurvRegr

Methods

Public methods:

- PipeOpTaskSurvRegr$new()
- PipeOpTaskSurvRegr$clone()

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

Usage:

```
PipeOpTaskSurvRegr$new(id = "trafotask_survregr", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Buckley, Jonathan, James, Ian (1979). "Linear Regression with Censored Data." *Biometrika*, **66**(3), 429–436. doi:10.2307/2335161, https://www.jstor.org/stable/2335161.

Klein, P J, Moeschberger, L M (2003). *Survival analysis: techniques for censored and truncated data*, 2 edition. Springer Science & Business Media. ISBN 0387216456.

Vock, M D, Wolfson, Julian, Bandyopadhyay, Sunayan, Adomavicius, Gediminas, Johnson, E P, Vazquez-Benitez, Gabriela, O'Connor, J P (2016). "Adapting machine learning techniques to censored time-to-event health record data: A general-purpose approach using inverse probability of censoring weighting." *Journal of Biomedical Informatics*, **61**, 119–131. doi:10.1016/j.jbi.2016.03.009, https://www.sciencedirect.com/science/article/pii/S1532046416000496.

See Also

Other PipeOps: PipeOpPredTransformer, PipeOpTaskTransformer, PipeOpTransformer, mlr_pipeops_survavg, mlr_pipeops_trafopred_classifsurv_IPCW, mlr_pipeops_trafopred_classifsurv_disctime, mlr_pipeops_trafopred_regrsurv, mlr_pipeops_trafopred_survregr, mlr_pipeops_trafotask_regrsurv, mlr_pipeops_trafotask_survclassif_IPCW, mlr_pipeops_trafotask_survclassif_disctime

Other Transformation PipeOps: mlr_pipeops_trafopred_classifsurv_IPCW, mlr_pipeops_trafopred_classifsurv_c
mlr_pipeops_trafopred_regrsurv, mlr_pipeops_trafopred_survregr, mlr_pipeops_trafotask_regrsurv, mlr_pipeops_trafotask_survclassif_IPCW, mlr_pipeops_trafotask_survclassif_disctime

Examples

```
## Not run:
  library(mlr3)
  library(mlr3pipelines)

  # these methods are generally only successful if censoring is not too high
  # create survival task by undersampling
  task = tsk("rats")$filter(
    c(which(tsk("rats")$truth()[, 2] == 1),
      sample(which(tsk("rats")$truth()[, 2] == 0), 42))
  )

  # deletion
  po = po("trafotask_survregr", method = "delete")
  po$train(list(task, NULL))[[1]] # 42 deleted

  # omission
  po = po("trafotask_survregr", method = "omit")
  po$train(list(task, NULL))[[1]]

  if (requireNamespace("mlr3extralearners", quietly = TRUE)) {
    # ipcw with Akritas
   po = po("trafotask_survregr", method = "ipcw", estimator = "akritas", lambda = 0.4, alpha = 0)
    new_task = po$train(list(task, NULL))[[1]]
    print(new_task)
    new_task$weights
  }

  # mrl with Kaplan-Meier
  po = po("trafotask_survregr", method = "mrl")
  new_task = po$train(list(task, NULL))[[1]]
  data.frame(new = new_task$truth(), old = task$truth())

  # Buckley-James imputation
  if (requireNamespace("bujar", quietly = TRUE)) {
    po = po("trafotask_survregr", method = "bj")
    new_task = po$train(list(task, NULL))[[1]]
    data.frame(new = new_task$truth(), old = task$truth())
  }

  # reorder - in practice this will be only be used in a few graphs
 po = po("trafotask_survregr", method = "reorder", features = c("sex", "rx", "time", "status"),
    target = "litter")
  new_task = po$train(list(task, NULL))[[1]]
  print(new_task)

  # reorder using another task for feature names
  po = po("trafotask_survregr", method = "reorder", target = "litter")
  new_task = po$train(list(task, task))[[1]]
  print(new_task)

## End(Not run)
```

mlr_tasks_actg *ACTG 320 Survival Task*

Description

A survival task for the actg data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("actg")
tsk("actg")
```

Meta Information

- Task type: "surv"
- Dimensions: 1151x13
- Properties: -
- Has Missings: `FALSE`
- Target: "time", "status"
- Features: "age", "cd4", "hemophil", "ivdrug", "karnof", "priorzdv", "raceth", "sexF", "strat2", "tx", "txgrp"

Pre-processing

- Column `sex` has been renamed to `sexF` and `censor` has been renamed to `status`.
- Columns `id`, `time_d`, and `censor_d` have been removed so target is `time` to AIDS diagnosis (in days).

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html`
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

mlr_tasks_faithful *Old Faithful Eruptions Density Task*

Description

A density task for the faithful data set.

Format

R6::R6Class inheriting from TaskDens.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("faithful")
tsk("faithful")
```

Meta Information

- Task type: “dens”
- Dimensions: 272x1
- Properties: -
- Has Missings: `FALSE`
- Target: -
- Features: “eruptions”

Preprocessing

- Only the `eruptions` column is kept in this task.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html`
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

mlr_tasks_gbcs *German Breast Cancer Study Survival Task*

Description

A survival task for the gbcs data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("gbcs")
tsk("gbcs")
```

Meta Information

- Task type: “surv”
- Dimensions: 686x10
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “age”, “estrg_recp”, “grade”, “hormone”, “menopause”, “nodes”, “prog_recp”, “size”

Preprocessing

- Column `id` and all date columns have been removed, as well as `rectime` and `censrec`.
- Target columns (`survtime`, `censdead`) have been renamed to (`time`, `status`).

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html`
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

mlr_tasks_gbsg *German Breast Cancer Study Survival Task*

Description

A survival task for the gbsg data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("gbsg")
tsk("gbsg")
```

Meta Information

- Task type: “surv”
- Dimensions: 686x10
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “age”, “er”, “grade”, “hormon”, “meno”, “nodes”, “pgr”, “size”

Pre-processing

- Removed column `pid`.
- Column `meno` has been converted to `factor` and 0/1 values have been replaced with `premenopausal` and `postmenopausal` respectively.
- Column `hormon` has been converted to `factor` and 0/1 values have been replaced with `no` and `yes` respectively.
- Column `grade` has been converted to `factor`.
- Renamed target column `rfstime` to `time`.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

`mlr_tasks_grace` *GRACE 1000 Survival Task*

Description

A survival task for the grace data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("grace")
tsk("grace")
```

Meta Information

- Task type: “surv”
- Dimensions: 1000x8
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “age”, “los”, “revasc”, “revascdays”, “stchange”, “sysbp”

Preprocessing

- Column `id` is removed.
- Target columns (`days`, `death`) have been renamed to (`time`, `status`).

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

`mlr_tasks_lung` *Lung Cancer Survival Task*

Description

A survival task for the lung data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("lung")
tsk("lung")
```

Meta Information

- Task type: “surv”
- Dimensions: 168x9
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “age”, “meal.cal”, “pat.karno”, “ph.ecog”, “ph.karno”, “sex”, “wt.loss”

Pre-processing

- Column `inst` has been removed.
- Column `sex` has been converted to a `factor`, all others have been converted to `integer`.
- Kept only complete cases (no missing values).

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

`mlr_tasks_mgus` *Monoclonal Gammopathy Survival Task*

Description

A survival task for the mgus data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("mgus")
tsk("mgus")
```

Meta Information

- Task type: “surv”
- Dimensions: 176x9
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “age”, “alb”, “creat”, “dxyr”, “hgb”, “mspike”, “sex”

Pre-processing

- Removed columns `id`, `pcdx` and `pctime`.
- Renamed target columns from (`fultime`, `death`) to (`time`, `status`).
- Kept only complete cases (no missing values).

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

`mlr_tasks_pbc` *Primary Biliary Cholangitis Survival Task*

Description

A survival task for the pbc data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("pbc")
tsk("pbc")
```

Meta Information

- Task type: “surv”
- Dimensions: 276x19
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “age”, “albumin”, “alk.phos”, “ascites”, “ast”, “bili”, “chol”, “copper”, “edema”, “hepato”, “platelet”, “protime”, “sex”, “spiders”, “stage”, “trig”, “trt”

Pre-processing

- Removed column `id`.
- Kept only complete cases (no missing values).
- Column `age` has been converted to `integer`.
- Columns `trt`, `stage`, `hepato`, `edema` and `ascites` have been converted to `factor`s.
- Column `trt` has levels `Dpenicillmain` and `placebo` instead of 1 and 2.
- Column `status` has 1 for death and 0 for censored or transplant.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html`
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

`mlr_tasks_precip` *Annual Precipitation Density Task*

Description

A density task for the precip data set.

Format

R6::R6Class inheriting from TaskDens.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("precip")
tsk("precip")
```

Meta Information

- Task type: “dens”
- Dimensions: 70x1
- Properties: -
- Has Missings: `FALSE`
- Target: -
- Features: “precip”

Preprocessing

- Only the `precip` column is kept in this task.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

`mlr_tasks_rats` *Rats Survival Task*

Description

A survival task for the rats data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("rats")
tsk("rats")
```

Meta Information

- Task type: “surv”
- Dimensions: 300x5
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “litter”, “rx”, “sex”

Pre-processing

- Column `sex` has been converted to a `factor`, all others have been converted to `integer`.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html`
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

`mlr_tasks_unemployment`

Unemployment Duration Survival Task

Description

A survival task for the Ecdat::UnempDur data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("unemployment")
tsk("unemployment")
```

Meta Information

- Task type: “surv”
- Dimensions: 3343x6
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “age”, “logwage”, “tenure”, “ui”

Preprocessing

- Only the columns `spell`, `censor1`, `age`, `logwage`, `tenure`, `ui` are kept in this task.
- Renamed target columns from (`spell`, `censor1`) to (`time`, `status`), so outcome is the duration until re-employment in a full-time job.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html`
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_veteran`, `mlr_tasks_whas`

`mlr_tasks_veteran` *Veteran Survival Task*

Description

A survival task for the veteran data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("veteran")
tsk("veteran")
```

Meta Information

- Task type: “surv”
- Dimensions: 137x8
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “age”, “celltype”, “diagtime”, “karno”, “prior”, “trt”

Pre-processing

- Columns `age`, `time`, `status`, `diagtime` and `karno` have been converted to `integer`.
- Columns `trt`, `prior` have been converted to `factors`. Prior therapy values are `no`/`yes` instead of 0/10.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html`
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_whas`

`mlr_tasks_whas` *Worcester Heart Attack Study (WHAS) Survival Task*

Description

A survival task for the whas data set.

Format

R6::R6Class inheriting from TaskSurv.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function `tsk()`:

```
mlr_tasks$get("whas")
tsk("whas")
```

Meta Information

- Task type: “surv”
- Dimensions: 481x11
- Properties: -
- Has Missings: `FALSE`
- Target: “time”, “status”
- Features: “age”, “chf”, “cpk”, “lenstay”, “miord”, “mitype”, “sexF”, “sho”, “year”

Preprocessing

- Columns `id`, `yrgrp`, and `dstat` are removed.
- Column `sex` is renamed to `sexF`, `lenfol` to `time`, and `fstat` to `status`.
- Target is total follow-up time from hospital admission.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session

Other Task: `TaskDens`, `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`

mlr_task_generators_coxed

Survival Task Generator for Package 'coxed'

Description

A mlr3::TaskGenerator calling `coxed::sim.survdata()`.

This generator creates a survival dataset using **coxed**, and exposes some parameters from the `sim.survdata()` function. We don't include the parameters `X` (user-specified variables), `covariate`, `low`, `high`, `compare`, `beta` and `hazard.fun` for this generator. The latter means that no user-specified hazard function can be used and the generated datasets always use the *flexible-hazard* method from the package.

Dictionary

This TaskGenerator can be instantiated via the dictionary mlr_task_generators or with the associated sugar function `tgen()`:

```
mlr_task_generators$get("coxed")
tgen("coxed")
```

Parameters

Id	Type	Default	Levels	Range
T	numeric	100		$[1, \infty)$
type	character	none	none, tvc, tvbeta	-
knots	integer	8		$[1, \infty)$
spline	logical	TRUE	TRUE, FALSE	-
xvars	integer	3		$[1, \infty)$
mu	untyped	0		-
sd	untyped	0.5		-
censor	numeric	0.1		$[0, 1]$
censor.cond	logical	FALSE	TRUE, FALSE	-

Super class

`mlr3::TaskGenerator` -> `TaskGeneratorCoxed`

Methods

Public methods:

- `TaskGeneratorCoxed$new()`
- `TaskGeneratorCoxed$help()`
- `TaskGeneratorCoxed$clone()`

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

Usage:

```
TaskGeneratorCoxed$new()
```

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

Usage:

```
TaskGeneratorCoxed$help()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Harden, J. J, Kropko, Jonathan (2019). "Simulating Duration Data for the Cox Model." *Political Science Research and Methods*, **7**(4), 921–928. doi:10.1017/PSRM.2018.19.

See Also

- Dictionary of TaskGenerators: mlr3::mlr_task_generators
- `as.data.table(mlr_task_generators)` for a table of available TaskGenerators in the running session

Other TaskGenerator: `mlr_task_generators_simdens`, `mlr_task_generators_simsurv`

Examples

```
library(mlr3)

# time horizon = 365 days, censoring proportion = 60%, 6 covariates normally
# distributed with mean = 1 and sd = 2, independent censoring, no time-varying
# effects
gen = tgen("coxed", T = 365, type = "none", censor = 0.6, xvars = 6,
           mu = 1, sd = 2, censor.cond = FALSE)
gen$generate(50)
```

```
# same as above, but with time-varying coefficients (counting process format)
gen$param_set$set_values(type = "tvc")
gen$generate(50)
```

mlr_task_generators_simdens

Density Task Generator for Package 'distr6'

Description

A mlr3::TaskGenerator calling `distr6::distrSimulate()`. See `distr6::distrSimulate()` for an explanation of the hyperparameters. See `distr6::listDistributions()` for the names of the available distributions.

Dictionary

This TaskGenerator can be instantiated via the dictionary mlr_task_generators or with the associated sugar function `tgen()`:

```
mlr_task_generators$get("simdens")
tgen("simdens")
```

Parameters

Id	Type	Default	Levels
distribution	character	Normal	Arcsine, Arrdist, Bernoulli, Beta, BetaNoncentral, Binomial, Categorical, Cauchy, ChiSq
pars	untyped	-	

Super class

`mlr3::TaskGenerator` -> `TaskGeneratorSimdens`

Methods

Public methods:

- `TaskGeneratorSimdens$new()`
- `TaskGeneratorSimdens$help()`
- `TaskGeneratorSimdens$clone()`

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

Usage:

```
TaskGeneratorSimdens$new()
```

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

Usage:

```
TaskGeneratorSimdens$help()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

- Dictionary of TaskGenerators: mlr3::mlr_task_generators
- `as.data.table(mlr_task_generators)` for a table of available TaskGenerators in the running session

Other TaskGenerator: `mlr_task_generators_coxed`, `mlr_task_generators_simsurv`

Examples

```
# generate 20 samples from a standard Normal distribution
dens_gen = tgen("simdens")
dens_gen$param_set

task = dens_gen$generate(20)
head(task)

# generate 50 samples from a Binomial distribution with specific parameters
dens_gen = tgen("simdens", distribution = "Bernoulli", pars = list(prob = 0.8))
task = dens_gen$generate(50)
task$data()[["x"]]
```

`mlr_task_generators_simsurv` *Survival Task Generator for Package 'simsurv'*

Description

A mlr3::TaskGenerator calling `simsurv::simsurv()` from package **simsurv**.

This generator currently only exposes a small subset of the flexibility of **simsurv**, and just creates a small dataset with the following numerical covariates:

- `treatment`: Bernoulli distributed with hazard ratio `0.5`.
- `height`: Normally distributed with hazard ratio `1`.
- `weight`: normally distributed with hazard ratio `1`.

See `simsurv::simsurv()` for an explanation of the hyperparameters. Initial values for hyperparameters are `lambdas` = 0.1, `gammas` = 1.5 and `maxt` = 5. The last one, by default generates samples which are administratively censored at $\tau = 5$, so increase this value if you want to change this.

Dictionary

This TaskGenerator can be instantiated via the dictionary mlr_task_generators or with the associated sugar function `tgen()`:

```
mlr_task_generators$get("simsurv")
tgen("simsurv")
```

Parameters

Id	Type	Default	Levels	Range
dist	character	weibull	weibull, exponential, gompertz	-
lambdas	numeric	-		$[0, \infty)$
gammas	numeric	-		$[0, \infty)$
maxt	numeric	-		$[0, \infty)$

Super class

`mlr3::TaskGenerator` -> `TaskGeneratorSimsurv`

Methods

Public methods:

- `TaskGeneratorSimsurv$new()`
- `TaskGeneratorSimsurv$help()`
- `TaskGeneratorSimsurv$clone()`

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

Usage:

```
TaskGeneratorSimsurv$new()
```

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

Usage:

```
TaskGeneratorSimsurv$help()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Brilleman, L. S, Wolfe, Rory, Moreno-Betancur, Margarita, Crowther, J. M (2021). "Simulating Survival Data Using the simsurv R Package." *Journal of Statistical Software*, **97**(3), 1–27. doi:10.18637/JSS.V097.I03.

See Also

- Dictionary of TaskGenerators: mlr3::mlr_task_generators
- `as.data.table(mlr_task_generators)` for a table of available TaskGenerators in the running session

Other TaskGenerator: `mlr_task_generators_coxed`, `mlr_task_generators_simdens`

Examples

```
# generate 20 samples with Weibull survival distribution
gen = tgen("simsurv")
task = gen$generate(20)
head(task)

# generate 100 samples with exponential survival distribution and tau = 40
gen = tgen("simsurv", dist = "exponential", gammas = NULL, maxt = 40)
task = gen$generate(100)
head(task)
```

`pecs` *Prediction Error Curves for PredictionSurv and LearnerSurv*

Description

Methods to plot prediction error curves (pecs) for either a PredictionSurv object or a list of trained LearnerSurvs.

Usage

```
pecs(x, measure = c("graf", "logloss"), times, n, eps = NULL, ...)

## S3 method for class 'list'
pecs(
  x,
  measure = c("graf", "logloss"),
  times,
  n,
  eps = NULL,
  task = NULL,
  row_ids = NULL,
  newdata = NULL,
  train_task = NULL,
  train_set = NULL,
  proper = TRUE,
  ...
)
```

```
## S3 method for class 'PredictionSurv'
pecs(
  x,
  measure = c("graf", "logloss"),
  times,
  n,
  eps = 1e-15,
  train_task = NULL,
  train_set = NULL,
  proper = TRUE,
  ...
)
```

Arguments

`x` (PredictionSurv or `list` of LearnerSurvs)

`measure` (`character(1)`)
Either `"graf"` for MeasureSurvGraf, or `"logloss"` for MeasureSurvIntLogloss

`times` (`numeric()`)
If provided then either a vector of time-points to evaluate `measure` or a range of time-points.

`n` (`integer()`)
If `times` is missing or given as a range, then `n` provide number of time-points to evaluate `measure` over.

`eps` (`numeric()`)
Small error value to prevent errors resulting from a log(0) or 1/0 calculation. Default is 1e-15 for log loss and 1e-3 for Graf.

`...` Additional arguments.

`task` (TaskSurv)

`row_ids` (`integer()`)
Passed to `Learner$predict`.

`newdata` (`data.frame()`)
If not missing `Learner$predict_newdata` is called instead of `Learner$predict`.

`train_task` (TaskSurv)
If not NULL then passed to measures for computing estimate of censoring distribution on training data.

`train_set` (`numeric()`)
If not NULL then passed to measures for computing estimate of censoring distribution on training data.

`proper` (`logical(1)`)
Passed to MeasureSurvGraf or MeasureSurvIntLogloss.

Details

If `times` and `n` are missing then `measure` is evaluated over all observed time-points from the PredictionSurv or TaskSurv object. If a range is provided for `times` without `n`, then all time-points between the range are returned.

Examples

```
## Not run:
  #' library(mlr3)
  task = tsk("rats")

  # Prediction Error Curves for prediction object
  learn = lrn("surv.coxph")
  p = learn$train(task)$predict(task)
  pecs(p)
  pecs(p, measure = "logloss", times = c(20, 40, 60, 80)) +
    ggplot2::geom_point() +
    ggplot2::ggtitle("Logloss Prediction Error Curve for Cox PH")

  # Access underlying data
  x = pecs(p)
  x$data

  # Prediction Error Curves for fitted learners
  learns = lrns(c("surv.kaplan", "surv.coxph"))
  lapply(learns, function(x) x$train(task))
  pecs(learns, task = task, measure = "logloss", times = c(20, 90), n = 10)

## End(Not run)
```

PipeOpPredTransformer *PipeOpPredTransformer*

Description

Parent class for PipeOps that transform Prediction objects to different types.

Input and Output Channels

`PipeOpPredTransformer` has one input and output channel named `"input"` and `"output"`. In training and testing these expect and produce mlr3::Prediction objects with the type depending on the transformers.

State

The `$state` is a named `list` with the `$state` elements

- `inpredtypes`: Predict types in the input prediction object during training.
- `outpredtypes` : Predict types in the input prediction object during prediction, checked against `inpredtypes`.

Internals

Classes inheriting from `PipeOpPredTransformer` transform Prediction objects from one class (e.g. regr, classif) to another.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3proba::PipeOpTransformer` -> `PipeOpPredTransformer`

Methods

Public methods:

- `PipeOpPredTransformer$new()`
- `PipeOpPredTransformer$clone()`

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

Usage:

```
PipeOpPredTransformer$new(
  id,
  param_set = ps(),
  param_vals = list(),
  packages = character(0),
  input = data.table(),
  output = data.table()
)
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

`packages` (`character()`)
Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.

`input` data.table::data.table
`data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`). Sets the `$input` slot, see PipeOp.

`output` data.table::data.table
`data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`). Sets the `$output` slot, see PipeOp.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: PipeOpTaskTransformer, PipeOpTransformer, mlr_pipeops_survavg, mlr_pipeops_trafopred_class mlr_pipeops_trafopred_classifsurv_disctime, mlr_pipeops_trafopred_regrsurv, mlr_pipeops_trafopred_surv mlr_pipeops_trafotask_regrsurv, mlr_pipeops_trafotask_survclassif_IPCW, mlr_pipeops_trafotask_survclas mlr_pipeops_trafotask_survregr

Other Transformers: PipeOpTaskTransformer, PipeOpTransformer

PipeOpTaskTransformer *PipeOpTaskTransformer*

Description

Parent class for PipeOps that transform task objects to different types.

Input and Output Channels

`PipeOpTaskTransformer` has one input and output channel named `"input"` and `"output"`. In training and testing these expect and produce mlr3::Task objects with the type depending on the transformers.

State

The `$state` is left empty (`list()`).

Internals

The commonality of methods using `PipeOpTaskTransformer` is that they take a mlr3::Task of one class and transform it to another class. This usually involves transformation of the data, which can be controlled via parameters.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3proba::PipeOpTransformer` -> `PipeOpTaskTransformer`

Methods

Public methods:

- `PipeOpTaskTransformer$new()`
- `PipeOpTaskTransformer$clone()`

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

Usage:

```
PipeOpTaskTransformer$new(
  id,
  param_set = ps(),
  param_vals = list(),
  packages = character(0),
  input,
  output
)
```

Arguments:

`id` (`character(1)`)
    Identifier of the resulting object.

`param_set` (paradox::ParamSet)
    Set of hyperparameters.

`param_vals` (`list()`)
    List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

`packages` (`character()`)
    Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.

`input` data.table::data.table
    `data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`). Sets the `$input` slot, see PipeOp.

`output` data.table::data.table
    `data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`). Sets the `$output` slot, see PipeOp.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `PipeOpPredTransformer`, `PipeOpTransformer`, `mlr_pipeops_survavg`, `mlr_pipeops_trafopred_class`
`mlr_pipeops_trafopred_classifsurv_disctime`, `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_surv`
`mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_IPCW`, `mlr_pipeops_trafotask_survcla`
`mlr_pipeops_trafotask_survregr`

Other Transformers: `PipeOpPredTransformer`, `PipeOpTransformer`

`PipeOpTransformer` *PipeOpTransformer*

Description

Parent class for PipeOps that transform Task and Prediction objects to different types.

Input and Output Channels

Determined by child classes.

State

The `$state` is left empty (`list()`).

Internals

The commonality of methods using PipeOpTransformer is that they take a Task or Prediction of one type (e.g. regr or classif) and transform it to another type.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpTransformer`

Methods

Public methods:

- `PipeOpTransformer$new()`
- `PipeOpTransformer$clone()`

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

Usage:

```
PipeOpTransformer$new(
  id,
  param_set = ps(),
  param_vals = list(),
  packages = character(),
  input = data.table(),
  output = data.table()
)
```

Arguments:

`id` (`character(1)`)
: Identifier of the resulting object.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`param_vals` (`list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

`packages` (`character()`)
: Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.

`input` data.table::data.table
: `data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`). Sets the `$input` slot, see PipeOp.

`output` data.table::data.table
: `data.table` with columns `name` (`character`), `train` (`character`), `predict` (`character`). Sets the `$output` slot, see PipeOp.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `PipeOpPredTransformer`, `PipeOpTaskTransformer`, `mlr_pipeops_survavg`, `mlr_pipeops_trafopred_c`
`mlr_pipeops_trafopred_classifsurv_disctime`, `mlr_pipeops_trafopred_regrsurv`, `mlr_pipeops_trafopred_sur`
`mlr_pipeops_trafotask_regrsurv`, `mlr_pipeops_trafotask_survclassif_IPCW`, `mlr_pipeops_trafotask_survclas`
`mlr_pipeops_trafotask_survregr`

Other Transformers: `PipeOpPredTransformer`, `PipeOpTaskTransformer`

`plot.LearnerSurv` *Visualization of fitted* `LearnerSurv` *objects*

Description

Wrapper around `predict.LearnerSurv` and `plot.Matdist`.

Usage

```
## S3 method for class 'LearnerSurv'
plot(
  x,
  task,
  fun = c("survival", "pdf", "cdf", "quantile", "hazard", "cumhazard"),
  row_ids = NULL,
  newdata,
  ...
)
```

Arguments

`x` (LearnerSurv)

`task` (TaskSurv)

`fun` (`character`)
Passed to `distr6::plot.Matdist`

`row_ids` (`integer()`)
Passed to `Learner$predict`

`newdata` (`data.frame()`)
If not missing `Learner$predict_newdata` is called instead of `Learner$predict`.

`...` Additional arguments passed to `distr6::plot.Matdist`

Examples

```
## Not run:
library(mlr3)
task = tsk("rats")

# Prediction Error Curves for prediction object
learn = lrn("surv.coxph")
learn$train(task)

plot(learn, task, "survival", ind = 10)
plot(learn, task, "survival", row_ids = 1:5)
plot(learn, task, "survival", newdata = task$data()[1:5, ])
plot(learn, task, "survival", newdata = task$data()[1:5, ], ylim = c(0, 1))

## End(Not run)
```

`plot_probregr` *Visualise probabilistic regression distribution predictions*

Description

Plots probability density functions from n predicted probability distributions.

Usage

```
plot_probregr(
  p,
  n,
  type = c("point", "line", "both", "none"),
  which_plot = c("random", "top"),
  rm_zero = TRUE,
  ...
)
```

Arguments

p
(PredictionRegr)
With at least column distr.

n
(integer(1))
Number of predictions to plot.

type
(character(1))
One of "point" (default), "line", "both", "none".

which_plot
(character(1))
One of "random" (default) or "top". See details.

rm_zero
(logical(1))
If TRUE (default) does not plot points where f(x) = 0.

...
Unused

Details

type:

- "point" (default) - Truth plotted as point (truth, predicted_pdf(truth))
- "line" - Truth plotted as vertical line intercepting x-axis at the truth.
- "both" - Plots both the above options.
- "none" - Truth not plotted (default if p$truth is missing).

which_plot:

- "random"(default) - Random selection ofn‘ distributions are plotted.
- "top"- Topn‘ distributions are plotted.

It is unlikely the plot will be interpretable when n >> 5.

Examples

```
## Not run:
library(mlr3verse)
task = tsk("boston_housing")
pipe = as_learner(ppl("probregr", lrn("regr.ranger"), dist = "Normal"))
p = pipe$train(task)$predict(task)
plot_probregr(p, 10, "point", "top")

## End(Not run)
```

`PredictionDens` *Prediction Object for Density*

Description

This object stores the predictions returned by a learner of class LearnerDens.

The `task_type` is set to `"dens"`.

Super class

`mlr3::Prediction` -> `PredictionDens`

Active bindings

`pdf` (`numeric()`)
Access the stored predicted probability density function.

`cdf` (`numeric()`)
Access the stored predicted cumulative distribution function.

`distr` (Distribution)
Access the stored estimated distribution.

Methods

Public methods:

- `PredictionDens$new()`
- `PredictionDens$clone()`

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

Usage:

```
PredictionDens$new(
  task = NULL,
  row_ids = task$row_ids,
  pdf = NULL,
  cdf = NULL,
  distr = NULL,
  check = TRUE
)
```

Arguments:

`task` (TaskSurv)
Task, used to extract defaults for `row_ids`.

`row_ids` (`integer()`)
Row ids of the predicted observations, i.e. the row ids of the test set.

`pdf` (`numeric()`)
Numeric vector of estimated probability density function, evaluated at values in test set. One element for each observation in the test set.

`cdf` (`numeric()`)
: Numeric vector of estimated cumulative distribution function, evaluated at values in test set. One element for each observation in the test set.

`distr` (Distribution)
: Distribution from distr6. The distribution from which `pdf` and `cdf` are derived.

`check` (`logical(1)`)
: If `TRUE`, performs argument checks and predict type conversions.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Prediction: `PredictionSurv`

Examples

```
library(mlr3)
task = mlr_tasks$get("precip")
learner = mlr_learners$get("dens.hist")
p = learner$train(task)$predict(task)
head(as.data.table(p))
```

PredictionSurv *Prediction Object for Survival*

Description

This object stores the predictions returned by a learner of class LearnerSurv.

The `task_type` is set to `"surv"`.

For accessing survival and hazard functions, as well as other complex methods from a PredictionSurv object, see public methods on `distr6::ExoticStatistics()` and example below.

Super class

`mlr3::Prediction` -> `PredictionSurv`

Active bindings

`truth` (`Surv`)
: True (observed) outcome.

`crank` (`numeric()`)
: Access the stored predicted continuous ranking.

`distr` (distr6::Matdist|distr6::Arrdist|distr6::VectorDistribution)
: Convert the stored survival array or matrix to a survival distribution.

`lp` (`numeric()`)
: Access the stored predicted linear predictor.

`response` (`numeric()`)
: Access the stored predicted survival time.

Methods

Public methods:

- `PredictionSurv$new()`
- `PredictionSurv$clone()`

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

Usage:

```
PredictionSurv$new(
  task = NULL,
  row_ids = task$row_ids,
  truth = task$truth(),
  crank = NULL,
  distr = NULL,
  lp = NULL,
  response = NULL,
  check = TRUE
)
```

Arguments:

`task` (TaskSurv)
: Task, used to extract defaults for `row_ids` and `truth`.

`row_ids` (`integer()`)
: Row ids of the predicted observations, i.e. the row ids of the test set.

`truth` (`survival::Surv()`)
: True (observed) response.

`crank` (`numeric()`)
: Numeric vector of predicted continuous rankings (or relative risks). One element for each observation in the test set. For a pair of continuous ranks, a higher rank indicates that the observation is more likely to experience the event.

`distr` (`matrix()|[distr6::Arrdist]|[distr6::Matdist]|[distr6::VectorDistribution]`)
: Either a matrix of predicted survival probabilities, a distr6::VectorDistribution, a distr6::Matdist or an distr6::Arrdist. If a matrix/array then column names must be given and correspond to survival times. Rows of matrix correspond to individual predictions. It is advised that

the first column should be time `0` with all entries `1` and the last with all entries `0`. If a `VectorDistribution` then each distribution in the vector should correspond to a predicted survival distribution.

`lp` (`numeric()`)
: Numeric vector of linear predictor scores. One element for each observation in the test set. $lp = X\beta$ where X is a matrix of covariates and β is a vector of estimated coefficients.

`response` (`numeric()`)
: Numeric vector of predicted survival times. One element for each observation in the test set.

`check` (`logical(1)`)
: If `TRUE`, performs argument checks and predict type conversions.

Details: Upon **initialization**, if the `distr` input is a Distribution, we try to coerce it either to a survival matrix or a survival array and store it in the `$data$distr` slot for internal use.

If the stored `$data$distr` is a Distribution object, the active field `$distr` (**external user API**) returns it without modification. Otherwise, if `$data$distr` is a survival matrix or array, `$distr` constructs a distribution out of the `$data$distr` object, which will be a Matdist or Arrdist respectively.

Note that if a survival 3d array is stored in `$data$distr`, the `$distr` field returns an Arrdist initialized with `which.curve = 0.5` by default (i.e. the median curve). This means that measures that require a `distr` prediction like MeasureSurvGraf, MeasureSurvRCLL, etc. will use the median survival probabilities. Note that it is possible to manually change `which.curve` after construction of the predicted distribution but we advise against this as it may lead to inconsistent results.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Prediction: `PredictionDens`

Examples

```
library(mlr3)
task = tsk("rats")
learner = lrn("surv.kaplan")
p = learner$train(task, row_ids = 1:26)$predict(task, row_ids = 27:30)
head(as.data.table(p))

p$distr # distr6::Matdist class (test obs x time points)

# survival probabilities of the 4 test rats at two time points
p$distr$survival(c(20, 100))
```

TaskDens *Density Task*

Description

This task specializes TaskUnsupervised for density estimation problems. The data in backend should be a numeric vector or a one column matrix-like object. The `task_type` is set to `"density"`.

Predefined tasks are stored in the dictionary mlr_tasks.

Super classes

`mlr3::Task` -> `mlr3::TaskUnsupervised` -> `TaskDens`

Methods

Public methods:

- `TaskDens$new()`
- `TaskDens$clone()`

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

Usage:

```
TaskDens$new(id, backend, label = NA_character_)
```

Arguments:

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

`backend` (DataBackend)
Either a DataBackend, a matrix-like object, or a numeric vector. If weights are used then two columns expected, otherwise one column. The weight column must be clearly specified (via `[Task]$col_roles`) or the learners will break.

`label` (`character(1)`)
Label for the new instance.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Task: `TaskSurv`, `mlr_tasks_actg`, `mlr_tasks_faithful`, `mlr_tasks_gbcs`, `mlr_tasks_gbsg`, `mlr_tasks_grace`, `mlr_tasks_lung`, `mlr_tasks_mgus`, `mlr_tasks_pbc`, `mlr_tasks_precip`, `mlr_tasks_rats`, `mlr_tasks_unemployment`, `mlr_tasks_veteran`, `mlr_tasks_whas`

Examples

```
task = TaskDens$new("precip", backend = precip)
task$task_type
```

TaskSurv *Survival Task*

Description

This task specializes mlr3::Task and mlr3::TaskSupervised for possibly-censored survival problems. The target is comprised of survival times and an event indicator. Predefined tasks are stored in mlr3::mlr_tasks.

The `task_type` is set to `"surv"`.

Super classes

`mlr3::Task` -> `mlr3::TaskSupervised` -> `TaskSurv`

Active bindings

`censtype (character(1))`
: Returns the type of censoring, one of `"right"`, `"left"`, `"counting"`, `"interval"`, `"interval2"` or `"mstate"`. Currently, only the `"right"`-censoring type is fully supported, the rest are experimental and the API will change in the future.

Methods

Public methods:

- `TaskSurv$new()`
- `TaskSurv$truth()`
- `TaskSurv$formula()`
- `TaskSurv$times()`
- `TaskSurv$status()`
- `TaskSurv$unique_times()`
- `TaskSurv$unique_event_times()`
- `TaskSurv$risk_set()`
- `TaskSurv$kaplan()`
- `TaskSurv$reverse()`
- `TaskSurv$cens_prop()`
- `TaskSurv$admin_cens_prop()`
- `TaskSurv$dep_cens_prop()`
- `TaskSurv$prop_haz()`
- `TaskSurv$clone()`

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

Usage:

```
TaskSurv$new(
  id,
  backend,
  time = "time",
  event = "event",
  time2,
  type = c("right", "left", "interval", "counting", "interval2", "mstate"),
  label = NA_character_
)
```

Arguments:

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

`backend (DataBackend)`
: Either a DataBackend, or any object which is convertible to a DataBackend with `as_data_backend()`. E.g., a `data.frame()` will be converted to a DataBackendDataTable.

`time (character(1))`
: Name of the column for event time if data is right censored, otherwise starting time if interval censored.

`event (character(1))`
: Name of the column giving the event indicator. If data is right censored then "0"/`FALSE` means alive (no event), "1"/`TRUE` means dead (event). If type is `"interval"` then "0" means right censored, "1" means dead (event), "2" means left censored, and "3" means interval censored. If `type` is `"interval2"` then `event` is ignored.

`time2 (character(1))`
: Name of the column for ending time of the interval for interval censored or counting process data, otherwise ignored.

`type (character(1))`
: Name of the column giving the type of censoring. Default is 'right' censoring.

`label (character(1))`
: Label for the new instance.

Details: Depending on the censoring type (`"type"`), the output of a survival task's `"$target_names"` is a `character()` vector with values the names of the columns given by the above initialization arguments. Specifically, the output is as follows (and in the specified order):

- For `type` = `"right"`, `"left"` or `"mstate"`: (`"time"`, `"event"`)
- For `type` = `"interval"` or `"counting"`: (`"time"`, `"time2"`, `"event"`)
- For `type` = `"interval2"`: (`"time"`, `"time2`)

Method `truth()`**:** True response for specified `row_ids`. This is the survival outcome using the Surv format and depends on the censoring type. Defaults to all rows with role `"use"`.

Usage:

```
TaskSurv$truth(rows = NULL)
```

Arguments:

`rows (integer())`
: Row indices.

Returns: `survival::Surv()`.

Method `formula()`**:** Creates a formula for survival models with `survival::Surv()` on the LHS (left hand side).

Usage:

```
TaskSurv$formula(rhs = NULL, reverse = FALSE)
```

Arguments:

`rhs` If `NULL`, RHS (right hand side) is `"."`, otherwise RHS is `"rhs"`.

`reverse` If `TRUE` then formula calculated with 1 - status.

Returns: `stats::formula()`.

Method `times()`**:** Returns the (unsorted) outcome times.

Usage:

```
TaskSurv$times(rows = NULL)
```

Arguments:

`rows (integer())`
Row indices.

Returns: `numeric()`

Method `status()`**:** Returns the event indicator (aka censoring/survival indicator). If `censtype` is `"right"` or `"left"` then `1` is event and `0` is censored. If `censtype` is `"mstate"` then `0` is censored and all other values are different events. If `censtype` is `"interval"` then `0` is right-censored, `1` is event, `2` is left-censored, `3` is interval-censored. See `survival::Surv()`.

Usage:

```
TaskSurv$status(rows = NULL)
```

Arguments:

`rows (integer())`
Row indices.

Returns: `integer()`

Method `unique_times()`**:** Returns the sorted unique outcome times for `"right"`, `"left"` and `"mstate"` types of censoring.

Usage:

```
TaskSurv$unique_times(rows = NULL)
```

Arguments:

`rows (integer())`
Row indices.

Returns: `numeric()`

Method `unique_event_times()`**:** Returns the sorted unique event (or failure) outcome times for `"right"`, `"left"` and `"mstate"` types of censoring.

Usage:

```
TaskSurv$unique_event_times(rows = NULL)
```

Arguments:

`rows` (`integer()`)
Row indices.

Returns: `numeric()`

Method `risk_set()`**:** Returns the `row_ids` of the observations **at risk** (not dead or censored or had other events in case of multi-state tasks) at the specified `time`.

Only designed for `"right"`, `"left"` and `"mstate"` types of censoring.

Usage:

```
TaskSurv$risk_set(time = NULL)
```

Arguments:

`time` (`numeric(1)`)
Time to return risk set for, if `NULL` returns all `row_ids`.

Returns: `integer()`

Method `kaplan()`**:** Calls `survival::survfit()` to calculate the Kaplan-Meier estimator.

Usage:

```
TaskSurv$kaplan(strata = NULL, rows = NULL, reverse = FALSE, ...)
```

Arguments:

`strata` (`character()`)
Stratification variables to use.

`rows` (`integer()`)
Subset of row indices.

`reverse` (`logical()`)
If `TRUE` calculates Kaplan-Meier of censoring distribution (1-status). Default `FALSE`.

`...` (any)
Additional arguments passed down to `survival::survfit.formula()`.

Returns: survival::survfit.object.

Method `reverse()`**:** Returns the same task with the status variable reversed, i.e., 1 - status. Only designed for `"left"` and `"right"` censoring.

Usage:

```
TaskSurv$reverse()
```

Returns: TaskSurv.

Method `cens_prop()`**:** Returns the **proportion of censoring** for this survival task. By default, this is returned for all observations, otherwise only the specified ones (`rows`).

Only designed for `"right"` and `"left"` censoring.

Usage:

```
TaskSurv$cens_prop(rows = NULL)
```

Arguments:

`rows` (`integer()`)
Row indices.

Returns: `numeric()`

Method `admin_cens_prop()`**:** Returns an estimated proportion of **administratively censored observations** (i.e. censored at or after a user-specified time point). Our main assumption here is that in an administratively censored dataset, the maximum censoring time is likely close to the maximum event time and so we expect higher proportion of censored subjects near the study end date.

Only designed for `"right"` and `"left"` censoring.

Usage:

```
TaskSurv$admin_cens_prop(rows = NULL, admin_time = NULL, quantile_prob = 0.99)
```

Arguments:

`rows (integer())`
Row indices.

`admin_time (numeric(1))`
Administrative censoring time (in case it is known *a priori*).

`quantile_prob (numeric(1))`
Quantile probability value with which we calculate the cutoff time for administrative censoring. Ignored, if `admin_time` is given. By default, `quantile_prob` is equal to 0.99, which translates to a time point very close to the maximum outcome time in the dataset. A lower value will result in an earlier time point and therefore in a more *relaxed* definition (i.e. higher proportion) of administrative censoring.

Returns: `numeric()`

Method `dep_cens_prop()`**:** Returns the proportion of covariates (task features) that are found to be significantly associated with censoring. This function fits a logistic regression model via glm with the censoring status as the response and using all features as predictors. If a covariate is significantly associated with the censoring status, it suggests that censoring may be *informative* (dependent) rather than *random* (non-informative). This methodology is more suitable for **low-dimensional datasets** where the number of features is relatively small compared to the number of observations.

Only designed for `"right"` and `"left"` censoring.

Usage:

```
TaskSurv$dep_cens_prop(rows = NULL, method = "holm", sign_level = 0.05)
```

Arguments:

`rows (integer())`
Row indices.

`method (character(1))`
Method to adjust p-values for multiple comparisons, see p.adjust.methods. Default is `"holm"`.

`sign_level (numeric(1))`
Significance level for each coefficient's p-value from the logistic regression model. Default is 0.05.

Returns: `numeric()`

Method `prop_haz()`**:** Checks if the data satisfy the *proportional hazards (PH)* assumption using the Grambsch-Therneau test, Grambsch (1994). Uses cox.zph. This method should be used only for **low-dimensional datasets** where the number of features is relatively small compared to the number of observations.

Only designed for "right" and "left" censoring.

Usage:

```
TaskSurv$prop_haz()
```

Returns: numeric()

If no errors, the p-value of the global chi-square test. A p-value < 0.05 is an indication of possible PH violation.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Grambsch, Patricia, Therneau, Terry (1994). "Proportional hazards tests and diagnostics based on weighted residuals." *Biometrika*, **81**(3), 515–526. doi:10.1093/biomet/81.3.515, https://doi.org/10.1093/biomet/81.3.515.

See Also

Other Task: TaskDens, mlr_tasks_actg, mlr_tasks_faithful, mlr_tasks_gbcs, mlr_tasks_gbsg, mlr_tasks_grace, mlr_tasks_lung, mlr_tasks_mgus, mlr_tasks_pbc, mlr_tasks_precip, mlr_tasks_rats, mlr_tasks_unemployment, mlr_tasks_veteran, mlr_tasks_whas

Examples

```
library(mlr3)
task = tsk("lung")

# meta data
task$target_names # target is always (time, status) for right-censoring tasks
task$feature_names
task$formula()

# survival data
task$truth() # survival::Surv() object
task$times() # (unsorted) times
task$status() # event indicators (1 = death, 0 = censored)
task$unique_times() # sorted unique times
task$unique_event_times() # sorted unique event times
task$risk_set(time = 700) # observation ids that are not censored or dead at t = 700
task$kaplan(strata = "sex") # stratified Kaplan-Meier
task$kaplan(reverse = TRUE) # Kaplan-Meier of the censoring distribution

# proportion of censored observations across all dataset
task$cens_prop()
```

```
# proportion of censored observations at or after the 95% time quantile
task$admin_cens_prop(quantile_prob = 0.95)
# proportion of variables that are significantly associated with the
# censoring status via a logistic regression model
task$dep_cens_prop() # 0 indicates independent censoring
# data barely satisfies proportional hazards assumption (p > 0.05)
task$prop_haz()
# veteran data is definitely non-PH (p << 0.05)
tsk("veteran")$prop_haz()
```

`whas` *Worcester Heart Attack Study (WHAS) Dataset*

Description

whas dataset from Hosmer et al. (2008)

Usage

```
whas
```

Format

id Identification Code

age Age (per chart) (years).

sex Sex. 0 = Male. 1 = Female.

cpk Peak cardiac enzyme (iu).

sho Cardiogenic shock complications. 1 = Yes. 0 = No.

chf Left heart failure complications. 1 = Yes. 0 = No.

miord MI Order. 1 = Recurrent. 0 = First.

mitype MI Type. 1 = Q-wave. 2 = Not Q-wave. 3 = Indeterminate.

year Cohort year.

yrgrp Grouped cohort year.

lenstay Days in hospital.

dstat Discharge status from hospital. 1 = Dead. 0 = Alive.

lenfol Total length of follow-up from hospital admission (days).

fstat Status as of last follow-up. 1 = Dead. 0 = Alive.

Source

https://onlinelibrary.wiley.com/doi/book/10.1002/9780470258019

References

Hosmer, D.W. and Lemeshow, S. and May, S. (2008) Applied Survival Analysis: Regression Modeling of Time to Event Data: Second Edition, John Wiley and Sons Inc., New York, NY

Index