

Package ‘mlr3proba’

October 22, 2021

Title Probabilistic Supervised Learning for 'mlr3'

Version 0.4.2

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 (>= 0.12.0), R (>= 3.5.0)

Imports checkmate, data.table, distr6 (>= 1.6.0), mlr3misc (>= 0.7.0),
paradox (>= 0.1.0), R6, Rcpp (>= 1.0.4), survival

Suggests bujar, cubature, ggplot2, knitr, lgr, mlr3pipelines (>= 0.3.4),
pracma, rpart, set6 (>= 0.1.7), simsurv, survAUC, testthat

LinkingTo Rcpp

ByteCompile true

Encoding UTF-8

LazyData true

NeedsCompilation yes

RoxygenNote 7.1.2

Collate 'LearnerDens.R' 'LearnerDensHistogram.R' 'LearnerDensKDE.R'
'LearnerRegrGaussian.R' 'LearnerSurv.R' 'LearnerSurvCoxPH.R'
'LearnerSurvKaplan.R' 'LearnerSurvRpart.R' 'MeasureDens.R'
'MeasureDensLogloss.R' 'MeasureRegrLogloss.R' 'MeasureSurv.R'
'MeasureSurvIntegrated.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'
 'MeasureSurvRMSE.R' 'MeasureSurvSchmid.R'
 'MeasureSurvSongAUC.R' 'MeasureSurvSongTNR.R'
 'MeasureSurvSongTPR.R' 'MeasureSurvUnoAUC.R'
 'MeasureSurvUnoTNR.R' 'MeasureSurvUnoTPR.R' 'MeasureSurvXuR2.R'
 'PipeOpCrankCompositor.R' 'PipeOpDistrCompositor.R'
 'PipeOpTransformer.R' 'PipeOpPredTransformer.R'
 'PipeOpPredRegrSurv.R' 'PipeOpPredSurvRegr.R'
 'PipeOpProbregrCompositor.R' 'PipeOpSurvAvg.R'
 'PipeOpTaskRegrSurv.R' 'PipeOpTaskSurvRegr.R'
 'PipeOpTaskTransformer.R' 'PredictionDataDens.R'
 'PredictionDataSurv.R' 'PredictionDens.R' 'PredictionSurv.R'
 'RcppExports.R' 'TaskDens.R' 'TaskDens_X.R'
 'TaskGeneratorSimdens.R' 'TaskGeneratorSimsurv.R' 'TaskSurv.R'
 'TaskSurv_X.R' 'as_prediction_surv.R' 'as_task_surv.R'
 'assertions.R' 'bibentries.R' 'cindex.R' 'data.R' 'helpers.R'
 'histogram.R' 'integrated_scores.R' 'pecs.R' 'pipelines.R'
 'plot.R' 'surv_measures.R' 'surv_return.R' 'zzz.R'

Author Raphael Sonabend [aut, cre] (<<https://orcid.org/0000-0001-9225-4654>>),
 Franz Kiraly [aut],
 Michel Lang [aut] (<<https://orcid.org/0000-0001-9754-0393>>),
 Nurul Ain Toha [ctb],
 Andreas Bender [ctb] (<<https://orcid.org/0000-0001-5628-8611>>)

Maintainer Raphael Sonabend <raphaelsonabend@gmail.com>

Repository CRAN

Date/Publication 2021-10-22 09:20:02 UTC

R topics documented:

mlr3proba-package	4
.surv_return	5
actg	6
assert_surv	7
as_prediction_surv	7
as_task_surv	8
gbcs	9
grace	10
LearnerDens	11
LearnerSurv	13
MeasureDens	15
MeasureSurv	17
MeasureSurvAUC	18
MeasureSurvIntegrated	20
mlr_graphs_crankcompositor	22

mlr_graphs_distcompositor	23
mlr_graphs_probregrcompositor	25
mlr_graphs_survaverager	27
mlr_graphs_survbagging	28
mlr_graphs_survtoregr	29
mlr_learners_dens.hist	33
mlr_learners_dens.kde	34
mlr_learners_surv.coxph	35
mlr_learners_surv.kaplan	36
mlr_learners_surv.rpart	38
mlr_measures_dens.logloss	39
mlr_measures_surv.calib_alpha	40
mlr_measures_surv.calib_beta	42
mlr_measures_surv.chambless_auc	44
mlr_measures_surv.cindex	45
mlr_measures_surv.dcalib	48
mlr_measures_surv.graf	50
mlr_measures_surv.hung_auc	53
mlr_measures_surv.intlogloss	54
mlr_measures_surv.logloss	57
mlr_measures_surv.mae	60
mlr_measures_surv.mse	61
mlr_measures_surv.nagelk_r2	63
mlr_measures_surv.oquigley_r2	64
mlr_measures_surv.rmse	66
mlr_measures_surv.schmid	67
mlr_measures_surv.song_auc	70
mlr_measures_surv.song_tnr	72
mlr_measures_surv.song_tpr	74
mlr_measures_surv.uno_auc	76
mlr_measures_surv.uno_tnr	78
mlr_measures_surv.uno_tpr	80
mlr_measures_surv.xu_r2	82
mlr_pipeops_compose_crank	83
mlr_pipeops_compose_distr	86
mlr_pipeops_compose_probregr	88
mlr_pipeops_survavg	90
mlr_pipeops_trafopred_regrsurv	92
mlr_pipeops_trafopred_survregr	94
mlr_pipeops_trafotask_regrsurv	95
mlr_pipeops_trafotask_survregr	97
mlr_tasks_actg	100
mlr_tasks_faithful	101
mlr_tasks_gbcs	101
mlr_tasks_grace	102
mlr_tasks_lung	102
mlr_tasks_precip	103
mlr_tasks_rats	103

mlr_tasks_unemployment	104
mlr_tasks_whas	104
mlr_task_generators_simdens	105
mlr_task_generators_simsurv	106
pecs	107
PipeOpPredTransformer	109
PipeOpTaskTransformer	111
PipeOpTransformer	112
plot.LearnerSurv	114
PredictionDens	115
PredictionSurv	116
TaskDens	118
TaskSurv	119
whas	123

Index	124
--------------	------------

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: Raphael Sonabend <raphaelsonabend@gmail.com> ([ORCID](#))

Authors:

- Franz Kiraly <f.kiraly@ucl.ac.uk>
- Michel Lang <michellang@gmail.com> ([ORCID](#))

Other contributors:

- Nurul Ain Toha <nurul.toha.15@ucl.ac.uk> [contributor]
- Andreas Bender <bender.at.R@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>

Description

Internal helper function to easily return the correct survival predict types and to automatically coerce a predicted survival probability matrix to a [distr6::VectorDistribution](#) of [distr6::WeightedDiscrete](#) distributions.

Usage

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

Arguments

times	(numeric()) Vector of survival times.
surv	(matrix()) Matrix of predicted survival probabilities, rows are observations, columns are times. Number of columns should be equal to length of times.
crank	(numeric()) Relative risk/continuous ranking. Higher value is associated with higher risk. If NULL then either set as lp if available or as the estimated survival expectation, computed by <code>colSums(surv)</code> .
lp	(numeric()) Predicted linear predictor, used to impute crank if NULL.
response	(numeric()) Predicted survival time, passed through function without modification.

Details

To avoid complications caused by degenerative distributions, if 0 is not in times, then this is added as the first time-point and a column of 1s is bound to the left of surv. Additionally if the final column of surv is not all 1s, then an additional time is added as `max(times) + 1e-3` and a column of 1s is bound to the right of surv.

actg

ACTG 320 Clinical Trial Dataset

Description

actg dataset from Hosmer et al.

Usage

actg

Format

id Identification Code

time Time to AIDS diagnosis or death (days).

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

time_d Time to death (days)

cancel_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

ftp://ftp.wiley.com/public/sci_tech_med/survival

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	<i>Assert survival object</i>
-------------	-------------------------------

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

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

as_prediction_surv	<i>Convert to a Survival Prediction</i>
--------------------	---

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_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)),
  ...
)

```

Arguments

x	(any) Object to convert, e.g. a <code>data.frame()</code> .
...	(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 for interval censored 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.

 gbcs

German Breast Cancer Study (GBCS) Dataset

Description

gbcs dataset from Hosmer et al.

Usage

gbcs

Format

id Identification Code

diagdate Date of diagnosis.

reccdate 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 nodes.

prog_recp Number of progesterone receptors.

estrg_recp Number of estrogen receptors.

rectime Time to recurrence (days).

consrec Recurrence status. 1 = Recurrence. 0 = Censored.

survtime Time to death (days).

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

Source

ftp://ftp.wiley.com/public/sci_tech_med/survival

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

grace

GRACE 1000 Dataset

Description

grace dataset from Hosmer et al.

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

ftp://ftp.wiley.com/public/sci_tech_med/survival

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$new(
  id,
  param_set = ParamSet$new(),
  predict_types = "cdf",
  feature_types = character(),
  properties = character(),
  data_formats = "data.table",
  packages = 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](#)).

`data_formats` (`character()`)

Set of supported data formats which can be processed during `$train()` and `$predict()`, e.g. "data.table".

`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\(\)](#).

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$new(
  id,
  param_set = ParamSet$new(),
  predict_types = "distr",
  feature_types = character(),
  properties = character(),
  packages = 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()`.

`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(),
  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\(\)](#).

`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(),
  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()`.

`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` (logical(1))
 If TRUE returns the standard error of the measure.

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.

Super classes

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

Methods

Public methods:

- [MeasureSurvAUC\\$new\(\)](#)
- [MeasureSurvAUC\\$clone\(\)](#)

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

Usage:

```
MeasureSurvAUC$new(
  integrated = TRUE,
  times,
  id,
  properties = character(),
  man = NA_character_
)
```

Arguments:

`integrated` (`logical(1)`)

If TRUE (default), returns the integrated score; otherwise, not integrated.

`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.

`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).

`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:

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

Arguments:

`deep` Whether to make a deep clone.

MeasureSurvIntegrated *Abstract Class for Integrated Measures*

Description

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

Super classes

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

Active bindings

`eps` (numeric(1))

Very small number used to prevent $\log(0)$ and $1/0$ error.

`integrated` (logical(1)) Returns if the measure should be integrated or not. Settable.

`times` (numeric()) Returns the times at which the measure should be evaluated at, or integrated over. Settable.

`method` (integer(1)) Returns which method is used for approximating integration. Settable.

`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 to weight the measure. If FALSE then weights scores by the Graf method which is the more common usage but the loss is not proper.

Methods

Public methods:

- `MeasureSurvIntegrated$new()`
- `MeasureSurvIntegrated$clone()`

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

Usage:

```
MeasureSurvIntegrated$new(
  integrated = TRUE,
  times,
  method = 2,
  proper = FALSE,
  eps = 0.000000000000001,
  id,
  range,
  minimize,
  packages,
  predict_type,
  properties = character(),
```

```

    man = NA_character_,
    se = FALSE
  )

```

Arguments:

`integrated` (logical(1))
 If TRUE (default), returns the integrated score; otherwise, not integrated.

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

`method` (integer(1))
 If `integrate == TRUE` 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 default to be in line with other packages.

`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 to weight the measure. If FALSE then weights scores by the Graf method which is the more common usage but the loss is not proper. In v0.5.0, the default will be changed to TRUE.

`eps` (numeric(1))
 Very small number to set zero-valued predicted probabilities to in order to prevent errors in `log(0)` and `1/0` calculation.

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

`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.

`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()`.

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

`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).

`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 (logical(1))
If TRUE returns the standard error of the measure.

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

Usage:
MeasureSurvIntegrated\$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("mean", "median", "mode"),  
  which = NULL,  
  response = FALSE,  
  overwrite = FALSE,  
  graph_learner = FALSE  
)  
  
crankcompositor(...)
```

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) One of mean (default), mode, or median; abbreviations allowed. Used to determine how crank is estimated from the predicted distr.
which	integer(1) If method = "mode" then specifies which mode to use if multi-modal, default is the first.
response	logical(1) If TRUE then the response predict type is also estimated with the same values as crank.

overwrite	logical(1) If TRUE then existing response and crank predict types are overwritten.
graph_learner	logical(1) If TRUE returns wraps the Graph as a GraphLearner otherwise (default) returns as a Graph .
...	ANY For use with crankcompositor , now deprecated.

Value

[mlr3pipelines::Graph](#) or [mlr3pipelines::GraphLearner](#)

See Also

Other pipelines: [mlr_graphs_distrcompositor](#), [mlr_graphs_probregrcompositor](#), [mlr_graphs_survaverager](#), [mlr_graphs_survbagging](#), [mlr_graphs_survtoregr](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  library("mlr3")
  library("mlr3pipelines")

  task = tsk("rats")
  pipe = ppl(
    "crankcompositor",
    learner = lrn("surv.coxph"),
    method = "median"
  )
  pipe$train(task)
  pipe$predict(task)
}

## End(Not run)
```

mlr_graphs_distrcompositor

Estimate Survival distr Predict Type Pipeline

Description

Wrapper around [PipeOpDistrCompositor](#) to simplify [Graph](#) creation.

Usage

```

pipeline_distrcompositor(
  learner,
  estimator = c("kaplan", "nelson"),
  form = c("aft", "ph", "po"),
  overwrite = FALSE,
  graph_learner = FALSE
)

distrcompositor(...)

```

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) or nelson, corresponding to the Kaplan-Meier and Nelson-Aalen 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.
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.
graph_learner	logical(1) If TRUE returns wraps the Graph as a GraphLearner otherwise (default) returns as a Graph.
...	ANY For use with <code>distrcompositor</code> , now deprecated.

Value

[mlr3pipelines::Graph](#) or [mlr3pipelines::GraphLearner](#)

See Also

Other pipelines: [mlr_graphs_crankcompositor](#), [mlr_graphs_probregrcompositor](#), [mlr_graphs_survaverager](#), [mlr_graphs_survbagging](#), [mlr_graphs_survtoregr](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE) &&
    requireNamespace("rpart", quietly = TRUE)) {
  library("mlr3")
  library("mlr3pipelines")

  task = tsk("rats")
  pipe = ppl(
    "distrcompositor",
    learner = lrn("surv.rpart"),
    estimator = "kaplan",
    form = "ph"
  )
  pipe$train(task)
  pipe$predict(task)
}

## End(Not run)
```

mlr_graphs_probregrcompositor

Estimate Regression distr Predict Type Pipeline

Description

Wrapper around [PipeOpProbregrCompositor](#) to simplify [Graph](#) creation.

Usage

```
pipeline_probregrcompositor(
  learner,
  learner_se = NULL,
  dist = "Normal",
  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 (default)'.
 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](#)

See Also

Other pipelines: [mlr_graphs_crankcompositor](#), [mlr_graphs_distrcompositor](#), [mlr_graphs_survaverager](#), [mlr_graphs_survbagging](#), [mlr_graphs_survtoregr](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE) &&
    requireNamespace("rpart", quietly = TRUE)) {
  library("mlr3")
  library("mlr3pipelines")

  task = tsk("boston_housing")

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

  # method 2 - one learner for response and one for se
  pipe = ppl(
    "probregrcompositor",
    learner = lrn("regr.rpart"),
    learner_se = lrn("regr.featureless", predict_type = "se"),
    dist = "Logistic"
  )
  pipe$train(task)
  pipe$predict(task)
}

## 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 LearnerSurv s 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](#)

See Also

Other pipelines: [mlr_graphs_crankcompositor](#), [mlr_graphs_distrcompositor](#), [mlr_graphs_probregrcompositor](#), [mlr_graphs_survbagging](#), [mlr_graphs_survtoregr](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  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](#)
[mlr3pipelines::GraphLearner](#)

See Also

Other pipelines: [mlr_graphs_crankcompositor](#), [mlr_graphs_distrcompositor](#), [mlr_graphs_probregrcompositor](#), [mlr_graphs_survaverager](#), [mlr_graphs_survtoregr](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  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_survtoregr` *Survival to Regression Reduction Pipeline*

Description

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

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 = "Normal"),
  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 <code>regr_se_learner</code> is NULL in method 2, then <code>regr_learner</code> must have <code>se predict_type</code> .
distrcompose	logical(1) For methods 1 and 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 <code>distrcompose = TRUE</code> then specifies the learner to estimate the baseline hazard, must have <code>predict_type distr</code> .
regr_se_learner	LearnerRegr For method 2 if <code>regr_learner</code> is not used to predict the <code>se</code> then a LearnerRegr with <code>se predict_type</code> must be provided.
surv_learner	LearnerSurv For method 3, a LearnerSurv with <code>lp predict type</code> to estimate linear predictors.
survregr_params	list() Parameters passed to PipeOpTaskSurvRegr , default are survival to regression transformation via <code>ipcw</code> , 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 PipeOpProbregrCompositor , default is Normal 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](#).
 - (d) Optionally: [PipeOpDistrCompositor](#) is used to compose a `distr predict_type` from the predicted response `predict_type`.
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) [PipeOpProbregrCompositor](#) 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.

Value

[mlr3pipelines::Graph](#) or [mlr3pipelines::GraphLearner](#)

See Also

Other pipelines: [mlr_graphs_crankcompositor](#), [mlr_graphs_distrcompositor](#), [mlr_graphs_probregrcompositor](#), [mlr_graphs_survaverager](#), [mlr_graphs_survbagging](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  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"),
    distrcompose = TRUE,
    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: **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: **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

- Type: "surv"
- Predict Types: `distr`, `crank`, `lp`
- Feature Types: logical, integer, numeric, factor
- Properties: `weights`
- Packages: **survival distr6**

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](https://doi.org/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 expectation of 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

- Type: "surv"
- Predict Types: crank, distr
- Feature Types: logical, integer, numeric, character, factor, ordered
- Properties: missings
- Packages: **survival distr6**

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](https://doi.org/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()`

Parameter `xval` is set to 0 in order to save some computation time.

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

- Type: "surv"
- Predict Types: `crank`, `distr`
- Feature Types: `logical`, `integer`, `numeric`, `character`, `factor`, `ordered`
- Properties: `importance`, `missings`, `selected_features`, `weights`
- Packages: **rpart distr6 survival**

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](https://doi.org/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.

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

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

Meta Information

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

Super classes

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

Active bindings

eps Returns eps parameter, see initialize.

Methods**Public methods:**

- MeasureDensLogloss\$new()
- MeasureDensLogloss\$clone()

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

Usage:

```
MeasureDensLogloss$new(eps = 0.000000000000001)
```

Arguments:

eps (numeric(1))

Very small number to set zero-valued predicted probabilities to in order to prevent errors in log(0) and 1/0 calculation.

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_surv.calib_alpha

Van Houwelingen's Alpha Survival Measure

Description

This calibration method is defined by estimating

$$\alpha = \sum \delta_i / \sum H_i(t_i)$$

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.

The standard error is given by

$$\exp(1/\sqrt{\sum \delta_i})$$

The model is well calibrated if the estimated α coefficient 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")
```

Meta Information

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

Super classes

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

Active bindings

```
se (logical(1))
  If TRUE returns the standard error of the measure.
```

Methods**Public methods:**

- [MeasureSurvCalibrationAlpha\\$new\(\)](#)
- [MeasureSurvCalibrationAlpha\\$clone\(\)](#)

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

Usage:

```
MeasureSurvCalibrationAlpha$new(se = FALSE)
```

Arguments:

```
se (logical(1))
  If TRUE returns the standard error of the measure.
```

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](https://doi.org/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.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_beta`, `mlr_measures_surv.dcalib`

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

`mlr_measures_surv.calib_beta`

Van Houwelingen's 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.

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

where l is the predicted linear predictor.

The model is well calibrated if the estimated β coefficient is equal to 1.

Assumes fitted model is Cox PH.

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")
```

Meta Information

- Type: "surv"
- Range: $(-\infty, \infty)$
- Minimize: FALSE
- Required prediction: 1p

Super classes

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

Active bindings

se (logical(1))
If TRUE returns the standard error of the measure.

Methods**Public methods:**

- [MeasureSurvCalibrationBeta\\$new\(\)](#)
- [MeasureSurvCalibrationBeta\\$clone\(\)](#)

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

Usage:

```
MeasureSurvCalibrationBeta$new(se = FALSE)
```

Arguments:

se (logical(1))
If TRUE returns the standard error of the measure.

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](https://doi.org/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.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.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 these measures will all 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")
```

Meta Information

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

Super classes

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

Methods

Public methods:

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

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

Usage:

```
MeasureSurvChamblessAUC$new(integrated = TRUE, times)
```

Arguments:

`integrated` (logical(1))

If TRUE (default), returns the integrated score; otherwise, not integrated.

`times` (numeric())

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

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](https://doi.org/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.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.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 and tied times solution, are equivalent to several proposed methods.

For the Kaplan-Meier estimate of the training survival distribution, S , and the Kaplan-Meier estimate of the training censoring distribution, G :

`weight_meth`:

- "I" = No weighting. (Harrell)
- "GH" = Gonen and Heller's Concordance Index
- "G" = Weights concordance by G^{-1} .
- "G2" = Weights concordance by 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](#).

@details 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")
```

Meta Information

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

Super classes

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

Active bindings

cutoff (numeric(1)) Cut-off time to evaluate concordance up to.

weight_meth (numeric(1)) Method for weighting concordance.

tiex (numeric(1)) Cut-off time to evaluate concordance up to.

Methods

Public methods:

- [MeasureSurvCindex\\$new\(\)](#)
- [MeasureSurvCindex\\$clone\(\)](#)

Method [new\(\)](#): This is an abstract class that should not be constructed directly.

Usage:

```
MeasureSurvCindex$new(
  cutoff = NULL,
  weight_meth = c("I", "G", "G2", "SG", "S", "GH"),
  tiex = 0.5
)
```

Arguments:

cutoff (numeric(1))

Cut-off time to evaluate concordance up to.

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 weighting.

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.
- Gönen 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](https://doi.org/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](https://doi.org/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](https://doi.org/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_tpr`, `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`

 mlr_measures_surv.dcalib

D-Calibration Survival Measure

Description

This calibration method is defined by calculating

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

where B is number of 'buckets', n is the number of predictions, and P_i is the predicted number of deaths in the i th interval $[0, 100/B)$, $[100/B, 50/B)$, ..., $[(B - 100)/B, 1)$.

A model is well-calibrated if $s \sim \text{Unif}(B)$, tested with `chisq.test` ($p > 0.05$ if well-calibrated). Model i is better calibrated than model j if $s_i < s_j$.

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-calibration. If `chisq = FALSE` and m is the predicted value then you can manually compute the p.value with `pchisq(m, 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")
```

Meta Information

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

Super classes

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


Active bindings

B (integer(1))

Number of buckets to test for uniform predictions over. Default of 10 is recommended by Haider et al. (2020).

chisq (logical(1))

If TRUE returns the p.value of the corresponding chisq.test instead of the measure. Otherwise this can be performed manually with `pchisq(m,B -1,lower.tail = FALSE)`. $p > 0.05$ indicates well-calibrated.

Methods**Public methods:**

- [MeasureSurvDCalibration\\$new\(\)](#)
- [MeasureSurvDCalibration\\$clone\(\)](#)

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

Usage:

```
MeasureSurvDCalibration$new(B = 10L, chisq = FALSE)
```

Arguments:

B (integer(1))

Number of buckets to test for uniform predictions over. Default of 10 is recommended by Haider et al. (2020).

chisq (logical(1))

If TRUE returns the p.value of the corresponding chisq.test instead of the measure. Otherwise this can be performed manually with `pchisq(m,B -1,lower.tail = FALSE)`. $p > 0.05$ indicates well-calibrated.

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_tr](#), [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 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.schmid

mlr_measures_surv.graf

Integrated Graf Score Survival Measure

Description

Calculates the Integrated Graf Score, aka integrated Brier score or squared loss.

For an individual who dies at time t , with predicted Survival function, S , the Graf Score at time t^* is given by

$$L(S, t|t^*) = [(S(t^*))^2 I(t \leq t^*, \delta = 1)(1/G(t))] + [((1 - S(t^*))^2) I(t > t^*)(1/G(t^*))]$$

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

The re-weighted IGS, IGS* is given by

$$L(S, t|t^*) = [(S(t^*))^2 I(t \leq t^*, \delta = 1)(1/G(t))] + [((1 - S(t^*))^2) I(t > t^*)(1/G(t))]$$

nolint where G is the Kaplan-Meier estimate of the censoring distribution, i.e. always weighted by $G(t)$. IGS* is strictly proper when the censoring distribution is independent of the survival distribution and when G is fit on a sufficiently large dataset. IGS is never proper. Use proper = FALSE for IGS and proper = TRUE for IGS*, in the future the default will be changed to proper = TRUE. Results may be very different if many observations are censored at the last observed time due to division by $1/\text{eps}$ in proper = TRUE.

Note: 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).

If integrated == FALSE then the sample mean is taken for the single specified times, t^* , and the returned score is given by

$$L(S, t|t^*) = \frac{1}{N} \sum_{i=1}^N L(S_i, t_i|t^*)$$

where N is the number of observations, S_i is the predicted survival function for individual i and t_i is their true survival time.

If integrated == TRUE then an approximation to integration is made by either taking the sample mean over all T unique time-points (method == 1), or by taking a mean weighted by the difference between time-points (method == 2). Then the sample mean is taken over all N observations.

$$L(S) = \frac{1}{NT} \sum_{i=1}^N \sum_{j=1}^T L(S_i, t_i|t_j^*)$$

Details

If `task` and `train_set` are passed to `$score` then `G` 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.

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")
```

Meta Information

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

Super classes

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

Active bindings

```
se (logical(1))
  If TRUE returns the standard error of the measure.
```

Methods**Public methods:**

- [MeasureSurvGraf\\$new\(\)](#)
- [MeasureSurvGraf\\$clone\(\)](#)

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

Usage:

```
MeasureSurvGraf$new(
  integrated = TRUE,
  times,
  method = 2,
  se = FALSE,
  proper = FALSE,
  eps = 0.001
)
```

Arguments:

integrated (logical(1))
 If TRUE (default), returns the integrated score; otherwise, not integrated.

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

method (integer(1))
 If integrate == TRUE 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 default to be in line with other packages.

se (logical(1))
 If TRUE returns the standard error of the measure.

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 to weight the measure. If FALSE then weights scores by the Graf method which is the more common usage but the loss is not proper. In v0.5.0, the default will be changed to TRUE.

eps (numeric(1))
 Very small number to set zero-valued predicted probabilities to in order to prevent errors in log(0) and 1/0 calculation.

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](https://doi.org/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.rmse](#), [mlr_measures_surv.schmid](#), [mlr_measures_surv.song_auc](#), [mlr_measures_surv.song_tpr](#), [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.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.schmid](#)

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 these measures will all 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")
```

Meta Information

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

Super classes

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

Methods

Public methods:

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

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

Usage:

```
MeasureSurvHungAUC$new(integrated = TRUE, times)
```

Arguments:

integrated (logical(1))

If TRUE (default), returns the integrated score; otherwise, not integrated.

times (numeric())

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

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.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 loss Survival Measure

Description

Calculates the integrated survival logarithmic (log) (ISLL), loss, aka integrated cross entropy.

For an individual who dies at time t , with predicted Survival function, S , the probabilistic log loss at time t^* is given by

$$L(S, t|t^*) = -[\log(1 - S(t^*))I(t \leq t^*, \delta = 1)(1/G(t))] - [\log(S(t^*))I(t > t^*)(1/G(t^*))]$$

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

The re-weighted ISLL, ISLL* is given by

$$L(S, t|t^*) = -[\log(1 - S(t^*))I(t \leq t^*, \delta = 1)(1/G(t))] - [\log(S(t^*))I(t > t^*)(1/G(t))]$$

nolint where G is the Kaplan-Meier estimate of the censoring distribution, i.e. always weighted by $G(t)$. ISLL* is strictly proper when the censoring distribution is independent of the survival distribution and when G is fit on a sufficiently large dataset. ISLL is never proper. Use proper = FALSE for ISLL and proper = TRUE for ISLL*, in the future the default will be changed to proper = TRUE. Results may be very different if many observations are censored at the last observed time due to division by $1/\text{eps}$ in proper = TRUE.

If integrated == FALSE then the sample mean is taken for the single specified times, t^* , and the returned score is given by

$$L(S, t|t^*) = \frac{1}{N} \sum_{i=1}^N L(S_i, t_i|t^*)$$

where N is the number of observations, S_i is the predicted survival function for individual i and t_i is their true survival time.

If integrated == TRUE then an approximation to integration is made by either taking the sample mean over all T unique time-points (method == 1), or by taking a mean weighted by the difference between time-points (method == 2). Then the sample mean is taken over all N observations.

$$L(S) = \frac{1}{NT} \sum_{i=1}^N \sum_{j=1}^T L(S_i, t_i|t_j^*)$$

Details

If task and train_set are passed to \$score then G 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.

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")
```

Meta Information

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

Super classes

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

Active bindings

se (logical(1))
If TRUE returns the standard error of the measure.

Methods**Public methods:**

- [MeasureSurvIntLogloss\\$new\(\)](#)
- [MeasureSurvIntLogloss\\$clone\(\)](#)

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

Usage:

```
MeasureSurvIntLogloss$new(
  integrated = TRUE,
  times,
  eps = 0.000000000000001,
  method = 2,
  se = FALSE,
  proper = FALSE
)
```

Arguments:

integrated (logical(1))

If TRUE (default), returns the integrated score; otherwise, not integrated.

times (numeric())

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

eps (numeric(1))

Very small number to set zero-valued predicted probabilities to in order to prevent errors in $\log(0)$ and $1/0$ calculation.

method (integer(1))

If integrate == TRUE 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 default to be in line with other packages.

se (logical(1))

If TRUE returns the standard error of the measure.

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 to weight the measure. If FALSE then weights scores by the Graf method which is the more common usage but the loss is not proper. In v0.5.0, the default will be changed to TRUE.

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](https://doi.org/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.rmse](#), [mlr_measures_surv.schmid](#), [mlr_measures_surv.song_auc](#), [mlr_measures_surv.song_tpr](#), [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.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.schmid](#)

mlr_measures_surv.logloss

Log loss Survival Measure

Description

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

The logloss, in the context of probabilistic predictions, is defined as the negative log probability density function, f , evaluated at the observation time, t ,

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

The standard error of the Logloss, 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 IPCW log loss is defined by

$$L(f, t, \Delta) = -\Delta \log(f(t))/G(t)$$

where Δ is the censoring indicator and G is the Kaplan-Meier estimator of the censoring distribution.

Details

If `task` and `train_set` are passed to `$score` then G 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.

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")
```

Meta Information

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

Super classes

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

Active bindings

```
eps (numeric(1))
  Very small number used to prevent log(0) and 1/0 error.
se (logical(1))
  If TRUE returns the standard error of the measure.
rm_cens (logical(1))
  Deprecated, please use IPCW instead.
IPCW (logical(1))
  If TRUE (default) removes censored observations and weights score with IPC weighting calculated from the survival probability of the censoring distribution at the time of death.
```

Methods

Public methods:

- [MeasureSurvLogloss\\$new\(\)](#)
- [MeasureSurvLogloss\\$clone\(\)](#)

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

Usage:

```
MeasureSurvLogloss$new(
  eps = 0.000000000000001,
  se = FALSE,
  rm_cens = TRUE,
  IPCW = TRUE
)
```

Arguments:

`eps` (numeric(1))

Very small number to set zero-valued predicted probabilities to in order to prevent errors in $\log(0)$ and $1/0$ calculation.

`se` (logical(1))

If TRUE returns the standard error of the measure.

`rm_cens` (logical(1))

Deprecated, please use IPCW instead.

`IPCW` (logical(1))

If TRUE (default) removes censored observations and weights score with IPC weighting calculated from the survival probability of the censoring distribution at the time of death.

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_auc](#), [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.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.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")
```

Meta Information

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

Super classes

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

Active bindings

```
se (logical(1))
  If TRUE returns the standard error of the measure.
```

Methods

Public methods:

- [MeasureSurvMAE\\$new\(\)](#)
- [MeasureSurvMAE\\$clone\(\)](#)

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

Usage:

```
MeasureSurvMAE$new(se = FALSE)
```

Arguments:

```
se (logical(1))
```

If TRUE returns the standard error of the measure.

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.rmse`, `mlr_measures_surv.schmid`, `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 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")
```

Meta Information

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

Super classes

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

Active bindings

`se` (logical(1))
If TRUE returns the standard error of the measure.

Methods**Public methods:**

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

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

Usage:

```
MeasureSurvMSE$new(se = FALSE)
```

Arguments:

`se` (logical(1))
If TRUE returns the standard error of the measure.

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.oquigley`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tpr`, `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 these measures will all 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")
```

Meta Information

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

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.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 these measures will all 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")
```


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](https://doi.org/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.rmse`, `mlr_measures_surv.schmid`, `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 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.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")
```

Meta Information

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

Super classes

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

Active bindings

```
se (logical(1))
  If TRUE returns the standard error of the measure.
```

Methods

Public methods:

- [MeasureSurvRMSE\\$new\(\)](#)
- [MeasureSurvRMSE\\$clone\(\)](#)

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

Usage:

```
MeasureSurvRMSE$new(se = FALSE)
```

Arguments:

```
se (logical(1))
```

If TRUE returns the standard error of the measure.

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.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.mse`

```
mlr_measures_surv.schmid
```

Integrated Schmid Score Survival Measure

Description

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

For an individual who dies at time t , with predicted Survival function, S , the Schmid Score at time t^* is given by

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

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

The re-weighted ISS, ISS* is given by

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

nolint where G is the Kaplan-Meier estimate of the censoring distribution, i.e. always weighted by $G(t)$. ISS* is strictly proper when the censoring distribution is independent of the survival distribution and when G is fit on a sufficiently large dataset. ISS is never proper. Use `proper = FALSE` for ISS and `proper = TRUE` for ISS*, in the future the default will be changed to `proper =`

TRUE. Results may be very different if many observations are censored at the last observed time due to division by $1/\text{eps}$ in `proper = TRUE`.

If `integrated == FALSE` then the sample mean is taken for the single specified times, t^* , and the returned score is given by

$$L(S, t|t^*) = \frac{1}{N} \sum_{i=1}^N L(S_i, t_i|t^*)$$

where N is the number of observations, S_i is the predicted survival function for individual i and t_i is their true survival time.

If `integrated == TRUE` then an approximation to integration is made by either taking the sample mean over all T unique time-points (`method == 1`), or by taking a mean weighted by the difference between time-points (`method == 2`). Then the sample mean is taken over all N observations.

$$L(S) = \frac{1}{NT} \sum_{i=1}^N \sum_{j=1}^T L(S_i, t_i|t_j^*)$$

Details

If `task` and `train_set` are passed to `$score` then `G` 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.

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")
```

Meta Information

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

Super classes

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

Active bindings

```
se (logical(1))
  If TRUE returns the standard error of the measure.
```

Methods

Public methods:

- [MeasureSurvSchmid\\$new\(\)](#)
- [MeasureSurvSchmid\\$clone\(\)](#)

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

Usage:

```
MeasureSurvSchmid$new(
  integrated = TRUE,
  times,
  method = 2,
  se = FALSE,
  proper = FALSE,
  eps = 0.001
)
```

Arguments:

`integrated` (logical(1))

If TRUE (default), returns the integrated score; otherwise, not integrated.

`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.

`method` (integer(1))

If `integrated == TRUE` 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 default to be in line with other packages.

`se` (logical(1))

If TRUE returns the standard error of the measure.

`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 to weight the measure. If FALSE then weights scores by the Graf method which is the more common usage but the loss is not proper. In v0.5.0, the default will be changed to TRUE.

`eps` (numeric(1))

Very small number to set zero-valued predicted probabilities to in order to prevent errors in $\log(0)$ and $1/0$ calculation.

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](https://doi.org/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](https://doi.org/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.rmse](#), [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.logloss](#)

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.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 these measures will all 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")
```

Meta Information

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

Super classes

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

Active bindings

```
type (character(1))
  Type of measure, one of: 'cumulative', 'incident'.
```

Methods**Public methods:**

- [MeasureSurvSongAUC\\$new\(\)](#)
- [MeasureSurvSongAUC\\$clone\(\)](#)

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

Usage:

```
MeasureSurvSongAUC$new(
  integrated = TRUE,
  times,
  type = c("incident", "cumulative")
)
```

Arguments:

```
integrated (logical(1))
  If TRUE (default), returns the integrated score; otherwise, not integrated.
times (numeric())
  If integrate == TRUE then a vector of time-points over which to integrate the score. If
  integrate == FALSE then a single time point at which to return the score.
type (character(1))
  Determines the type of score, one of: 'cumulative', 'incident'.
```

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

mlr3probasong_2008

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_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_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 these measures will all 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")
```


Meta Information

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

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> mlr3proba::MeasureSurvIntegrated -> mlr3proba::MeasureSurvAU
-> MeasureSurvSongTNR

Active bindings

lp_thresh numeric(1)
Threshold for linear predictor when calculating TPR/TNR.

Methods**Public methods:**

- [MeasureSurvSongTNR\\$new\(\)](#)
- [MeasureSurvSongTNR\\$clone\(\)](#)

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

Usage:

```
MeasureSurvSongTNR$new(times = 0, lp_thresh = 0)
```

Arguments:

times numeric()

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

lp_thresh numeric(1)

Determines where to threshold the linear predictor for calculating the TPR/TNR.

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.rmse`, `mlr_measures_surv.schmid`, `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 these measures will all 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")
```

Meta Information

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

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> mlr3proba::MeasureSurvIntegrated -> mlr3proba::MeasureSurvAU
 -> MeasureSurvSongTPR

Active bindings

lp_thresh numeric(1)
 Threshold for linear predictor when calculating TPR/TNR.

type (character(1))
 Type of measure, one of: 'cumulative', 'incident'.

Methods**Public methods:**

- [MeasureSurvSongTPR\\$new\(\)](#)
- [MeasureSurvSongTPR\\$clone\(\)](#)

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

Usage:

```
MeasureSurvSongTPR$new(
  times = 0,
  lp_thresh = 0,
  type = c("incident", "cumulative")
)
```

Arguments:

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

`lp_thresh` numeric(1)
 Determines where to threshold the linear predictor for calculating the TPR/TNR.

`type` (character(1))
 Determines the type of score, one of: 'cumulative', 'incident'.

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.rmse`, `mlr_measures_surv.schmid`, `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 these measures will all 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")
```

Meta Information

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

Super classes

mlr3::Measure -> mlr3proba::MeasureSurv -> mlr3proba::MeasureSurvIntegrated -> mlr3proba::MeasureSurvUnoAUC
 -> MeasureSurvUnoAUC

Methods**Public methods:**

- `MeasureSurvUnoAUC$new()`
- `MeasureSurvUnoAUC$clone()`

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

Usage:

```
MeasureSurvUnoAUC$new(integrated = TRUE, times)
```

Arguments:

`integrated` (logical(1))

If TRUE (default), returns the integrated score; otherwise, not integrated.

`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.

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 for Year Survivors With Censored Regression Models.” *Journal of the American Statistical Association*, **102**(478), 527–537. doi: [10.1198/016214507000000149](https://doi.org/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.rmse`, `mlr_measures_surv.schmid`, `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 these measures will all 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")
```

Meta Information

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

Super classes

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

Active bindings

```
lp_thresh numeric(1)
  Threshold for linear predictor when calculating TPR/TNR.
```

Methods

Public methods:

- [MeasureSurvUnoTNR\\$new\(\)](#)
- [MeasureSurvUnoTNR\\$clone\(\)](#)

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

Usage:

```
MeasureSurvUnoTNR$new(times = 0, lp_thresh = 0)
```

Arguments:

`times` `numeric()`

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

`lp_thresh` `numeric(1)`

Determines where to threshold the linear predictor for calculating the TPR/TNR.

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 for Year Survivors With Censored Regression Models.” *Journal of the American Statistical Association*, **102**(478), 527–537. doi: [10.1198/016214507000000149](https://doi.org/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.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 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](#)

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](#)

```
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 these measures will all 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")
```

Meta Information

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

Super classes

```
mlr3::Measure -> mlr3proba::MeasureSurv -> mlr3proba::MeasureSurvIntegrated -> mlr3proba::MeasureSurvAU
-> MeasureSurvUnoTPR
```

Active bindings

```
lp_thresh numeric(1)
  Threshold for linear predictor when calculating TPR/TNR.
```


Methods

Public methods:

- [MeasureSurvUnoTPR\\$new\(\)](#)
- [MeasureSurvUnoTPR\\$clone\(\)](#)

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

Usage:

```
MeasureSurvUnoTPR$new(times = 0, lp_thresh = 0)
```

Arguments:

`times` `numeric()`

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

`lp_thresh` `numeric(1)`

Determines where to threshold the linear predictor for calculating the TPR/TNR.

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 for Year Survivors With Censored Regression Models.” *Journal of the American Statistical Association*, **102**(478), 527–537. doi: [10.1198/016214507000000149](https://doi.org/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.rmse](#), [mlr_measures_surv.schmid](#), [mlr_measures_surv.song_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.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::X0()`.

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 these measures will all 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")
```

Meta Information

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

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](https://doi.org/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.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](#)

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_crank
```

```
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 "pred" 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. One of median, mode, or mean corresponding to the respective functions in the predicted survival distribution. Note that for models with a proportional hazards form, the ranking implied by mean and median will be identical (but not the value of crank itself). Default is mean.
- `which` :: numeric(1)
If `method = "mode"` then specifies which mode to use if multi-modal, default is the first.
- `response` :: logical(1)
If TRUE then the response predict type is estimated with the same values as crank.
- `overwrite` :: logical(1)
If FALSE (default) then if the "pred" input already has a crank, the compositor only composes a response type if `response = TRUE` and does not already exist. If TRUE then both the crank and response are overwritten.

Internals

The median, mode, or mean will use analytical expressions if possible but if not they are calculated using methods from **distr6**. mean requires **cubature**.

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 = "compose_crank",
  param_vals = list(method = "mean", response = FALSE, overwrite = FALSE)
)

```

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.

See Also

[pipeline_crankcompositor](#)

Other survival compositors: [mlr_pipeops_compose_distr](#)

Examples

```

## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  library(mlr3)
  library(mlr3pipelines)
  task = tsk("rats")

  learn = lrn("surv.coxph")$train(task)$predict(task)
  poc = po("crankcompose", param_vals = list(method = "median"))
  poc$predict(list(learn))[[1]]

  if (requireNamespace("cubature", quietly = TRUE)) {
    learn = lrn("surv.coxph")$train(task)$predict(task)
    poc = po("crankcompose", param_vals = list(method = "mean"))
    poc$predict(list(learn))[[1]]
  }
}

## End(Not run)

```

```
mlr_pipeops_compose_distr
      PipeOpDistrCompositor
```

Description

Estimates (or 'composes') a survival distribution from a predicted baseline `distr` and a crank or `lp` from two [PredictionSurv](#)s.

Compositor Assumptions:

- The baseline `distr` is a discrete estimator, e.g. [surv.kaplan](#).
- The composed `distr` is of a linear form
- If `lp` is missing then crank is equivalent

These assumptions are strong and may not be reasonable. Future updates will upgrade this compositor to be more flexible.

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 `distr` predict type; which is composed from the `distr` of "base" and `lp` or crank of "pred".

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/crank - this is useful for changing the prediction distr from one model form to another.

Internals

The respective forms above have respective survival distributions:

$$aft : S(t) = S_0\left(\frac{t}{\exp(lp)}\right)$$

$$ph : S(t) = S_0(t)^{\exp(lp)}$$

$$po : S(t) = \frac{S_0(t)}{\exp(-lp) + (1 - \exp(-lp))S_0(t)}$$

nolint where S_0 is the estimated baseline survival distribution, and lp is the predicted linear predictor. If the input model does not predict a linear predictor then crank is assumed to be the lp - **this may be a strong and unreasonable assumption.**

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 = "compose_distr",
  param_vals = list(form = "aft", overwrite = FALSE)
)
```

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.

See Also

[pipeline_distrcompositor](#)

Other survival compositors: [mlr_pipeops_compose_crank](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  library(mlr3)
  library(mlr3pipelines)
  task = tsk("rats")

  base = lrn("surv.kaplan")$train(task)$predict(task)
  pred = lrn("surv.coxph")$train(task)$predict(task)
  pod = po("distrcompose", param_vals = list(form = "aft", overwrite = TRUE))
  pod$predict(list(base = base, pred = pred))[[1]]
}

## End(Not run)
```

```
mlr_pipeops_compose_probregr
  PipeOpProbregrCompositor
```

Description

Combines a predicted reponse 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()`:

```
PipeOpProbregrCompositor$new()
mlr_pipeops$get("compose_probregr")
po("compose_probregr")
```

Input and Output Channels

[PipeOpProbregrCompositor](#) 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 "Normal" (default), "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 -> PipeOpProbregrCompositor
```

Methods**Public methods:**

- `PipeOpProbregrCompositor$new()`
- `PipeOpProbregrCompositor$clone()`

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

Usage:

```
PipeOpProbregrCompositor$new(
  id = "compose_probregr",
  param_vals = list(dist = "Normal")
)
```

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:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE) &&
    requireNamespace("rpart", quietly = TRUE)) {
  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_survavg *PipeOpSurvAvg*

Description

Perform (weighted) prediction averaging from survival [PredictionSurv](#)s 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_regr](#), [mlr_pipeops_trafopred_survregr](#), [mlr_pipeops_trafotask_regrsurv](#), [mlr_pipeops_trafotask_survregr](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  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_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_survregr](#), [mlr_pipeops_trafotask_regrsurv](#), [mlr_pipeops_trafotask_survregr](#)

Other Transformation PipeOps: [mlr_pipeops_trafopred_survregr](#), [mlr_pipeops_trafotask_regrsurv](#), [mlr_pipeops_trafotask_survregr](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  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_regrsurv](#), [mlr_pipeops_trafotask_regrsurv](#), [mlr_pipeops_trafotask_survregr](#)

Other Transformation PipeOps: [mlr_pipeops_trafopred_regrsurv](#), [mlr_pipeops_trafotask_regrsurv](#), [mlr_pipeops_trafotask_survregr](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  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 0/1s 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_regrsurv](#), [mlr_pipeops_trafopred_survreg](#), [mlr_pipeops_trafotask_survreg](#)

Other Transformation PipeOps: [mlr_pipeops_trafopred_regrsurv](#), [mlr_pipeops_trafopred_survreg](#), [mlr_pipeops_trafotask_survreg](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  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_survreg", 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_survreg
      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 is column is removed and a weights column is added, weights are inverse estimated survival probability of the censoring distribution evaluated at survival time; "mr1" (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 "mr1" and "ipcw" will perform worse with Type I censoring.

- estimator::(character(1))
Method for calculating censoring weights or mean residual lifetime in "mr1", 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 cnetr, 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_survreg", 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://doi.org/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://doi.org/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_regrsurv](#), [mlr_pipeops_trafopred_survreg](#), [mlr_pipeops_trafotask_regrsurv](#)

Other Transformation PipeOps: [mlr_pipeops_trafopred_regrsurv](#), [mlr_pipeops_trafopred_survreg](#), [mlr_pipeops_trafotask_regrsurv](#)

Examples

```
## Not run:
if (requireNamespace("mlr3pipelines", quietly = TRUE)) {
  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_survreg", method = "delete")
  po$train(list(task, NULL))[[1]] # 42 deleted

  # omission
  po = po("trafotask_survreg", method = "omit")
  po$train(list(task, NULL))[[1]]

  if (requireNamespace("mlr3extralearners", quietly = TRUE)) {
    # ipcw with Akritas
    po = po("trafotask_survreg", 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_survreg", 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](#).

Details

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 or death.

Construction

```

mlr3::mlr_tasks$get("actg")
mlr3::tsk("actg")

```

mlr_tasks_faithful	<i>Old Faithful Eruptions Density Task</i>
--------------------	--

Description

A density task for the [faithful](#) data set.

Format

[R6::R6Class](#) inheriting from [TaskDens](#).

Details

Only the eruptions column is kept in this task.

Construction

```
mlr3::mlr_tasks$get("faithful")  
mlr3::tsk("faithful")
```

mlr_tasks_gbcs	<i>German Breast Cancer Study Survival Task</i>
----------------	---

Description

A survival task for the [gbcs](#) data set.

Format

[R6::R6Class](#) inheriting from [TaskSurv](#).

Details

Column "id" and all date columns removed, as well as "rectime" and "censrec". Target is time to death.

Construction

```
mlr3::mlr_tasks$get("gbcs")  
mlr3::tsk("gbcs")
```

mlr_tasks_grace	<i>GRACE 1000 Survival Task</i>
-----------------	---------------------------------

Description

A survival task for the [grace](#) data set.

Format

[R6::R6Class](#) inheriting from [TaskSurv](#).

Details

Column "id" is removed. Columns "days" and "death" renamed to "time" and "status" resp.

Construction

```
mlr3::mlr_tasks$get("grace")
mlr3::tsk("grace")
```

mlr_tasks_lung	<i>Lung Cancer Survival Task</i>
----------------	----------------------------------

Description

A survival task for the [lung](#) data set.

Format

[R6::R6Class](#) inheriting from [TaskSurv](#).

Details

Column "sex" has been converted to a factor, all others have been converted to integer.

Construction

```
mlr3::mlr_tasks$get("lung")
mlr3::tsk("lung")
```

mlr_tasks_precip	<i>Annual Precipitation Density Task</i>
------------------	--

Description

A density task for the [precip](#) data set.

Format

[R6::R6Class](#) inheriting from [TaskDens](#).

Construction

```
mlr3::mlr_tasks$get("precip")  
mlr3::tsk("precip")
```

mlr_tasks_rats	<i>Rats Survival Task</i>
----------------	---------------------------

Description

A survival task for the [rats](#) data set.

Format

[R6::R6Class](#) inheriting from [TaskSurv](#).

Details

Column "sex" has been converted to a factor, all others have been converted to integer.

Construction

```
mlr3::mlr_tasks$get("rats")  
mlr3::tsk("rats")
```

mlr_tasks_unemployment

Unemployment Duration Survival Task

Description

A survival task for the UnempDur data set.

Format

[R6::R6Class](#) inheriting from [TaskSurv](#).

Details

A survival task for the "UnempDur" data set in package **Ecdat**. Contains the following columns of the original data set: "spell" (time), "censor1" (status), "age", "ui", "logwage", and "tenure".

Construction

```
mlr3::mlr_tasks$get("unemployment")
mlr3::tsk("unemployment")
```

See Also

Dictionary of Tasks: [mlr3::mlr_tasks](#)

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](#).

Details

Columns "id", "yrgrp", and "dstat" are removed so target is status at last follow-up. Column "sex" renamed to "sexF", "lenfol" to "time", and "fstat" to "status".

Construction

```
mlr3::mlr_tasks$get("whas")
mlr3::tsk("whas")
```

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.

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")
```

Super class

```
mlr3::TaskGenerator -> TaskGeneratorSimdens
```

Methods

Public methods:

- `TaskGeneratorSimdens$new()`
- `TaskGeneratorSimdens$clone()`

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

Usage:

```
TaskGeneratorSimdens$new()
```

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`

Examples

```
generator = mlr3::mlr_task_generators$get("simdens")  
task = generator$generate(20)  
task$head()
```

 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 data set with the following numerical covariates:

- `treatment`: Bernoulli distributed with log hazard ratio -0.5 .
- `height`: Normally distributed with log hazard ratio 1 .
- `weight`: normally distributed with log hazard ratio 0 .

See `simsurv::simsurv()` for an explanation of the hyperparameters.

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")
```

Super class

```
mlr3::TaskGenerator -> TaskGeneratorSimsurv
```

Methods

Public methods:

- `TaskGeneratorSimsurv$new()`
- `TaskGeneratorSimsurv$clone()`

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

Usage:

```
TaskGeneratorSimsurv$new()
```

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.

See Also

Dictionary of TaskGenerators: [mlr3::mlr_task_generators](#)

Examples

```
if (requireNamespace("simsurv", quietly = TRUE)) {
  generator = mlr3::mlr_task_generators$get("simsurv")
  task = generator$generate(20)
  task$head()
}
```

 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 [LearnerSurv](#)s.

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 = 0.000000000000001,
  train_task = NULL,
```

```

    train_set = NULL,
    proper = TRUE,
    ...
  )

```

Arguments

x	(PredictionSurv or list of LearnerSurv s)
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 <code>Learner\$predict</code> .
newdata	(data.frame()) If not missing <code>Learner\$predict_newdata</code> is called instead of <code>Learner\$predict</code> .
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:
if (requireNamespace("ggplot2", quietly = TRUE)) {
  #' 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$new(
  id,
  param_set = ParamSet$new(),
  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_regr](#), [mlr_pipeops_trafopred_survreg](#), [mlr_pipeops_trafotask_regrsurv](#), [mlr_pipeops_trafotask_survreg](#)

Other Transformers: [PipeOpTaskTransformer](#), [PipeOpTransformer](#)

PipeOpTaskTransformer *PipeOpTaskTransformer*

Description

Parent class for [PipeOps](#) that transform task objects top 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$new(
  id,
  param_set = ParamSet$new(),
  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_regr](#), [mlr_pipeops_trafopred_survregr](#), [mlr_pipeops_trafotask_regrsurv](#), [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$new(
  id,
  param_set = ParamSet$new(),
  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_r](#), [mlr_pipeops_trafopred_survreg](#), [mlr_pipeops_trafotask_regrsurv](#), [mlr_pipeops_trafotask_survreg](#)

Other Transformers: [PipeOpPredTransformer](#), [PipeOpTaskTransformer](#)

plot.LearnerSurv *Visualization of fitted LearnerSurv objects*

Description

Wrapper around `predict.LearnerSurv` and `plot.VectorDistribution`.

Usage

```
## S3 method for class 'LearnerSurv'
plot(
  x,
  task,
  fun = c("survival", "pdf", "cdf", "quantile", "hazard", "cumhazard"),
  row_ids = NULL,
  newdata,
  ...
)
```

Arguments

<code>x</code>	(LearnerSurv)
<code>task</code>	(TaskSurv)
<code>fun</code>	(character) Passed to <code>distr6::plot.VectorDistribution</code>
<code>row_ids</code>	(<code>integer()</code>) Passed to <code>Learner\$predict</code>
<code>newdata</code>	(<code>data.frame()</code>) If not missing <code>Learner\$predict_newdata</code> is called instead of <code>Learner\$predict</code> .
<code>...</code>	Additional arguments passed to <code>distr6::plot.VectorDistribution</code>

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)

```

PredictionDens	<i>Prediction Object for Density</i>
----------------	--------------------------------------

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".

Super class

[mlr3::Prediction](#) -> PredictionSurv

Active bindings

truth (Surv)
True (observed) outcome.

crank (numeric())
Access the stored predicted continuous ranking.

distr ([VectorDistribution](#))
Convert the stored survival 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::VectorDistribution](#)])
Either a matrix of predicted survival probabilities or a [distr6::VectorDistribution](#). If a matrix 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.

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$train(task, row_ids = 1:20)$predict(task, row_ids = 21:30)
head(as.data.table(p))
```

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)
```

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.

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](#)

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", or "mstate".

Methods**Public methods:**

- [TaskSurv\\$new\(\)](#)
- [TaskSurv\\$truth\(\)](#)
- [TaskSurv\\$formula\(\)](#)
- [TaskSurv\\$times\(\)](#)
- [TaskSurv\\$status\(\)](#)
- [TaskSurv\\$unique_times\(\)](#)
- [TaskSurv\\$unique_event_times\(\)](#)
- [TaskSurv\\$risk_set\(\)](#)
- [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")
)
```

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 for interval censored data, otherwise ignored.

`type` (character(1))

Name of the column giving the type of censoring. Default is 'right' censoring.

Method `truth()`: True response for specified `row_ids`. Format depends on the task type. Defaults to all rows with role "use".

Usage:

```
TaskSurv$truth(rows = NULL)
```

Arguments:

```
rows integer()  
  Row indices.
```

Returns: `numeric()`.

Method `formula()`: Creates a formula for survival models with `survival::Surv` on the LHS.

Usage:

```
TaskSurv$formula(rhs = NULL)
```

Arguments:

```
rhs If NULL RHS is ., otherwise gives RHS of formula.
```

Returns: `numeric()`.

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'.

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.

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) at time.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Task: [TaskDens](#)

Examples

```
library(mlr3)
lung = survival::lung
lung$status = (lung$status == 2L)
task = TaskSurv$new("lung",
  backend = lung, time = "time",
  event = "status")

# meta data
task$target_names
task$feature_names
task$formula()

# survival data
task$truth()
task$times()
task$status()
```

```
task$unique_times()
task$unique_event_times()
task$risk_set(time = 700)
```

whas

Worcester Heart Attack Study (WHAS) Dataset

Description

whas dataset from Hosmer et al.

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

ftp://ftp.wiley.com/public/sci_tech_med/survival

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

- * **AUC survival measures**
 - mlr_measures_surv.chambless_auc, 44
 - mlr_measures_surv.hung_auc, 53
 - mlr_measures_surv.song_auc, 70
 - mlr_measures_surv.song_tnr, 72
 - mlr_measures_surv.song_tpr, 74
 - mlr_measures_surv.uno_auc, 76
 - mlr_measures_surv.uno_tnr, 78
 - mlr_measures_surv.uno_tpr, 80
- * **Density estimation measures**
 - mlr_measures_dens.logloss, 39
- * **Ensembles**
 - mlr_pipeops_survavg, 90
- * **Learner**
 - LearnerDens, 11
 - LearnerSurv, 13
- * **Measure**
 - MeasureDens, 15
 - MeasureSurv, 17
- * **PipeOps**
 - mlr_pipeops_survavg, 90
 - mlr_pipeops_trafopred_regrsurv, 92
 - mlr_pipeops_trafopred_survregr, 94
 - mlr_pipeops_trafotask_regrsurv, 95
 - mlr_pipeops_trafotask_survregr, 97
 - PipeOpPredTransformer, 109
 - PipeOpTaskTransformer, 111
 - PipeOpTransformer, 112
- * **Prediction**
 - PredictionDens, 115
 - PredictionSurv, 116
- * **Probabilistic survival measures**
 - mlr_measures_surv.graf, 50
 - mlr_measures_surv.intlogloss, 54
 - mlr_measures_surv.logloss, 57
 - mlr_measures_surv.schmid, 67
- * **R2 survival measures**
 - mlr_measures_surv.nagelk_r2, 63
 - mlr_measures_surv.oquigley_r2, 64
 - mlr_measures_surv.xu_r2, 82
- * **Task**
 - TaskDens, 118
 - TaskSurv, 119
- * **Transformation PipeOps**
 - mlr_pipeops_trafopred_regrsurv, 92
 - mlr_pipeops_trafopred_survregr, 94
 - mlr_pipeops_trafotask_regrsurv, 95
 - mlr_pipeops_trafotask_survregr, 97
- * **Transformers**
 - PipeOpPredTransformer, 109
 - PipeOpTaskTransformer, 111
 - PipeOpTransformer, 112
- * **calibration survival measures**
 - mlr_measures_surv.calib_alpha, 40
 - mlr_measures_surv.calib_beta, 42
 - mlr_measures_surv.dcalib, 48
- * **datasets**
 - actg, 6
 - gbcs, 9
 - grace, 10
 - whas, 123
- * **density estimators**
 - mlr_learners_dens.hist, 33
 - mlr_learners_dens.kde, 34
- * **density measures**
 - mlr_measures_dens.logloss, 39
- * **distr survival measures**
 - mlr_measures_surv.calib_alpha, 40
 - mlr_measures_surv.dcalib, 48
 - mlr_measures_surv.graf, 50
 - mlr_measures_surv.intlogloss, 54
 - mlr_measures_surv.logloss, 57
 - mlr_measures_surv.schmid, 67
- * **lp survival measures**
 - mlr_measures_surv.calib_beta, 42
 - mlr_measures_surv.chambless_auc, 44

- mlr_measures_surv.hung_auc, 53
- mlr_measures_surv.nagelk_r2, 63
- mlr_measures_surv.oquigley_r2, 64
- mlr_measures_surv.song_auc, 70
- mlr_measures_surv.song_tnr, 72
- mlr_measures_surv.song_tpr, 74
- mlr_measures_surv.uno_auc, 76
- mlr_measures_surv.uno_tnr, 78
- mlr_measures_surv.uno_tpr, 80
- mlr_measures_surv.xu_r2, 82
- * **pipelines**
 - mlr_graphs_crankcompositor, 22
 - mlr_graphs_distrcompositor, 23
 - mlr_graphs_probregrcompositor, 25
 - mlr_graphs_survaverager, 27
 - mlr_graphs_survbagging, 28
 - mlr_graphs_survtoregr, 29
- * **response survival measures**
 - mlr_measures_surv.mae, 60
 - mlr_measures_surv.mse, 61
 - mlr_measures_surv.rmse, 66
- * **survival compositors**
 - mlr_pipeops_compose_crank, 83
 - mlr_pipeops_compose_distr, 86
- * **survival learners**
 - mlr_learners_surv.coxph, 35
 - mlr_learners_surv.kaplan, 36
 - mlr_learners_surv.rpart, 38
- * **survival measures**
 - mlr_measures_surv.calib_alpha, 40
 - mlr_measures_surv.calib_beta, 42
 - mlr_measures_surv.chambless_auc, 44
 - mlr_measures_surv.cindex, 45
 - mlr_measures_surv.dcalib, 48
 - mlr_measures_surv.graf, 50
 - mlr_measures_surv.hung_auc, 53
 - mlr_measures_surv.intlogloss, 54
 - mlr_measures_surv.logloss, 57
 - mlr_measures_surv.mae, 60
 - mlr_measures_surv.mse, 61
 - mlr_measures_surv.nagelk_r2, 63
 - mlr_measures_surv.oquigley_r2, 64
 - mlr_measures_surv.rmse, 66
 - mlr_measures_surv.schmid, 67
 - mlr_measures_surv.song_auc, 70
 - mlr_measures_surv.song_tnr, 72
 - mlr_measures_surv.song_tpr, 74
 - mlr_measures_surv.uno_auc, 76
 - mlr_measures_surv.uno_tnr, 78
 - mlr_measures_surv.uno_tpr, 80
 - mlr_measures_surv.xu_r2, 82
 - .surv_return, 5
- actg, 6, 100
- as_prediction_surv, 7
- as_task_surv, 8
- assert_surv, 7
- bujar::bujar, 97, 98
- crankcompositor
 - (mlr_graphs_crankcompositor), 22
- data.table::data.table, 110, 112, 113
- DataBackend, 119, 120
- DataBackendDataTable, 120
- dens.logloss, 16
- Dictionary, 104, 105, 107
- dictionary, 15, 17, 33–36, 38, 41, 42, 44, 46, 48, 51, 53, 55, 58, 60, 61, 63, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 83, 86, 88, 105, 106, 118
- distr6::Distribution, 33, 34
- distr6::distrSimulate(), 105
- distr6::MixtureDistribution, 90
- distr6::VectorDistribution, 5, 90, 117
- distr6::WeightedDiscrete, 5
- distrcompositor
 - (mlr_graphs_distrcompositor), 23
- Distribution, 115, 116
- faithful, 101
- gbcs, 9, 101
- grace, 10, 102
- Graph, 22–28, 31
- graphics::hist(), 33
- GraphLearner, 23, 24, 26–28, 31
- Learner, 11–14, 16, 18, 19, 21, 33–36, 38
- LearnerDens, 11, 15, 115
- LearnerDensHistogram
 - (mlr_learners_dens.hist), 33
- LearnerDensKDE (mlr_learners_dens.kde), 34

- LearnerRegr, [25](#), [30](#), [31](#)
- LearnerSurv, [13](#), [13](#), [22](#), [24](#), [27](#), [28](#), [30](#), [31](#), [107](#), [108](#), [114](#), [116](#)
- LearnerSurvCoxPH, [46](#)
- LearnerSurvCoxPH
(mlr_learners_surv.coxph), [35](#)
- LearnerSurvKaplan
(mlr_learners_surv.kaplan), [36](#)
- LearnerSurvRpart
(mlr_learners_surv.rpart), [38](#)
- lrn(), [33–36](#), [38](#)
- lung, [102](#)

- mean(), [16](#), [18](#)
- Measure, [15](#), [17](#), [41](#), [42](#), [44](#), [46](#), [48](#), [51](#), [53](#), [55](#), [58](#), [60](#), [61](#), [63](#), [64](#), [66](#), [68](#), [70](#), [72](#), [74](#), [76](#), [78](#), [80](#), [82](#)
- MeasureDens, [15](#), [18](#)
- MeasureDensLogloss
(mlr_measures_dens.logloss), [39](#)
- MeasureSurv, [17](#), [17](#)
- MeasureSurvAUC, [18](#)
- MeasureSurvBrier
(mlr_measures_surv.graf), [50](#)
- MeasureSurvCalibrationAlpha
(mlr_measures_surv.calib_alpha), [40](#)
- MeasureSurvCalibrationBeta
(mlr_measures_surv.calib_beta), [42](#)
- MeasureSurvChamblessAUC
(mlr_measures_surv.chambless_auc), [44](#)
- MeasureSurvCindex
(mlr_measures_surv.cindex), [45](#)
- MeasureSurvDCalibration
(mlr_measures_surv.dcalib), [48](#)
- MeasureSurvGraf, [108](#)
- MeasureSurvGraf
(mlr_measures_surv.graf), [50](#)
- MeasureSurvHungAUC
(mlr_measures_surv.hung_auc), [53](#)
- MeasureSurvIntegrated, [20](#)
- MeasureSurvIntLogloss, [108](#)
- MeasureSurvIntLogloss
(mlr_measures_surv.intlogloss), [54](#)

- MeasureSurvLogloss
(mlr_measures_surv.logloss), [57](#)
- MeasureSurvMAE (mlr_measures_surv.mae), [60](#)
- MeasureSurvMSE (mlr_measures_surv.mse), [61](#)
- MeasureSurvNage1kR2
(mlr_measures_surv.nage1k_r2), [63](#)
- MeasureSurvOQuigleyR2
(mlr_measures_surv.oquigley_r2), [64](#)
- MeasureSurvRMSE
(mlr_measures_surv.rmse), [66](#)
- MeasureSurvSchmid
(mlr_measures_surv.schmid), [67](#)
- MeasureSurvSongAUC
(mlr_measures_surv.song_auc), [70](#)
- MeasureSurvSongTNR
(mlr_measures_surv.song_tnr), [72](#)
- MeasureSurvSongTPR
(mlr_measures_surv.song_tpr), [74](#)
- MeasureSurvUnoAUC
(mlr_measures_surv.uno_auc), [76](#)
- MeasureSurvUnoTNR
(mlr_measures_surv.uno_tnr), [78](#)
- MeasureSurvUnoTPR
(mlr_measures_surv.uno_tpr), [80](#)
- MeasureSurvXuR2
(mlr_measures_surv.xu_r2), [82](#)
- MixtureDistribution, [29](#)
- mlr3::Learner, [11](#), [13](#), [33](#), [34](#), [36–38](#)
- mlr3::Measure, [15](#), [17](#), [18](#), [20](#), [40–42](#), [44](#), [46](#), [48](#), [51](#), [53](#), [56](#), [58](#), [60](#), [62](#), [63](#), [65](#), [66](#), [68](#), [71](#), [73](#), [75](#), [77](#), [78](#), [80](#), [82](#)
- mlr3::mlr_measures, [15](#), [17](#)
- mlr3::mlr_task_generators, [105](#), [107](#)
- mlr3::mlr_tasks, [104](#), [119](#)
- mlr3::Prediction, [109](#), [115](#), [116](#)
- mlr3::Task, [111](#), [118](#), [119](#)
- mlr3::TaskGenerator, [105](#), [106](#)
- mlr3::TaskSupervised, [119](#)
- mlr3::TaskUnsupervised, [118](#)
- mlr3pipelines::Graph, [22–29](#), [32](#)
- mlr3pipelines::GraphLearner, [23](#), [24](#), [26](#),

- 27, 29, 32
- mlr3pipelines::mlr_pipeops, 83, 86, 88
- mlr3pipelines::PipeOp, 84, 87, 89, 91, 92, 94, 96, 98, 110, 111, 113
- mlr3pipelines::PipeOpEnsemble, 91
- mlr3pipelines::PipeOpLearner, 22, 24, 25, 28
- mlr3pipelines::po(), 83, 86, 88
- mlr3proba (mlr3proba-package), 4
- mlr3proba-package, 4
- mlr3proba::LearnerDens, 33, 34
- mlr3proba::LearnerSurv, 36–38
- mlr3proba::MeasureDens, 40
- mlr3proba::MeasureSurv, 18, 20, 41, 42, 44, 46, 48, 51, 53, 56, 58, 60, 62, 63, 65, 66, 68, 71, 73, 75, 77, 78, 80, 82
- mlr3proba::MeasureSurvAUC, 44, 53, 71, 73, 75, 77, 78, 80
- mlr3proba::MeasureSurvIntegrated, 18, 44, 51, 53, 56, 68, 71, 73, 75, 77, 78, 80
- mlr3proba::PipeOpPredTransformer, 92, 94
- mlr3proba::PipeOpTaskTransformer, 96, 98
- mlr3proba::PipeOpTransformer, 92, 94, 96, 98, 110, 111
- mlr_graphs_crankcompositor, 22, 24, 26, 27, 29, 32
- mlr_graphs_distrcompositor, 23, 23, 26, 27, 29, 32
- mlr_graphs_probregrcompositor, 23, 24, 25, 27, 29, 32
- mlr_graphs_survaverager, 23, 24, 26, 27, 29, 32
- mlr_graphs_survbagging, 23, 24, 26, 27, 28, 32
- mlr_graphs_survtoregr, 23, 24, 26, 27, 29, 29
- mlr_learners, 33–36, 38
- mlr_learners_dens.hist, 33, 35
- mlr_learners_dens.kde, 34, 34
- mlr_learners_surv.coxph, 35, 37, 39
- mlr_learners_surv.kaplan, 36, 36, 39
- mlr_learners_surv.rpart, 36, 37, 38
- mlr_measures, 41, 42, 44, 46, 48, 51, 53, 55, 58, 60, 61, 63, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82
- mlr_measures_dens.logloss, 39
- mlr_measures_surv.brier (mlr_measures_surv.graf), 50
- mlr_measures_surv.calib_alpha, 40, 43, 45, 47, 49, 50, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.calib_beta, 42, 42, 45, 47, 49, 50, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.chambless_auc, 42, 43, 44, 47, 49, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.cindex, 42, 43, 45, 45, 49, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.dcalib, 42, 43, 45, 47, 48, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.graf, 42, 43, 45, 47, 49, 50, 50, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.hung_auc, 42, 43, 45, 47, 49, 52, 53, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.intlogloss, 42, 43, 45, 47, 49, 50, 52, 54, 54, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.logloss, 42, 43, 45, 47, 49, 50, 52, 54, 57, 57, 61, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.mae, 42, 43, 45, 47, 49, 52, 54, 57, 59, 60, 62, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.mse, 42, 43, 45, 47, 49, 52, 54, 57, 59, 61, 61, 64, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.nagelk_r2, 42, 43, 45, 47, 49, 52, 54, 57, 59, 61, 62, 63, 65, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.oquigley_r2, 42, 43, 45, 47, 49, 52, 54, 57, 59, 61, 62, 64, 64, 67, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.rmse, 42, 43, 45, 47, 49, 52, 54, 57, 59, 61, 62, 64, 65, 66, 70, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.schmid, 42, 43, 45, 47, 49, 50, 52, 54, 57, 59, 61, 62, 64, 65,

- 67, 67, 72, 74, 76, 77, 79, 81, 83
- mlr_measures_surv.song_auc, 42, 43, 45, 47, 49, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 70, 74, 76–79, 81, 83
- mlr_measures_surv.song_tnr, 42, 43, 45, 47, 49, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 72, 76–79, 81, 83
- mlr_measures_surv.song_tpr, 42, 43, 45, 47, 50, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 74, 77–79, 81, 83
- mlr_measures_surv.uno_auc, 42, 43, 45, 47, 50, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76, 79, 81, 83
- mlr_measures_surv.uno_tnr, 42, 43, 45, 47, 50, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76–78, 78, 81, 83
- mlr_measures_surv.uno_tpr, 42, 43, 45, 47, 50, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76–79, 80, 83
- mlr_measures_surv.xu_r2, 42, 43, 45, 47, 50, 52, 54, 57, 59, 61, 62, 64, 65, 67, 70, 72, 74, 76–79, 81, 82
- mlr_pipeops_compose_crank, 83, 88
- mlr_pipeops_compose_distr, 85, 86
- mlr_pipeops_compose_probregr, 88
- mlr_pipeops_survavg, 90, 93, 95, 96, 99, 111, 112, 114
- mlr_pipeops_trafopred_regrsurv, 91, 92, 95, 96, 99, 111, 112, 114
- mlr_pipeops_trafopred_survregr, 91, 93, 94, 96, 99, 111, 112, 114
- mlr_pipeops_trafotask_regrsurv, 91, 93, 95, 95, 99, 111, 112, 114
- mlr_pipeops_trafotask_survregr, 91, 93, 95, 96, 97, 111, 112, 114
- mlr_reflections\$learner_predict_types, 12, 14, 16, 18, 21
- mlr_reflections\$learner_properties, 12, 14
- mlr_reflections\$measure_properties, 16, 18, 19, 21
- mlr_reflections\$task_feature_types, 12, 14
- mlr_task_generators, 105, 106
- mlr_task_generators_simdens, 105
- mlr_task_generators_simsurv, 106
- mlr_tasks, 118
- mlr_tasks_actg, 100
- mlr_tasks_faithful, 101
- mlr_tasks_gbcs, 101
- mlr_tasks_grace, 102
- mlr_tasks_lung, 102
- mlr_tasks_precip, 103
- mlr_tasks_rats, 103
- mlr_tasks_unemployment, 104
- mlr_tasks_whas, 104
- msr(), 41, 42, 44, 46, 48, 51, 53, 55, 58, 60, 61, 63, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82
- Normal, 30
- paradox::ParamSet, 12, 14, 16, 17, 110, 112, 113
- pecs, 107
- pipeline_crankcompositor, 85
- pipeline_crankcompositor (mlr_graphs_crankcompositor), 22
- pipeline_distrcompositor, 88
- pipeline_distrcompositor (mlr_graphs_distrcompositor), 23
- pipeline_probregrcompositor (mlr_graphs_probregrcompositor), 25
- pipeline_survaverager (mlr_graphs_survaverager), 27
- pipeline_survbagging (mlr_graphs_survbagging), 28
- pipeline_survtoregr (mlr_graphs_survtoregr), 29
- PipeOp, 29, 83, 86, 88, 109–113
- PipeOpCrankCompositor, 22, 84
- PipeOpCrankCompositor (mlr_pipeops_compose_crank), 83
- PipeOpDistrCompositor, 23, 30, 31, 86
- PipeOpDistrCompositor (mlr_pipeops_compose_distr), 86
- PipeOpEnsemble, 90, 91
- PipeOpLearner, 90
- PipeOpLearnerCV, 31
- PipeOpPredRegrSurv, 31
- PipeOpPredRegrSurv (mlr_pipeops_trafopred_regrsurv), 92

- PipeOpPredSurvRegr
 - (mlr_pipeops_trafopred_survreg),
94
- PipeOpPredTransformer, 91–96, 99, 109,
109, 112, 114
- PipeOpProbregrCompositor, 25, 30, 31, 88
- PipeOpProbregrCompositor
 - (mlr_pipeops_compose_probreg),
88
- PipeOpSubsample, 28, 29
- PipeOpSurvAvg, 27, 28
- PipeOpSurvAvg(mlr_pipeops_survavg), 90
- PipeOpTaskRegrSurv
 - (mlr_pipeops_trafotask_regrsurv),
95
- PipeOpTaskSurvRegr, 30, 31
- PipeOpTaskSurvRegr
 - (mlr_pipeops_trafotask_survreg),
97
- PipeOpTaskTransformer, 91, 93, 95–97, 99,
111, 111, 114
- PipeOpTransformer, 91, 93, 95, 96, 99, 111,
112, 112, 113
- plot.LearnerSurv, 114
- precip, 103
- Prediction, 11, 13, 109, 112, 113
- PredictionDens, 11, 115, 118
- PredictionRegr, 31, 88, 92, 94
- PredictionSurv, 7, 8, 13, 31, 83, 84, 86, 87,
90, 92, 94, 107, 108, 116, 116
- R6, 12, 14, 15, 17, 19, 33, 34, 36–38, 40, 41,
43, 44, 49, 51, 53, 56, 59, 60, 62, 63,
65, 66, 69, 71, 73, 75, 77, 79, 81, 82,
84, 87, 89, 91, 93, 94, 96, 98, 105,
106, 110, 111, 113, 115, 117, 119,
120
- R6::R6Class, 100–104
- rats, 103
- requireNamespace(), 12, 14, 16, 18, 21, 110,
112, 113
- Resampling, 16, 18, 19, 21
- rpart::predict.rpart(), 38
- rpart::rpart(), 38
- simsurv::simsurv(), 106
- surv.cindex, 18
- surv.kaplan, 86
- survAUC::AUC.cd(), 44
- survAUC::AUC.hc(), 53
- survAUC::AUC.sh(), 70
- survAUC::AUC.uno(), 76
- survAUC::Nagelk(), 63
- survAUC::OXS(), 64
- survAUC::sens.sh(), 74
- survAUC::sens.uno(), 80
- survAUC::spec.sh(), 72
- survAUC::spec.uno(), 78
- survAUC::X0(), 82
- survival::concordance, 46
- survival::coxph(), 35
- survival::predict.coxph(), 35
- survival::Surv, 7, 121
- survival::survfit(), 36
- survival::survfit.coxph(), 35
- Task, 16, 18, 19, 21, 112, 113
- TaskDens, 101, 103, 118, 122
- TaskGenerator, 105, 106
- TaskGenerators, 105, 107
- TaskGeneratorSimdens
 - (mlr_task_generators_simdens),
105
- TaskGeneratorSimsurv
 - (mlr_task_generators_simsurv),
106
- TaskRegr, 30, 31, 95, 97
- Tasks, 104
- TaskSurv, 8, 31, 95, 97, 100–104, 108, 114,
116, 117, 119, 119
- TaskUnsupervised, 118
- tgen(), 105, 106
- VectorDistribution, 117
- whas, 104, 123