

# Package ‘mlr3proba’

August 28, 2020

**Title** Probabilistic Supervised Learning for 'mlr3'

**Version** 0.2.1

**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** R (>= 3.5.0)

**Imports** checkmate, data.table, distr6 (>= 1.4.3), mlr3 (>= 0.5.0), mlr3misc (>= 0.1.7), paradox (>= 0.1.0), R6, Rcpp (>= 1.0.4), survival

**Suggests** bibtex, ggplot2, knitr, lgr, Matrix, mlr3pipelines (>= 0.2.1), mlr3tuning, pracma, rmarkdown, rpart, set6 (>= 0.1.7), simsurv, survAUC, testthat

**LinkingTo** Rcpp

**VignetteBuilder** knitr

**RdMacros** mlr3misc

**ByteCompile** true

**Encoding** UTF-8

**LazyData** true

**NeedsCompilation** yes

**RoxygenNote** 7.1.0.9000

**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'

'MeasureSurvBeggC.R' 'MeasureSurvCalibrationAlpha.R'  
 'MeasureSurvCalibrationBeta.R' 'MeasureSurvChamblessAUC.R'  
 'MeasureSurvCindex.R' 'MeasureSurvGonenHellersC.R'  
 'MeasureSurvGraf.R' 'MeasureSurvGrafSE.R'  
 'MeasureSurvHarrellC.R' 'MeasureSurvHungAUC.R'  
 'MeasureSurvIntLogloss.R' 'MeasureSurvIntLoglossSE.R'  
 'MeasureSurvLogloss.R' 'MeasureSurvLoglossSE.R'  
 'MeasureSurvMAE.R' 'MeasureSurvMAESE.R' 'MeasureSurvMSE.R'  
 'MeasureSurvMSESE.R' 'MeasureSurvNagelkR2.R'  
 'MeasureSurvOQuigleyR2.R' 'MeasureSurvRMSE.R'  
 'MeasureSurvRMSESE.R' 'MeasureSurvSchmid.R'  
 'MeasureSurvSongAUC.R' 'MeasureSurvSongTNR.R'  
 'MeasureSurvSongTPR.R' 'MeasureSurvUnoAUC.R'  
 'MeasureSurvUnoC.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' 'PredictionDens.R' 'PredictionSurv.R'  
 'RcppExports.R' 'TaskDens.R' 'TaskDens\_X.R'  
 'TaskGeneratorSimdens.R' 'TaskGeneratorSimsurv.R' 'TaskSurv.R'  
 'TaskSurv\_X.R' 'assertions.R' 'cindex.R' 'data.R' 'helpers.R'  
 'histogram.R' 'integrated\_scores.R' 'pecs.R' 'pipelines.R'  
 'plot.R' 'surv\_measures.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 <[raphael.sonabend.15@uc1.ac.uk](mailto:raphael.sonabend.15@uc1.ac.uk)>

**Repository** CRAN

**Date/Publication** 2020-08-28 11:10:08 UTC

## R topics documented:

mlr3proba-package . . . . .	4
actg . . . . .	5
gbcs . . . . .	6
grace . . . . .	7
LearnerDens . . . . .	7
LearnerSurv . . . . .	9
MeasureDens . . . . .	11
MeasureSurv . . . . .	13
MeasureSurvAUC . . . . .	14
MeasureSurvIntegrated . . . . .	16

mlr_graphs_crankcompositor . . . . .	17
mlr_graphs_distrcompositor . . . . .	19
mlr_graphs_probregrcompositor . . . . .	20
mlr_graphs_survaverager . . . . .	22
mlr_graphs_survbagging . . . . .	23
mlr_graphs_survtoregr . . . . .	25
mlr_learners_dens.hist . . . . .	28
mlr_learners_dens.kde . . . . .	29
mlr_learners_surv.coxph . . . . .	31
mlr_learners_surv.kaplan . . . . .	32
mlr_learners_surv.rpart . . . . .	33
mlr_measures_dens.logloss . . . . .	35
mlr_measures_surv.beggC . . . . .	36
mlr_measures_surv.calib_alpha . . . . .	37
mlr_measures_surv.calib_beta . . . . .	39
mlr_measures_surv.chambless_auc . . . . .	41
mlr_measures_surv.cindex . . . . .	43
mlr_measures_surv.gonenC . . . . .	45
mlr_measures_surv.graf . . . . .	47
mlr_measures_surv.grafSE . . . . .	49
mlr_measures_surv.harrellC . . . . .	51
mlr_measures_surv.hung_auc . . . . .	52
mlr_measures_surv.intlogloss . . . . .	54
mlr_measures_surv.intloglossSE . . . . .	57
mlr_measures_surv.logloss . . . . .	59
mlr_measures_surv.loglossSE . . . . .	61
mlr_measures_surv.mae . . . . .	62
mlr_measures_surv.maeSE . . . . .	64
mlr_measures_surv.mse . . . . .	65
mlr_measures_surv.mseSE . . . . .	67
mlr_measures_surv.nagelk_r2 . . . . .	68
mlr_measures_surv.oquigley_r2 . . . . .	70
mlr_measures_surv.rmse . . . . .	71
mlr_measures_surv.rmseSE . . . . .	73
mlr_measures_surv.schmid . . . . .	74
mlr_measures_surv.song_auc . . . . .	77
mlr_measures_surv.song_tnr . . . . .	79
mlr_measures_surv.song_tpr . . . . .	81
mlr_measures_surv.unoC . . . . .	83
mlr_measures_surv.uno_auc . . . . .	84
mlr_measures_surv.uno_tnr . . . . .	86
mlr_measures_surv.uno_tpr . . . . .	88
mlr_measures_surv.xu_r2 . . . . .	90
mlr_pipeops_survavg . . . . .	92
mlr_pipeops_trafopred_regrsurv . . . . .	93
mlr_pipeops_trafopred_survregr . . . . .	95
mlr_pipeops_trafotask_regrsurv . . . . .	97
mlr_pipeops_trafotask_survregr . . . . .	98

mlr_tasks_actg . . . . .	101
mlr_tasks_faithful . . . . .	102
mlr_tasks_gbcs . . . . .	102
mlr_tasks_grace . . . . .	103
mlr_tasks_lung . . . . .	103
mlr_tasks_precip . . . . .	104
mlr_tasks_rats . . . . .	104
mlr_tasks_unemployment . . . . .	105
mlr_tasks_whas . . . . .	105
mlr_task_generators_simdens . . . . .	106
mlr_task_generators_simsurv . . . . .	107
pecs . . . . .	108
PipeOpCrankCompositor . . . . .	110
PipeOpDistrCompositor . . . . .	112
PipeOpPredTransformer . . . . .	114
PipeOpProbegrCompositor . . . . .	116
PipeOpTaskTransformer . . . . .	118
PipeOpTransformer . . . . .	119
plot.LearnerSurv . . . . .	121
PredictionDens . . . . .	122
PredictionSurv . . . . .	123
TaskDens . . . . .	125
TaskSurv . . . . .	126
whas . . . . .	129

<b>Index</b>	<b>130</b>
--------------	------------

---

mlr3proba-package	<i>mlr3proba: Probabilistic Supervised Learning for 'mlr3'</i>
-------------------	--

---

## 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 <raphael.sonabend.15@ucl.ac.uk> ([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>

---

 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).

**ensor** Event indicator. 1 = AIDS defining diagnosis, 0 = Otherwise.

**time\_d** Time to death (days)

**ensor\_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](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

---

gbcs

*German Breast Cancer Study (GBCS) Dataset*

---

**Description**

gbcs dataset from Hosmer et al.

**Usage**

gbcs

**Format**

**id** Identification Code  
**diagdate** Date of diagnosis.  
**recdate** Date of recurrence free survival.  
**deathdate** Date of death.  
**age** Age at diagnosis (years).  
**menopause** Menopausal status. 1 = Yes, 0 = No.  
**hormone** Hormone therapy. 1 = Yes. 0 = No.  
**size** Tumor size (mm).  
**grade** Tumor grade (1-3).  
**nodes** Number of nodes.  
**prog\_recp** Number of progesterone receptors.  
**estrg\_recp** Number of estrogen receptors.  
**rectime** Time to recurrence (days).  
**censrec** Recurrence status. 1 = Recurrence. 0 = Censored.  
**survtime** Time to death (days).  
**censdead** Censoring status. 1 = Death. 0 = Censored.

**Source**

[ftp://ftp.wiley.com/public/sci\\_tech\\_med/survival](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	<i>GRACE 1000 Dataset</i>
-------	---------------------------

---

**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](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	<i>Density Learner</i>
-------------	------------------------

---

**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 pack-
  ages 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,
  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.

`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,
  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.

`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.harrellC](#)

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

`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.

### 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,
  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.

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 <a href="#">mlr3pipelines::PipeOpLearner</a> , a PipeOp which will be wrapped in <a href="#">mlr3pipelines::Graph</a> or a Graph itself. Underlying Learner should be <a href="#">LearnerSurv</a> .
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 <a href="#">Graph</a> as a <a href="#">GraphLearner</a> 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:
library("mlr3")
library("mlr3pipelines")

task = tgen("simsurv")$generate(10)
pipe = ppl(
  "crankcompositor",
  learner = lrn("surv.coxph"),
  method = "median"
)
pipe$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 <a href="#">mlr3pipelines::PipeOpLearner</a> , a PipeOp which will be wrapped in <a href="#">mlr3pipelines::Graph</a> or a Graph itself. Underlying Learner should be <a href="#">LearnerSurv</a> .
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 <a href="#">Graph</a> as a <a href="#">GraphLearner</a> otherwise (default) returns as a <a href="#">Graph</a> .
...	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:
library("mlr3")
library("mlr3pipelines")

task = tgen("simSurv")$generate(10)
pipe = ppl(
  "distrcompositor",
  learner = lrn("surv.rpart"),
  estimator = "kaplan",
  form = "ph"
)
pipe$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 <a href="#">mlr3pipelines::PipeOpLearner</a> , a PipeOp which will be wrapped in <a href="#">mlr3pipelines::Graph</a> or a Graph itself. Underlying Learner should be <a href="#">LearnerRegr</a> .
learner_se	[mlr3::Learner] [mlr3pipelines::PipeOp] Optional <a href="#">LearnerRegr</a> 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 <a href="#">Graph</a> as a <a href="#">GraphLearner</a> 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:
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 <a href="#">LearnerSurv</a> s to average.
param_vals	(list()) Parameters, including weights, to pass to <a href="#">PipeOpSurvAvg</a> .
graph_learner	logical(1) If TRUE returns wraps the <a href="#">Graph</a> as a <a href="#">GraphLearner</a> otherwise (default) returns as a <a href="#">Graph</a> .

## 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:
library("mlr3")
library("mlr3pipelines")

task = tgen("simsurv")$generate(10)
pipe = ppl(
  "survaverager",
  learners = lrns(c("surv.kaplan", "surv.coxph")),
  param_vals = list(weights = c(0.1, 0.9)),
  graph_learner = FALSE
)
pipe$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 <a href="#">mlr3pipelines::PipeOpLearner</a> , a PipeOp which will be wrapped in <a href="#">mlr3pipelines::Graph</a> or a Graph itself. Underlying Learner should be <a href="#">LearnerSurv</a> .
iterations	integer(1) Number of bagging iterations. Defaults to 10.
frac	numeric(1) Percentage of rows to keep during subsampling. See <a href="#">PipeOpSubsample</a> for more information. Defaults to 0.7.

avg	logical(1) If TRUE (default) predictions are aggregated with <a href="#">PipeOpSurvAvg</a> , 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 <a href="#">PipeOpSurvAvg</a> .
graph_learner	logical(1) If TRUE returns wraps the <a href="#">Graph</a> as a <a href="#">GraphLearner</a> otherwise (default) returns as a <a href="#">Graph</a> .

### 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\\_probregcompositor](#), [mlr\\_graphs\\_survaverager](#), [mlr\\_graphs\\_survtoregr](#)

### Examples

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

task = tgen("simsurv")$generate(10)
pipe = ppl(
  "survbagging",
  learner = lrn("surv.coxph"),
  iterations = 5,
  graph_learner = FALSE
)
pipe$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	<a href="#">LearnerRegr</a> Regression learner to fit to the transformed <a href="#">TaskRegr</a> . 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 <a href="#">PipeOpDistrCompositor</a> is utilised to transform the deterministic predictions to a survival distribution.
distr_estimator	<a href="#">LearnerSurv</a> 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	<a href="#">LearnerRegr</a> For method 2 if <code>regr_learner</code> is not used to predict the se then a <a href="#">LearnerRegr</a> with <code>se predict_type</code> must be provided.
surv_learner	<a href="#">LearnerSurv</a> For method 3, a <a href="#">LearnerSurv</a> with <code>lp predict_type</code> to estimate linear predictors.

survreg_params	list() Parameters passed to <a href="#">PipeOpTaskSurvRegr</a> , default are survival to regression transformation via ipcw, with weighting determined by Kaplan-Meier and no additional penalty for censoring.
distrcompose_params	list() Parameters passed to <a href="#">PipeOpDistrCompositor</a> , default is accelerated failure time model form.
probreg_params	list() Parameters passed to <a href="#">PipeOpProbregCompositor</a> , default is <a href="#">Normal</a> distribution for composition.
learnercv_params	list() Parameters passed to <a href="#">PipeOpLearnerCV</a> , default is to use insampling.
graph_learner	logical(1) If TRUE returns wraps the <a href="#">Graph</a> as a <a href="#">GraphLearner</a> otherwise (default) returns as a <a href="#">Graph</a> .

## 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) [PipeOpProbregCompositor](#) 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 by 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:
library("mlr3")
library("mlr3pipelines")

task = tgen("simsurv")$generate(10)

# 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 (mr1), one regr learner
pipe = ppl(
  "survtoregr",
  method = 2,
  regr_learner = lrn("regr.featureless", predict_type = "se"),
  survregr_params = list(method = "mr1")
)
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
- Feature Types: logical, integer, numeric, character, factor, ordered
- 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.

**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
- Feature Types: logical, integer, numeric, character, factor, ordered
- 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 BW (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 = 1e-15)
```

*Arguments:*

`eps` (numeric(1))

Very small number to set zero-valued predicted probabilities to in order to prevent errors in `log(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.beggC

*Begg's C-Index Survival Measure*


---

## Description

Calls `survAUC::BeggC()`.

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()`:

```
MeasureSurvBeggC$new()
mlr_measures$get("surv.beggC")
msr("surv.beggC")
```

## Meta Information

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

## Super classes

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

## Methods

### Public methods:

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

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

*Usage:*

```
MeasureSurvBeggC$new()
```

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

*Usage:*

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

*Arguments:*

```
deep Whether to make a deep clone.
```

**References**

Begg CB, Cramer LD, Venkatraman ES, Rosai J (2000). “Comparing tumour staging and grading systems: a case study and a review of the issues, using thymoma as a model.” *Statistics in Medicine*, **19**(15), 1997–2014. doi: [10.1002/10970258\(20000815\)19:15<1997::aid-sim511>3.0.co;2c](https://doi.org/10.1002/10970258(20000815)19:15<1997::aid-sim511>3.0.co;2c).

**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.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other Concordance survival measures: `mlr_measures_surv.gonenC`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.unoC`

Other lp survival measures: `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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.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  $\delta$  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  $\alpha$  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 HC (2000). "Validation, calibration, revision and combination of prognostic survival models." *Statistics in Medicine*, **19**(24), 3401–3415. ISSN 02776715, doi: [10.1002/1097-0258\(20001230\)19:24<3401::AIDSIM554>3.0.CO;22](https://doi.org/10.1002/1097-0258(20001230)19:24<3401::AIDSIM554>3.0.CO;22).

**See Also**

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambles`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other calibration survival measures: `mlr_measures_surv.calib_beta`

Other distr survival measures: `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `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  $\beta$  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 HC (2000). “Validation, calibration, revision and combination of prognostic survival models.” *Statistics in Medicine*, **19**(24), 3401–3415. ISSN 02776715, doi: [10.1002/1097-0258\(20001230\)19:24<3401::AIDSIM554>3.0.CO;22](https://doi.org/10.1002/1097-0258(20001230)19:24<3401::AIDSIM554>3.0.CO;22).

**See Also**

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.chamble`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other calibration survival measures: `mlr_measures_surv.calib_alpha`



Other lp survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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: lp

## 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.beggC](#), [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.calib\\_beta](#), [mlr\\_measures\\_surv.cindex](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.harrellC](#), [mlr\\_measures\\_surv.hung\\_auc](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.logloss](#), [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.mse](#), [mlr\\_measures\\_surv.nagelk\\_r2](#), [mlr\\_measures\\_surv.oquigley\\_r2](#), [mlr\\_measures\\_surv.rmseSE](#), [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.unoC](#), [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.beggC](#), [mlr\\_measures\\_surv.calib\\_beta](#), [mlr\\_measures\\_surv.gonenC](#), [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.

### 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 R, Peto J (1972). "Asymptotically efficient rank invariant test procedures." *Journal of the Royal Statistical Society: Series A (General)*, **135**(2), 185–198. ISSN 0035-9238.
- Harrell FE, Califf RM, Pryor DB, Lee KL, Rosati RA (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 M, Wakounig S, Heinze G (2009). "The estimation of average hazard ratios by weighted Cox regression." *Statistics in Medicine*, **28**(19), 2473–2489. ISSN 0277-6715, doi: [10.1002/sim.3623](https://doi.org/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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intl`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

---

`mlr_measures_surv.gonenC`

*Gonen and Heller's C-Index Survival Measure*

---

**Description**

Calls `survAUC::GHCI()`.

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()`:

```
MeasureSurvGonenC$new()
mlr_measures$get("surv.gonenC")
msr("surv.gonenC")
```

**Meta Information**

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

**Super classes**

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

## Methods

### Public methods:

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

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

*Usage:*

```
MeasureSurvGonenC$new()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

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).

## See Also

Other survival measures: [mlr\\_measures\\_surv.beggC](#), [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.calib\\_beta](#), [mlr\\_measures\\_surv.chambless\\_auc](#), [mlr\\_measures\\_surv.cindex](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.harrellC](#), [mlr\\_measures\\_surv.hung\\_auc](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.logloss](#), [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.mse](#), [mlr\\_measures\\_surv.nagelk\\_r2](#), [mlr\\_measures\\_surv.oquigley\\_r2](#), [mlr\\_measures\\_surv.rmseSE](#), [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.unoC](#), [mlr\\_measures\\_surv.uno\\_auc](#), [mlr\\_measures\\_surv.uno\\_tnr](#), [mlr\\_measures\\_surv.uno\\_tpr](#), [mlr\\_measures\\_surv.xu\\_r2](#)

Other Concordance survival measures: [mlr\\_measures\\_surv.beggC](#), [mlr\\_measures\\_surv.harrellC](#), [mlr\\_measures\\_surv.unoC](#)

Other lp survival measures: [mlr\\_measures\\_surv.beggC](#), [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\\_measures\\_surv.xu\\_r2](#)

---

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.

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.

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

### 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)`

*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.

**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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `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.grafSE`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.schmid`

Other distr survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.schmid`

---

`mlr_measures_surv.grafSE`

*Standard Error of Integrated Graf Score Survival Measure*

---

**Description**

Calculates the standard error of [MeasureSurvGraf](#).

If `integrated == FALSE` then the standard error of the loss,  $L$ , is approximated via,

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

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

If `integrated == TRUE` then correlations between time-points need to be taken into account, therefore

$$se(L) = \sqrt{\frac{\sum_{i=1}^M \sum_{j=1}^M \Sigma_{i,j}}{NT^2}}$$

where  $\Sigma_{i,j}$  is the sample covariance matrix over  $M$  distinct time-points.

**Dictionary**

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

```
MeasureSurvGrafSE$new()
mlr_measures$get("surv.grafSE")
msr("surv.grafSE")
```

**Meta Information**

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

**Super classes**

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

**Methods****Public methods:**

- `MeasureSurvGrafSE$new()`
- `MeasureSurvGrafSE$clone()`

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

*Usage:*

```
MeasureSurvGrafSE$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:*

```
MeasureSurvGrafSE$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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`,

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.unoC, mlr\_measures\_surv.uno\_auc, mlr\_measures\_surv.uno\_tpr, mlr\_measures\_surv.xu\_r2

Other Probabilistic survival measures: mlr\_measures\_surv.graf, mlr\_measures\_surv.intloglossSE, mlr\_measures\_surv.intlogloss, mlr\_measures\_surv.loglossSE, mlr\_measures\_surv.logloss, mlr\_measures\_surv.schmid

Other distr survival measures: mlr\_measures\_surv.calib\_alpha, mlr\_measures\_surv.graf, mlr\_measures\_surv.intloglossSE, mlr\_measures\_surv.intlogloss, mlr\_measures\_surv.loglossSE, mlr\_measures\_surv.logloss, mlr\_measures\_surv.schmid

---

mlr\_measures\_surv.harrellC

*Harrell's C-Index Survival Measure*

---

## Description

Calculates Harrell's C, equivalent to the Fortran implementation in **Hmisc**.

## Dictionary

This [Measure](#) can be instantiated via the [dictionary mlr\\_measures](#) or with the associated sugar function [msr\(\)](#):

```
MeasureSurvHarrellC$new()
mlr_measures$get("surv.harrellC")
msr("surv.harrellC")
```

## Meta Information

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

## Super classes

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

## Methods

### Public methods:

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

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

*Usage:*

```
MeasureSurvHarrellC$new()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

Harrell FE, Califf RM, Pryor DB, Lee KL, Rosati RA (1982). “Evaluating the yield of medical tests.” *Jama*, **247**(18), 2543–2546.

## See Also

Other survival measures: [mlr\\_measures\\_surv.beggC](#), [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.calib\\_b](#), [mlr\\_measures\\_surv.chambless\\_auc](#), [mlr\\_measures\\_surv.cindex](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.hung\\_auc](#), [mlr\\_measures\\_surv.intlog](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.logloss](#), [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.mse](#), [mlr\\_measures\\_surv.nagelk\\_r2](#), [mlr\\_measures\\_surv.oquigley\\_r2](#), [mlr\\_measures\\_surv.rmseSE](#), [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.unoC](#), [mlr\\_measures\\_surv.uno\\_auc](#), [mlr\\_measures\\_surv.uno\\_tr](#), [mlr\\_measures\\_surv.uno\\_tpr](#), [mlr\\_measures\\_surv.xu\\_r2](#)

Other Concordance survival measures: [mlr\\_measures\\_surv.beggC](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.unoC](#)

Other crank survival measures: [mlr\\_measures\\_surv.unoC](#)

---

```
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 `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:*

```
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. <http://www.jstor.org/stable/27805213>.

**See Also**

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `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.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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 logarithmic (log), 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.

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.

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

**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**

```
eps (numeric(1))
  Very small number used to prevent log(0) error.
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 = 1e-15,
  method = 2,
  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.
```

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

**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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_a`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other Probabilistic survival measures: `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.schmid`

Other distr survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.schmid`



---

 mlr\_measures\_surv.intloglossSE

*Standard Error of Integrated Log loss Survival Measure*


---

## Description

Calculates the standard error of [MeasureSurvIntLogloss](#).

If `integrated == FALSE` then the standard error of the loss,  $L$ , is approximated via,

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

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

If `integrated == TRUE` then correlations between time-points need to be taken into account, therefore

$$se(L) = \sqrt{\frac{\sum_{i=1}^M \sum_{j=1}^M \Sigma_{i,j}}{NT^2}}$$

where  $\Sigma_{i,j}$  is the sample covariance matrix over  $M$  distinct time-points.

## Dictionary

This [Measure](#) can be instantiated via the [dictionary mlr\\_measures](#) or with the associated sugar function [msr\(\)](#):

```
MeasureSurvIntLoglossSE$new()
mlr_measures$get("surv.intloglossSE")
msr("surv.intloglossSE")
```

## Meta Information

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

## Super classes

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

## Active bindings

```
eps (numeric(1))
  Very small number used to prevent log(0) error.
```

## Methods

### Public methods:

- `MeasureSurvIntLoglossSE$new()`
- `MeasureSurvIntLoglossSE$clone()`

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

*Usage:*

```
MeasureSurvIntLoglossSE$new(integrated = TRUE, times, eps = 1e-15)
```

*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)` calculation.

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

*Usage:*

```
MeasureSurvIntLoglossSE$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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_a`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

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

Other distr survival measures: `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `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.

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\(\)](#):

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

## Methods

### Public methods:

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

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

*Usage:*

```
MeasureSurvLogloss$new(id = "surv.logloss", eps = 1e-15, se = FALSE)
```

*Arguments:*

`id` (character(1))

Identifier for the new instance.

`eps` (numeric(1))

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

`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:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

Other survival measures: [mlr\\_measures\\_surv.beggC](#), [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.calib\\_b](#), [mlr\\_measures\\_surv.chambless\\_auc](#), [mlr\\_measures\\_surv.cindex](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.harrellC](#), [mlr\\_measures\\_surv.hung\\_auc](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.mse](#), [mlr\\_measures\\_surv.nagelk\\_r2](#), [mlr\\_measures\\_surv.oquigley\\_r2](#), [mlr\\_measures\\_surv.rmseSE](#), [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.unoC](#), [mlr\\_measures\\_surv.uno\\_auc](#), [mlr\\_measures\\_surv.uno\\_tpr](#), [mlr\\_measures\\_surv.xu\\_r2](#)

Other Probabilistic survival measures: [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.schmid](#)

Other distr survival measures: [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.schmid](#)

---

mlr\_measures\_surv.loglossSE  
*Standard Error of Log loss Survival Measure*

---

### Description

Calculates the standard error of [MeasureSurvLogloss](#).

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.

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\(\)](#):

```
MeasureSurvLoglossSE$new()
mlr_measures$get("surv.loglossSE")
msr("surv.loglossSE")
```

### Meta Information

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

### Super classes

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

### Methods

#### Public methods:

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

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

*Usage:*

```
MeasureSurvLoglossSE$new(eps = 1e-15)
```

*Arguments:*

eps (numeric(1))

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

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

## See Also

Other survival measures: [mlr\\_measures\\_surv.beggC](#), [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.calib\\_b](#), [mlr\\_measures\\_surv.chambless\\_auc](#), [mlr\\_measures\\_surv.cindex](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.harrellC](#), [mlr\\_measures\\_surv.hung\\_a](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.logloss](#), [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.mse](#), [mlr\\_measures\\_surv.nagelk\\_r2](#), [mlr\\_measures\\_surv.oquigley\\_r2](#), [mlr\\_measures\\_surv.rmseSE](#), [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.unoC](#), [mlr\\_measures\\_surv.uno\\_auc](#), [mlr\\_measures\\_surv.uno\\_tr](#), [mlr\\_measures\\_surv.uno\\_tpr](#), [mlr\\_measures\\_surv.xu\\_r2](#)

Other Probabilistic survival measures: [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.logloss](#), [mlr\\_measures\\_surv.schmid](#)

Other distr survival measures: [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.logloss](#), [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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other response survival measures: `mlr_measures_surv.maeSE`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.rmseSE`, `mlr_measures_surv.rmse`

---

```
mlr_measures_surv.maeSE
```

*Standard Error of Mean Absolute Error Survival Measure*

---

### Description

Calculates the standard error of [MeasureSurvMAE](#).

The standard error of the MAE,  $L$ , is approximated via

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

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\(\)](#):

```
MeasureSurvMAESE$new()
mlr_measures$get("surv.maeSE")
msr("surv.maeSE")
```

### Meta Information

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

### Super classes

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

### Methods

#### Public methods:

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

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

*Usage:*

```
MeasureSurvMAESE$new()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.



**See Also**

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_a`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other response survival measures: `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.rmseSE`, `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.beggC](#), [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.calib\\_b](#), [mlr\\_measures\\_surv.chambless\\_auc](#), [mlr\\_measures\\_surv.cindex](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.harrellC](#), [mlr\\_measures\\_surv.hung\\_a](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.logloss](#), [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.nagelk\\_r2](#), [mlr\\_measures\\_surv.oquigley\\_r2](#), [mlr\\_measures\\_surv.rmseSE](#), [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.unoC](#), [mlr\\_measures\\_surv.uno\\_auc](#), [mlr\\_measures\\_surv.uno\\_tr](#), [mlr\\_measures\\_surv.uno\\_tpr](#), [mlr\\_measures\\_surv.xu\\_r2](#)

Other response survival measures: [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.rmseSE](#), [mlr\\_measures\\_surv.rmse](#)

---

mlr\_measures\_surv.mseSE

*Standard Error of Mean Squared Error Survival Measure*


---

### Description

Calculates the standard error of [MeasureSurvMSE](#).

The standard error of the MSE,  $L$ , is approximated via

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

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\(\)](#):

```
MeasureSurvMSESE$new()
mlr_measures$get("surv.mseSE")
msr("surv.mseSE")
```

### Meta Information

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

### Super classes

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

### Methods

#### Public methods:

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

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

*Usage:*

```
MeasureSurvMSESE$new()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

**See Also**

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_a`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other response survival measures: `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mse`, `mlr_measures_surv.rmseSE`, `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 NJ, 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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `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.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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: 1p

## 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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `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.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_a`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tr`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other response survival measures: `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.rmseSE`

---

```
mlr_measures_surv.rmseSE
```

*Standard Error of Root Mean Squared Error Survival Measure*

---

**Description**

Calculates the standard error of [MeasureSurvRMSE](#).

The standard error of the RMSE, L, is approximated via first order approximation by

$$sd(RMSE) = Var(\sqrt{MSE})$$

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()`:

```
MeasureSurvRMSESE$new()
mlr_measures$get("surv.rmseSE")
msr("surv.rmseSE")
```

**Meta Information**

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

**Super classes**

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

## Methods

### Public methods:

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

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

*Usage:*

```
MeasureSurvRMSESE$new()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

Other survival measures: [mlr\\_measures\\_surv.beggC](#), [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.calib\\_b](#), [mlr\\_measures\\_surv.chambless\\_auc](#), [mlr\\_measures\\_surv.cindex](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.harrellC](#), [mlr\\_measures\\_surv.hung\\_auc](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.logloss](#), [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [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.unoC](#), [mlr\\_measures\\_surv.uno\\_auc](#), [mlr\\_measures\\_surv.uno\\_tpr](#), [mlr\\_measures\\_surv.uno\\_tpr](#), [mlr\\_measures\\_surv.xu\\_r2](#)

Other response survival measures: [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.mse](#), [mlr\\_measures\\_surv.rmse](#)

---

```
mlr_measures_surv.schmid
```

*Integrated Schmid Score Survival Measure*

---

## Description

Calculates the Integrated Schmid Score, 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.

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.

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

## 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)

*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.

**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 M, Henderson R (2000). “Predictive Accuracy and Explained Variation in Cox Regression.” *Biometrics*, **56**, 249–255. ISSN 02776715, doi: [10.1002/sim.1486](https://doi.org/10.1002/sim.1486).

Schmid M, Hielscher T, Augustin T, Gefeller O (2011). “A Robust Alternative to the Schemper-Henderson Estimator of Prediction Error.” *Biometrics*, **67**(2), 524–535. ISSN 0006341X, 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.beggC](#), [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.calib\\_b](#), [mlr\\_measures\\_surv.chambless\\_auc](#), [mlr\\_measures\\_surv.cindex](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.harrellC](#), [mlr\\_measures\\_surv.hung\\_auc](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.logloss](#), [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.mse](#), [mlr\\_measures\\_surv.nagelk\\_r2](#), [mlr\\_measures\\_surv.oquigley\\_r2](#), [mlr\\_measures\\_surv.rmseSE](#), [mlr\\_measures\\_surv.rmse](#), [mlr\\_measures\\_surv.song\\_auc](#), [mlr\\_measures\\_surv.song\\_tpr](#), [mlr\\_measures\\_surv.song\\_tpr](#), [mlr\\_measures\\_surv.unoC](#), [mlr\\_measures\\_surv.uno\\_auc](#), [mlr\\_measures\\_surv.uno\\_tpr](#), [mlr\\_measures\\_surv.uno\\_tpr](#), [mlr\\_measures\\_surv.xu\\_r2](#)

Other Probabilistic survival measures: [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.logloss](#)

Other distr survival measures: [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [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: 1p

## 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

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

## See Also

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.unoC`, `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.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chamb1`, `mlr_measures_surv.gonenC`, `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_tnr`

---

`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::MeasureSurvAUC
-> 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 X, Zhou X (2008). “A semiparametric approach for the covariate specific ROC curve with survival outcome.” *Statistica Sinica*, **18**(3), 947–65. <http://www.jstor.org/stable/24308524>.

## See Also

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tpr`, `mlr_measures_surv.unoC`, `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.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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.

`t` 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::MeasureSurvAUC
-> 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 X, Zhou X (2008). “A semiparametric approach for the covariate specific ROC curve with survival outcome.” *Statistica Sinica*, **18**(3), 947–65. <http://www.jstor.org/stable/24308524>.

## See Also

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `mlr_measures_surv.rmse`, `mlr_measures_surv.schmid`, `mlr_measures_surv.song_auc`, `mlr_measures_surv.song_tnr`, `mlr_measures_surv.unoC`, `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.beggC](#), [mlr\\_measures\\_surv.calib\\_beta](#), [mlr\\_measures\\_surv.chamb1](#), [mlr\\_measures\\_surv.gonenC](#), [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\\_r](#)

---

`mlr_measures_surv.unoC`

*Uno's C-Index Survival Measure*

---

### Description

Calls [survAUC::UnoC\(\)](#).

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\(\)](#):

```
MeasureSurvUnoC$new()
mlr_measures$get("surv.unoC")
msr("surv.unoC")
```

### Meta Information

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

### Super classes

[mlr3::Measure](#) -> [mlr3proba::MeasureSurv](#) -> `MeasureSurvUnoC`

**Methods****Public methods:**

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

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

*Usage:*

```
MeasureSurvUnoC$new()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

**References**

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.beggC](#), [mlr\\_measures\\_surv.calib\\_alpha](#), [mlr\\_measures\\_surv.calib\\_b](#), [mlr\\_measures\\_surv.chambless\\_auc](#), [mlr\\_measures\\_surv.cindex](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.grafSE](#), [mlr\\_measures\\_surv.graf](#), [mlr\\_measures\\_surv.harrellC](#), [mlr\\_measures\\_surv.hung\\_a](#), [mlr\\_measures\\_surv.intloglossSE](#), [mlr\\_measures\\_surv.intlogloss](#), [mlr\\_measures\\_surv.loglossSE](#), [mlr\\_measures\\_surv.logloss](#), [mlr\\_measures\\_surv.maeSE](#), [mlr\\_measures\\_surv.mae](#), [mlr\\_measures\\_surv.mseSE](#), [mlr\\_measures\\_surv.mse](#), [mlr\\_measures\\_surv.nagelk\\_r2](#), [mlr\\_measures\\_surv.oquigley\\_r2](#), [mlr\\_measures\\_surv.rmseSE](#), [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 Concordance survival measures: [mlr\\_measures\\_surv.beggC](#), [mlr\\_measures\\_surv.gonenC](#), [mlr\\_measures\\_surv.harrellC](#)

Other crank survival measures: [mlr\\_measures\\_surv.harrellC](#)

---

`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::MeasureSurvAU
-> 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 `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:*

```
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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `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.song_tpr`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tnr`, `mlr_measures_surv.uno_tpr`

Other lp survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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.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::MeasureSurvAU
-> 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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `mlr_measures_surv.uno_auc`, `mlr_measures_surv.uno_tpr`, `mlr_measures_surv.xu_r2`

Other lp survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `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.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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: lp

**Super classes**

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `MeasureSurvXuR2`

**Methods****Public methods:**

- `MeasureSurvXuR2$new()`
- `MeasureSurvXuR2$clone()`

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

*Usage:*

`MeasureSurvXuR2$new()`

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

*Usage:*

`MeasureSurvXuR2$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

**References**

Xu R, O'Quigley J (1999). "A R2 type measure of dependence for proportional hazards models." *Journal of Nonparametric Statistics*, **12**(1), 83–107. doi: [10.1080/10485259908832799](https://doi.org/10.1080/10485259908832799).

**See Also**

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.cindex`, `mlr_measures_surv.gonenC`, `mlr_measures_surv.grafSE`, `mlr_measures_surv.graf`, `mlr_measures_surv.harrellC`, `mlr_measures_surv.hung_auc`, `mlr_measures_surv.intloglossSE`, `mlr_measures_surv.intlogloss`, `mlr_measures_surv.loglossSE`, `mlr_measures_surv.logloss`, `mlr_measures_surv.maeSE`, `mlr_measures_surv.mae`, `mlr_measures_surv.mseSE`, `mlr_measures_surv.mse`, `mlr_measures_surv.nagelk_r2`, `mlr_measures_surv.oquigley_r2`, `mlr_measures_surv.rmseSE`, `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.unoC`, `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.beggC`, `mlr_measures_surv.calib_beta`, `mlr_measures_surv.chambless_auc`, `mlr_measures_surv.gonenC`, `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\_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:
library(mlr3)
library(mlr3pipelines)
set.seed(1)

task = tgen("simsurv")$generate(10)
p1 = lrn("surv.coxph")$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\\_survreg](#), [mlr\\_pipeops\\_trafotask\\_regrsurv](#), [mlr\\_pipeops\\_trafotask\\_survreg](#)

Other Transformation PipeOps: [mlr\\_pipeops\\_trafopred\\_survreg](#), [mlr\\_pipeops\\_trafotask\\_regrsurv](#), [mlr\\_pipeops\\_trafotask\\_survreg](#)

**Examples**

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

# simple example
pred = PredictionRegr$new(row_ids = 1:10, truth = 1:10, response = 1:10)
po = po("trafopred_regrsurv")

# assume no censoring
new_pred = po$predict(list(pred = pred, task = NULL))[[1]]
po$train(list(NULL, NULL))
print(new_pred)

# add censoring
task_surv = tsk("rats")
task_regr = po("trafotask_survreg", 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_survreg
      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_survreg")
```

*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\\_survreg](#)

Other Transformation PipeOps: [mlr\\_pipeops\\_trafopred\\_regrsurv](#), [mlr\\_pipeops\\_trafotask\\_regrsurv](#), [mlr\\_pipeops\\_trafotask\\_survreg](#)

**Examples**

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

# simple example
pred = PredictionSurv$new(row_ids = 1:10, truth = Surv(1:10, rbinom(10, 1, 0.5)),
  response = 1:10)
po = po("trafopred_survreg")
new_pred = po$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\\_survregr](#), [mlr\\_pipeops\\_trafotask\\_survregr](#)

Other Transformation PipeOps: [mlr\\_pipeops\\_trafopred\\_regrsurv](#), [mlr\\_pipeops\\_trafopred\\_survregr](#), [mlr\\_pipeops\\_trafotask\\_survregr](#)

**Examples**

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

task = tsk("boston_housing")
po = po("trafotask_regrsurv")

# assume no censoring
new_task = po$train(list(task_regr = task, task_surv = NULL))[[1]]
print(new_task)

# add censoring
task_surv = tsk("rats")
task_regr = po("trafotask_survregr", method = "omit")$train(list(task_surv, NULL))[[1]]
print(task_regr)
new_task = po$train(list(task_regr = task_regr, task_surv = task_surv))[[1]]
new_task$truth()
task_surv$truth()

## End(Not run)
```

---

```
mlr_pipeops_trafotask_survregr
      PipeOpTaskSurvRegr
```

---

**Description**

Transform [TaskSurv](#) to [TaskRegr](#).

**Input and Output Channels**

Input and output channels are inherited from [PipeOpTaskTransformer](#).

The output is the input [TaskSurv](#) transformed to a [TaskRegr](#).

**State**

The \$state is a named list with the \$state elements

- instatus: Censoring status from input training task.
- outstatus : Censoring status from input prediction task.

## Parameters

The parameters are

- `method::character(1)`  
Method to use for dealing with censoring. Options are "ipcw" (Vock et al., 2016): censoring 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  $\delta$  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 `mlr3learners.proba::akritas` estimator, 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 J, James I (1979). "Linear Regression with Censored Data." *Biometrika*, **66**(3), 429–436. ISSN 00063444, doi: [10.2307/2335161](https://doi.org/10.2307/2335161), <http://www.jstor.org/stable/2335161>.
- Klein JP, Moeschberger ML (2003). *Survival analysis: techniques for censored and truncated data*, 2 edition. Springer Science & Business Media. ISBN 0387216456.
- Vock DM, Wolfson J, Bandyopadhyay S, Adomavicius G, Johnson PE, Vazquez-Benitez G, O'Connor PJ (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. ISSN 1532-0464, 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:
library(mlr3)
library(mlr3pipelines)

# these methods are generally only successful if censoring is not too high
# create survival task by undersampling
set.seed(1)
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]]

# 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
po = po("trafotask_survreg", 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_survreg", 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_survreg", 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()` from package **simsurv**. 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**

```
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 = 1e-15, ...)
```

```
## S3 method for class 'list'
pecs(
  x,
  measure = c("graf", "logloss"),
  times,
  n,
  eps = 1e-15,
  task = NULL,
  row_ids = NULL,
  newdata,
  ...
)
```

```
## S3 method for class 'PredictionSurv'
pecs(x, measure = c("graf", "logloss"), times, n, eps = 1e-15, ...)
```

**Arguments**

x	( <a href="#">PredictionSurv</a> or list of <a href="#">LearnerSurv</a> s)
measure	(character(1)) Either "graf" for <a href="#">MeasureSurvGraf</a> , or "logloss" for <a href="#">MeasureSurvIntLogloss</a>
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 pass to <a href="#">MeasureSurvIntLogloss</a> to prevent errors resulting from a log(0) calculation.
...	Additional arguments.
task	( <a href="#">TaskSurv</a> )
row_ids	(integer()) Passed to <code>Learner\$predict</code> .
newdata	( <code>data.frame()</code> ) If not missing <code>Learner\$predict_newdata</code> is called instead of <code>Learner\$predict</code> .

### Details

If times and n are missing then measure is evaluated over all observed time-points from the [PredictionSurv](#) or [TaskSurv](#) object. If a range is provided for times without n, then all time-points between the range are returned.

### Examples

```
## Not run:
library(mlr3)
task = tsk("rats")

# Prediction Error Curves for prediction object
learn = lrn("surv.coxph")
p = learn$train(task)$predict(task)
pecs(p)
pecs(p, measure = "logloss", times = c(20, 40, 60, 80)) +
  ggplot2::geom_point() +
  ggplot2::ggtitle("Logloss Prediction Error Curve for Cox PH")

# Access underlying data
x = pecs(p)
x$data

# Prediction Error Curves for fitted learners
learns = lrns(c("surv.kaplan", "surv.coxph"))
lapply(learns, function(x) x$train(task))
pecs(learns, task = task, measure = "logloss", times = c(20, 90), n = 10)

## End(Not run)
```

---

 PipeOpCrankCompositor *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 `distr6::median.Distribution`, `distr6::mode`, or `distr6::mean.Distribution` respectively.

**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: [PipeOpDistrCompositor](#)

**Examples**

```
## Not run:
library(mlr3)
library(mlr3pipelines)
set.seed(1)
task = tgen("simsurv")$generate(20)

learn = lrn("surv.coxph")$train(task)$predict(task)
```

```
poc = po("crankcompose", param_vals = list(method = "mean"))
poc$predict(list(learn))[[1]]

## End(Not run)
```

---

PipeOpDistrCompositor *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

```
m1r3pipelines::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: [PipeOpCrankCompositor](#)

### Examples

```
## Not run:
library(mlr3)
library(mlr3pipelines)
set.seed(1)
task = tgen("simSurv")$generate(20)

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)
```

---

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\\_regrs](#), [mlr\\_pipeops\\_trafopred\\_survreg](#), [mlr\\_pipeops\\_trafotask\\_regrsurv](#), [mlr\\_pipeops\\_trafotask\\_survreg](#)

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

---

PipeOpProbregrCompositor

*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_probreg")
po("compose_probreg")
```

**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 -> PipeOpProbregCompositor
```

**Methods****Public methods:**

- `PipeOpProbregCompositor$new()`
- `PipeOpProbregCompositor$clone()`

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

*Usage:*

```
PipeOpProbregCompositor$new(
  id = "compose_probreg",
  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:*

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

*Arguments:*

`deep` Whether to make a deep clone.

**Examples**

```
## Not run:
library(mlr3)
library(mlr3pipelines)
set.seed(1)
task = tsk("boston_housing")

# Option 1: Use a learner that can predict se
learn = lrn("regr.featureless", predict_type = "se")
pred = learn$train(task)$predict(task)
poc = po("compose_probreg")
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_probreg")
poc$predict(list(pred_response, pred_se))[[1]]

## End(Not run)
```

---

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**

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

**Examples**

```
## Not run:
library(mlr3)
task = tgen("simsurv")$generate(20)

# 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

*Prediction Object for Density*


---

### Description

This object stores the predictions returned by a learner of class [LearnerDens](#).

The `task_type` is set to "dens".

### Super class

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

### Active bindings

`pdf` (numeric())  
 Access the stored predicted probability density function.

`cdf` (numeric())  
 Access the stored predicted cumulative distribution function.

`missing` (integer())  
 Returns `row_ids` for which the predictions are missing or incomplete.

### 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,
  truth = task$truth(),
  pdf = NULL,
  cdf = NULL
)

```

*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` (numeric())  
True (observed) response.

`pdf` (numeric())  
Numeric vector of estimated probability density function, evaluated at 'target' column of test set. One element for each observation in the test set.

`cdf` (numeric())  
Numeric vector of estimated cumulative distribution function, evaluated at 'target' column of test set. One element for each observation in the test set.

**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](#))  
Access the stored predicted survival distribution.

lp (numeric())  
Access the stored predicted linear predictor.

response (numeric())  
Access the stored predicted survival time.

missing (integer())  
Returns row\_ids for which the predictions are missing or incomplete.

**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
)
```

*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 (numeric())  
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 ([VectorDistribution](#))  
[VectorDistribution](#) from [distr6](#). Each individual distribution in the vector represents the random variable 'survival time' for an individual observation.

`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  $\beta$  is a vector of estimated coefficients.

`response` (`numeric()`)  
 Numeric vector of predicted survival times. One element for each observation in the test set.

**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 = tgen("simsurv")$generate(20)
learner = mlr_learners$get("surv.rpart")
p = learner$train(task)$predict(task)
head(as.data.table(p))
```

---

TaskDens

*Density Task*

---

## Description

This task specializes [Task](#) for density estimation problems. The target column is assumed to be numeric. The `task_type` is set to "density".

Predefined tasks are stored in the [dictionary mlr\\_tasks](#).

## Super class

[mlr3::Task](#) -> TaskDens

## Methods

### Public methods:

- [TaskDens\\$new\(\)](#)
- [TaskDens\\$truth\(\)](#)
- [TaskDens\\$clone\(\)](#)

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

*Usage:*

```
TaskDens$new(id, backend, target)
```

*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](#).

```
target (character(1))
```

Name of the target column.

**Method** `truth()`: Returns the target column for specified `row_ids`, this is unsupervised so should not be thought of as a 'true' prediction. Defaults to all rows with role "use".

*Usage:*

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

*Arguments:*

```
rows integer()
```

Row indices.

*Returns:* `numeric()`.

**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$new("precip", backend = data.frame(target = precip), target = "target")
task$task_type
task$truth()
```

---

TaskSurv

*Survival Task*

---

**Description**

This task specializes [mlr3::Task](#) and [mlr3::TaskSupervised](#) for possibly-censored survival problems. The target is comprised of survival times and an event indicator. Predefined tasks are stored in [mlr3::mlr\\_tasks](#).

The `task_type` is set to "surv".

**Super classes**

`mlr3::Task` -> `mlr3::TaskSupervised` -> `TaskSurv`

**Active bindings**

`censtype` `character(1)`

Returns the type of censoring, one of "right", "left", "counting", "interval", "interval2" or "mstate".

**Methods****Public methods:**

- `TaskSurv$new()`
- `TaskSurv$truth()`
- `TaskSurv$formula()`
- `TaskSurv$clone()`

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

*Usage:*

```
TaskSurv$new(
  id,
  backend,
  time,
  event,
  time2,
  type = c("right", "left", "counting", "interval", "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** `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 = mlr3misc::load_dataset("lung", package = "survival")
lung$status = (lung$status == 2L)
b = as_data_backend(lung)
task = TaskSurv$new("lung",
  backend = b, time = "time",
  event = "status")

task$target_names
task$feature_names
task$formula()
task$truth()
```



---

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](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, 41
  - mlr\_measures\_surv.hung\_auc, 52
  - mlr\_measures\_surv.song\_auc, 77
  - mlr\_measures\_surv.song\_tnr, 79
  - mlr\_measures\_surv.song\_tpr, 81
  - mlr\_measures\_surv.uno\_auc, 84
  - mlr\_measures\_surv.uno\_tnr, 86
  - mlr\_measures\_surv.uno\_tpr, 88
- \* **Concordance survival measures**
  - mlr\_measures\_surv.beggC, 36
  - mlr\_measures\_surv.gonenC, 45
  - mlr\_measures\_surv.harrellC, 51
  - mlr\_measures\_surv.unoC, 83
- \* **Density estimation measures**
  - mlr\_measures\_dens.logloss, 35
- \* **Ensembles**
  - mlr\_pipeops\_survavg, 92
- \* **Learner**
  - LearnerDens, 7
  - LearnerSurv, 9
- \* **Measure**
  - MeasureDens, 11
  - MeasureSurv, 13
- \* **PipeOps**
  - mlr\_pipeops\_survavg, 92
  - mlr\_pipeops\_trafopred\_regrsurv, 93
  - mlr\_pipeops\_trafopred\_survregr, 95
  - mlr\_pipeops\_trafotask\_regrsurv, 97
  - mlr\_pipeops\_trafotask\_survregr, 98
  - PipeOpPredTransformer, 114
  - PipeOpTaskTransformer, 118
  - PipeOpTransformer, 119
- \* **Prediction**
  - PredictionDens, 122
  - PredictionSurv, 123
- \* **Probabilistic survival measures**
  - mlr\_measures\_surv.graf, 47
  - mlr\_measures\_surv.grafSE, 49
  - mlr\_measures\_surv.intlogloss, 54
  - mlr\_measures\_surv.intloglossSE, 57
  - mlr\_measures\_surv.logloss, 59
  - mlr\_measures\_surv.loglossSE, 61
  - mlr\_measures\_surv.schmid, 74
- \* **R2 survival measures**
  - mlr\_measures\_surv.nagelk\_r2, 68
  - mlr\_measures\_surv.oquigley\_r2, 70
  - mlr\_measures\_surv.xu\_r2, 90
- \* **Task**
  - TaskDens, 125
  - TaskSurv, 126
- \* **Transformation PipeOps**
  - mlr\_pipeops\_trafopred\_regrsurv, 93
  - mlr\_pipeops\_trafopred\_survregr, 95
  - mlr\_pipeops\_trafotask\_regrsurv, 97
  - mlr\_pipeops\_trafotask\_survregr, 98
- \* **Transformers**
  - PipeOpPredTransformer, 114
  - PipeOpTaskTransformer, 118
  - PipeOpTransformer, 119
- \* **calibration survival measures**
  - mlr\_measures\_surv.calib\_alpha, 37
  - mlr\_measures\_surv.calib\_beta, 39
- \* **crank survival measures**
  - mlr\_measures\_surv.harrellC, 51
  - mlr\_measures\_surv.unoC, 83
- \* **datasets**
  - actg, 5
  - gbcs, 6
  - grace, 7
  - whas, 129
- \* **density estimators**
  - mlr\_learners\_dens.hist, 28
  - mlr\_learners\_dens.kde, 29
- \* **density measures**
  - mlr\_measures\_dens.logloss, 35
- \* **distr survival measures**

- mlr\_measures\_surv.calib\_alpha, 37
- mlr\_measures\_surv.graf, 47
- mlr\_measures\_surv.grafSE, 49
- mlr\_measures\_surv.intlogloss, 54
- mlr\_measures\_surv.intloglossSE, 57
- mlr\_measures\_surv.logloss, 59
- mlr\_measures\_surv.loglossSE, 61
- mlr\_measures\_surv.schmid, 74
- \* **lp survival measures**
  - mlr\_measures\_surv.beggC, 36
  - mlr\_measures\_surv.calib\_beta, 39
  - mlr\_measures\_surv.chambless\_auc, 41
  - mlr\_measures\_surv.gonenC, 45
  - mlr\_measures\_surv.hung\_auc, 52
  - mlr\_measures\_surv.nagelk\_r2, 68
  - mlr\_measures\_surv.oquigley\_r2, 70
  - mlr\_measures\_surv.song\_auc, 77
  - mlr\_measures\_surv.song\_tnr, 79
  - mlr\_measures\_surv.song\_tpr, 81
  - mlr\_measures\_surv.uno\_auc, 84
  - mlr\_measures\_surv.uno\_tnr, 86
  - mlr\_measures\_surv.uno\_tpr, 88
  - mlr\_measures\_surv.xu\_r2, 90
- \* **pipelines**
  - mlr\_graphs\_crankcompositor, 17
  - mlr\_graphs\_distrcompositor, 19
  - mlr\_graphs\_probregrcompositor, 20
  - mlr\_graphs\_survaverager, 22
  - mlr\_graphs\_survbagging, 23
  - mlr\_graphs\_survtoregr, 25
- \* **response survival measures**
  - mlr\_measures\_surv.mae, 62
  - mlr\_measures\_surv.maeSE, 64
  - mlr\_measures\_surv.mse, 65
  - mlr\_measures\_surv.mseSE, 67
  - mlr\_measures\_surv.rmse, 71
  - mlr\_measures\_surv.rmseSE, 73
- \* **survival compositors**
  - PipeOpCrankCompositor, 110
  - PipeOpDistrCompositor, 112
- \* **survival learners**
  - mlr\_learners\_surv.coxph, 31
  - mlr\_learners\_surv.kaplan, 32
  - mlr\_learners\_surv.rpart, 33
- \* **survival measures**
  - mlr\_measures\_surv.beggC, 36
  - mlr\_measures\_surv.calib\_alpha, 37
  - mlr\_measures\_surv.calib\_beta, 39
  - mlr\_measures\_surv.chambless\_auc, 41
  - mlr\_measures\_surv.cindex, 43
  - mlr\_measures\_surv.gonenC, 45
  - mlr\_measures\_surv.graf, 47
  - mlr\_measures\_surv.grafSE, 49
  - mlr\_measures\_surv.harrellC, 51
  - mlr\_measures\_surv.hung\_auc, 52
  - mlr\_measures\_surv.intlogloss, 54
  - mlr\_measures\_surv.intloglossSE, 57
  - mlr\_measures\_surv.logloss, 59
  - mlr\_measures\_surv.loglossSE, 61
  - mlr\_measures\_surv.mae, 62
  - mlr\_measures\_surv.maeSE, 64
  - mlr\_measures\_surv.mse, 65
  - mlr\_measures\_surv.mseSE, 67
  - mlr\_measures\_surv.nagelk\_r2, 68
  - mlr\_measures\_surv.oquigley\_r2, 70
  - mlr\_measures\_surv.rmse, 71
  - mlr\_measures\_surv.rmseSE, 73
  - mlr\_measures\_surv.schmid, 74
  - mlr\_measures\_surv.song\_auc, 77
  - mlr\_measures\_surv.song\_tnr, 79
  - mlr\_measures\_surv.song\_tpr, 81
  - mlr\_measures\_surv.uno\_auc, 84
  - mlr\_measures\_surv.uno\_tnr, 86
  - mlr\_measures\_surv.uno\_tpr, 88
  - mlr\_measures\_surv.unoC, 83
  - mlr\_measures\_surv.xu\_r2, 90
- actg, 5, 101
- bujar::bujar, 99
- crankcompositor
  - (mlr\_graphs\_crankcompositor), 17
- data.table::data.table, 115, 119, 120
- DataBackend, 126, 127
- DataBackendDataTable, 126, 127
- dens.logloss, 12
- Dictionary, 105, 106, 108
- dictionary, 11, 13, 28, 29, 31–33, 36, 38, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 62, 64, 65, 67, 68, 70, 72, 73, 75, 77, 79, 81, 83, 85, 87, 89, 90, 106, 107, 110, 112, 116, 125

- distr6::Distribution, [28](#), [29](#)
- distr6::distrSimulate(), [106](#)
- distr6::mean.Distribution, [111](#)
- distr6::median.Distribution, [111](#)
- distr6::MixtureDistribution, [92](#)
- distr6::mode, [111](#)
- distr6::VectorDistribution, [92](#)
- distrcompositor
  - (mlr\_graphs\_distrcompositor), [19](#)
- faithful, [102](#)
- gbcs, [6](#), [102](#)
- grace, [7](#), [103](#)
- Graph, [17–24](#), [26](#)
- graphics::hist(), [28](#)
- GraphLearner, [18](#), [20–22](#), [24](#), [26](#)
- Learner, [7–10](#), [12](#), [14](#), [15](#), [17](#), [28](#), [29](#), [31–33](#)
- LearnerDens, [7](#), [11](#), [122](#)
- LearnerDensHistogram
  - (mlr\_learners\_dens.hist), [28](#)
- LearnerDensKDE (mlr\_learners\_dens.kde), [29](#)
- LearnerRegr, [21](#), [25](#), [26](#)
- LearnerSurv, [9](#), [9](#), [18](#), [19](#), [22](#), [23](#), [25](#), [26](#), [108](#), [121](#), [123](#)
- LearnerSurvCoxPH
  - (mlr\_learners\_surv.coxph), [31](#)
- LearnerSurvKaplan
  - (mlr\_learners\_surv.kaplan), [32](#)
- LearnerSurvRpart
  - (mlr\_learners\_surv.rpart), [33](#)
- lrn(), [28](#), [29](#), [31–33](#)
- lung, [103](#)
- mean(), [12](#), [14](#)
- Measure, [11](#), [13](#), [36](#), [38](#), [39](#), [41](#), [43](#), [45](#), [47](#), [49](#), [51](#), [53](#), [55](#), [57](#), [59](#), [61](#), [62](#), [64](#), [65](#), [67](#), [68](#), [70](#), [72](#), [73](#), [75](#), [77](#), [79](#), [81](#), [83](#), [85](#), [87](#), [89](#), [90](#)
- MeasureDens, [11](#), [14](#)
- MeasureDensLogloss
  - (mlr\_measures\_dens.logloss), [35](#)
- MeasureSurv, [12](#), [13](#)
- MeasureSurvAUC, [14](#)
- MeasureSurvBeggC
  - (mlr\_measures\_surv.beggC), [36](#)
- MeasureSurvBrier
  - (mlr\_measures\_surv.graf), [47](#)
- MeasureSurvBrierSE
  - (mlr\_measures\_surv.grafSE), [49](#)
- MeasureSurvCalibrationAlpha
  - (mlr\_measures\_surv.calib\_alpha), [37](#)
- MeasureSurvCalibrationBeta
  - (mlr\_measures\_surv.calib\_beta), [39](#)
- MeasureSurvChamblessAUC
  - (mlr\_measures\_surv.chambless\_auc), [41](#)
- MeasureSurvCindex
  - (mlr\_measures\_surv.cindex), [43](#)
- MeasureSurvGonenC
  - (mlr\_measures\_surv.gonenC), [45](#)
- MeasureSurvGraf, [49](#), [108](#)
- MeasureSurvGraf
  - (mlr\_measures\_surv.graf), [47](#)
- MeasureSurvGrafSE
  - (mlr\_measures\_surv.grafSE), [49](#)
- MeasureSurvHarrellC
  - (mlr\_measures\_surv.harrellC), [51](#)
- MeasureSurvHungAUC
  - (mlr\_measures\_surv.hung\_auc), [52](#)
- MeasureSurvIntegrated, [16](#)
- MeasureSurvIntLogloss, [57](#), [108](#), [109](#)
- MeasureSurvIntLogloss
  - (mlr\_measures\_surv.intlogloss), [54](#)
- MeasureSurvIntLoglossSE
  - (mlr\_measures\_surv.intloglossSE), [57](#)
- MeasureSurvLogloss, [61](#)
- MeasureSurvLogloss
  - (mlr\_measures\_surv.logloss), [59](#)
- MeasureSurvLoglossSE
  - (mlr\_measures\_surv.loglossSE), [61](#)
- MeasureSurvMAE, [64](#)
- MeasureSurvMAE (mlr\_measures\_surv.mae), [62](#)
- MeasureSurvMAESE
  - (mlr\_measures\_surv.maeSE), [64](#)
- MeasureSurvMSE, [67](#)

- MeasureSurvMSE (mlr\_measures\_surv.mse), 65
- MeasureSurvMSESE (mlr\_measures\_surv.mseSE), 67
- MeasureSurvNagelkR2 (mlr\_measures\_surv.nagelk\_r2), 68
- MeasureSurvOQuigleyR2 (mlr\_measures\_surv.oquigley\_r2), 70
- MeasureSurvRMSE, 73
- MeasureSurvRMSE (mlr\_measures\_surv.rmse), 71
- MeasureSurvRMSESE (mlr\_measures\_surv.rmseSE), 73
- MeasureSurvSchmid (mlr\_measures\_surv.schmid), 74
- MeasureSurvSongAUC (mlr\_measures\_surv.song\_auc), 77
- MeasureSurvSongTNR (mlr\_measures\_surv.song\_tnr), 79
- MeasureSurvSongTPR (mlr\_measures\_surv.song\_tpr), 81
- MeasureSurvUnoAUC (mlr\_measures\_surv.uno\_auc), 84
- MeasureSurvUnoC (mlr\_measures\_surv.unoC), 83
- MeasureSurvUnoTNR (mlr\_measures\_surv.uno\_tnr), 86
- MeasureSurvUnoTPR (mlr\_measures\_surv.uno\_tpr), 88
- MeasureSurvXuR2 (mlr\_measures\_surv.xu\_r2), 90
- MixtureDistribution, 24
- mlr3::Learner, 8, 9, 28, 30–32, 34
- mlr3::Measure, 11, 13, 14, 16, 35, 36, 38, 40, 41, 43, 45, 48, 50, 51, 53, 55, 57, 59, 61, 63–65, 67, 69, 70, 72, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91
- mlr3::mlr\_measures, 11, 13
- mlr3::mlr\_task\_generators, 106, 108
- mlr3::mlr\_tasks, 105, 126
- mlr3::Prediction, 114, 122, 123
- mlr3::Task, 118, 125–127
- mlr3::TaskGenerator, 106, 107
- mlr3::TaskSupervised, 126, 127
- mlr3pipelines::Graph, 18–24, 27
- mlr3pipelines::GraphLearner, 18, 20–22, 24, 27
- mlr3pipelines::mlr\_pipeops, 110, 112, 116
- mlr3pipelines::PipeOp, 92, 94, 96, 97, 99, 111, 113, 115, 117, 118, 120
- mlr3pipelines::PipeOpEnsemble, 92, 93
- mlr3pipelines::PipeOpLearner, 18, 19, 21, 23
- mlr3pipelines::po(), 110, 112, 116
- mlr3proba (mlr3proba-package), 4
- mlr3proba-package, 4
- mlr3proba::LearnerDens, 28, 30
- mlr3proba::LearnerSurv, 31, 32, 34
- mlr3proba::MeasureDens, 35
- mlr3proba::MeasureSurv, 14, 16, 36, 38, 40, 41, 43, 45, 48, 50, 51, 53, 55, 57, 59, 61, 63–65, 67, 69, 70, 72, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91
- mlr3proba::MeasureSurvAUC, 41, 53, 77, 79, 81, 85, 87, 89
- mlr3proba::MeasureSurvIntegrated, 14, 41, 48, 50, 53, 55, 57, 75, 77, 79, 81, 85, 87, 89
- mlr3proba::MeasureSurvLogloss, 61
- mlr3proba::PipeOpPredTransformer, 94, 96
- mlr3proba::PipeOpTaskTransformer, 97, 99
- mlr3proba::PipeOpTransformer, 94, 96, 97, 99, 115, 118
- mlr\_graphs\_crankcompositor, 17, 20–22, 24, 27
- mlr\_graphs\_distrcompositor, 18, 19, 21, 22, 24, 27
- mlr\_graphs\_probregrcompositor, 18, 20, 20, 22, 24, 27
- mlr\_graphs\_survaverager, 18, 20, 21, 22, 24, 27
- mlr\_graphs\_survbagging, 18, 20–22, 23, 27
- mlr\_graphs\_survtoregr, 18, 20–22, 24, 25
- mlr\_learners, 28, 29, 31–33
- mlr\_learners\_dens.hist, 28, 30
- mlr\_learners\_dens.kde, 29, 29
- mlr\_learners\_surv.coxph, 31, 33, 34
- mlr\_learners\_surv.kaplan, 32, 32, 34

- `mlr_learners_surv.rpart`, 32, 33, 33  
`mlr_measures`, 36, 38, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 62, 64, 65, 67, 68, 70, 72, 73, 75, 77, 79, 81, 83, 85, 87, 89, 90  
`mlr_measures_dens.logloss`, 35  
`mlr_measures_surv.beggC`, 36, 39–42, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78–80, 82–84, 86, 88, 90, 91  
`mlr_measures_surv.brier`  
    (`mlr_measures_surv.graf`), 47  
`mlr_measures_surv.brier_se`  
    (`mlr_measures_surv.grafSE`), 49  
`mlr_measures_surv.calib_alpha`, 37, 37, 40, 42, 45, 46, 49–52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.calib_beta`, 37, 39, 39, 42, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78–80, 82–84, 86, 88, 90, 91  
`mlr_measures_surv.chambless_auc`, 37, 39–41, 41, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78–80, 82–84, 86, 88, 90, 91  
`mlr_measures_surv.cindex`, 37, 39, 40, 42, 43, 46, 49, 50, 52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.gonenC`, 37, 39–42, 45, 45, 49, 50, 52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78–80, 82–84, 86, 88, 90, 91  
`mlr_measures_surv.graf`, 37, 39, 40, 42, 45, 46, 47, 50–52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.grafSE`, 37, 39, 40, 42, 45, 46, 49, 49, 52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.harrellC`, 37, 39, 40, 42, 45, 46, 49, 50, 51, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.hung_auc`, 37, 39–42, 45, 46, 49, 50, 52, 52, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78–80, 82–84, 86, 88, 90, 91  
`mlr_measures_surv.intlogloss`, 37, 39, 40, 42, 45, 46, 49–52, 54, 54, 58, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.intloglossSE`, 37, 39, 40, 42, 45, 46, 49–52, 54, 56, 57, 60, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.logloss`, 37, 39, 40, 42, 45, 46, 49–52, 54, 56, 58, 59, 62, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.loglossSE`, 37, 39, 40, 42, 45, 46, 49–52, 54, 56, 58, 60, 61, 63, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.mae`, 37, 39, 40, 42, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62, 62, 65, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.maeSE`, 37, 39, 40, 42, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62, 63, 64, 66, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.mse`, 37, 39, 40, 42, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62, 63, 65, 65, 68, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.mseSE`, 37, 39, 40, 42, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62, 63, 65, 66, 67, 69, 71, 73, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.nagelk_r2`, 37, 39–42, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 68, 71, 73, 74, 76, 78–80, 82–84, 86, 88, 90, 91  
`mlr_measures_surv.oquigley_r2`, 37, 39–42, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 70, 73, 74, 76, 78–80, 82–84, 86, 88, 90, 91  
`mlr_measures_surv.rmse`, 37, 39, 40, 42, 45, 46, 49, 51, 52, 54, 56, 58, 60, 62, 63, 65, 66, 68, 69, 71, 71, 74, 76, 78, 80, 82, 84, 86, 88, 90, 91  
`mlr_measures_surv.rmseSE`, 37, 39, 40, 42, 45, 46, 49, 50, 52, 54, 56, 58, 60, 62,

- 63, 65, 66, 68, 69, 71, 73, 73, 76, 78,  
80, 82, 84, 86, 88, 90, 91
- mlr\_measures\_surv.schmid, 37, 39, 40, 42,  
45, 46, 49, 51, 52, 54, 56, 58, 60, 62,  
63, 65, 66, 68, 69, 71, 73, 74, 74, 78,  
80, 82, 84, 86, 88, 90, 91
- mlr\_measures\_surv.song\_auc, 37, 39–42,  
45, 46, 49, 51, 52, 54, 56, 58, 60, 62,  
63, 65, 66, 68, 69, 71, 73, 74, 76, 77,  
80, 82–84, 86, 88, 90, 91
- mlr\_measures\_surv.song\_tnr, 37, 39–42,  
45, 46, 49, 51, 52, 54, 56, 58, 60, 62,  
63, 65, 66, 68, 69, 71, 73, 74, 76, 78,  
79, 79, 82–84, 86, 88, 90, 91
- mlr\_measures\_surv.song\_tpr, 37, 39–42,  
45, 46, 49, 51, 52, 54, 56, 58, 60, 62,  
63, 65, 66, 68, 69, 71, 73, 74, 76,  
78–80, 81, 84, 86, 88, 90–92
- mlr\_measures\_surv.uno\_auc, 37, 39–42, 45,  
46, 49, 51, 52, 54, 56, 58, 60, 62, 63,  
65, 66, 68, 69, 71, 73, 74, 76, 78–80,  
82–84, 84, 88, 90–92
- mlr\_measures\_surv.uno\_tnr, 37, 39–42, 45,  
46, 49, 51, 52, 54, 56, 58, 60, 62, 63,  
65, 66, 68, 69, 71, 73, 74, 76, 78–80,  
82–84, 86, 86, 90–92
- mlr\_measures\_surv.uno\_tpr, 37, 39–42, 45,  
46, 49, 51, 52, 54, 56, 58, 60, 62, 63,  
65, 66, 68, 69, 71, 73, 74, 76, 78–80,  
82–84, 86, 88, 88, 91, 92
- mlr\_measures\_surv.unoC, 37, 39, 40, 42, 45,  
46, 49, 51, 52, 54, 56, 58, 60, 62, 63,  
65, 66, 68, 69, 71, 73, 74, 76, 78, 80,  
82, 83, 86, 88, 90, 91
- mlr\_measures\_surv.xu\_r2, 37, 39–42, 45,  
46, 49, 51, 52, 54, 56, 58, 60, 62, 63,  
65, 66, 68, 69, 71, 73, 74, 76, 78–80,  
82–84, 86, 88, 90, 90
- mlr\_pipeops\_crankcompose  
(PipeOpCrankCompositor), 110
- mlr\_pipeops\_distrcompose  
(PipeOpDistrCompositor), 112
- mlr\_pipeops\_probregr  
(PipeOpProbregrCompositor), 116
- mlr\_pipeops\_survavg, 92, 95, 96, 98, 100,  
116, 119, 121
- mlr\_pipeops\_trafopred\_regrsurv, 93, 93,  
96, 98, 100, 116, 119, 121
- mlr\_pipeops\_trafopred\_survreg, 93, 95,  
95, 98, 100, 116, 119, 121
- mlr\_pipeops\_trafotask\_regrsurv, 93, 95,  
96, 97, 100, 116, 119, 121
- mlr\_pipeops\_trafotask\_survreg, 93, 95,  
96, 98, 98, 116, 119, 121
- mlr\_reflections\$learner\_predict\_types,  
8, 10, 12, 14, 17
- mlr\_reflections\$learner\_properties, 8,  
10
- mlr\_reflections\$measure\_properties, 12,  
14, 15, 17
- mlr\_reflections\$task\_feature\_types, 8,  
10
- mlr\_task\_generators, 106, 107
- mlr\_task\_generators\_simdens, 106
- mlr\_task\_generators\_simsurv, 107
- mlr\_tasks, 125
- mlr\_tasks\_actg, 101
- mlr\_tasks\_faithful, 102
- mlr\_tasks\_gbcs, 102
- mlr\_tasks\_grace, 103
- mlr\_tasks\_lung, 103
- mlr\_tasks\_precip, 104
- mlr\_tasks\_rats, 104
- mlr\_tasks\_unemployment, 105
- mlr\_tasks\_whas, 105
- msr(), 36, 38, 39, 41, 43, 45, 47, 49, 51, 53,  
55, 57, 59, 61, 62, 64, 65, 67, 68, 70,  
72, 73, 75, 77, 79, 81, 83, 85, 87, 89,  
90
- Normal, 26
- paradox::ParamSet, 8, 10, 115, 119, 120
- pecs, 108
- pipeline\_crankcompositor, 111
- pipeline\_crankcompositor  
(mlr\_graphs\_crankcompositor),  
17
- pipeline\_distrcompositor, 114
- pipeline\_distrcompositor  
(mlr\_graphs\_distrcompositor),  
19
- pipeline\_probregrcompositor  
(mlr\_graphs\_probregrcompositor),  
20
- pipeline\_survaverager  
(mlr\_graphs\_survaverager), 22

- pipeline\_survbagging  
(mlr\_graphs\_survbagging), 23
- pipeline\_survtoregr  
(mlr\_graphs\_survtoregr), 25
- PipeOp, 25, 110, 112, 114–116, 118–120
- PipeOpCrankCompositor, 17, 110, 110, 114
- PipeOpDistrCompositor, 19, 25, 26, 111, 112, 112
- PipeOpEnsemble, 92
- PipeOpLearner, 92
- PipeOpLearnerCV, 26
- PipeOpPredRegrSurv, 26
- PipeOpPredRegrSurv  
(mlr\_pipeops\_trafopred\_regrsurv), 93
- PipeOpPredSurvRegr  
(mlr\_pipeops\_trafopred\_survregr), 95
- PipeOpPredTransformer, 93–96, 98, 100, 114, 114, 115, 119, 121
- PipeOpProbRegrCompositor, 21, 26, 116, 116
- PipeOpSubsample, 23, 24
- PipeOpSurvAvg, 22–24
- PipeOpSurvAvg (mlr\_pipeops\_survavg), 92
- PipeOpTaskRegrSurv  
(mlr\_pipeops\_trafotask\_regrsurv), 97
- PipeOpTaskSurvRegr, 26
- PipeOpTaskSurvRegr  
(mlr\_pipeops\_trafotask\_survregr), 98
- PipeOpTaskTransformer, 93, 95–98, 100, 116, 118, 118, 121
- PipeOpTransformer, 93, 95, 96, 98, 100, 116, 119, 119, 120
- plot.LearnerSurv, 121
- precip, 104
- Prediction, 7, 9, 114, 115, 119, 120
- PredictionDens, 7, 122, 125
- PredictionRegr, 26, 93–95, 116
- PredictionSurv, 9, 26, 92–95, 108–110, 112, 113, 123, 123
  
- R6, 8, 10, 11, 13, 15, 29–31, 33–36, 38, 40, 42, 46, 48, 50, 51, 53, 55, 58, 60, 61, 63, 64, 66, 67, 69, 70, 72, 74, 75, 78, 80, 82, 84, 85, 87, 89, 91, 92, 94, 96, 97, 99, 106, 107, 111, 113, 115, 117, 118, 120, 122, 124, 125, 127
- R6::R6Class, 101–105
- rats, 104
- requireNamespace(), 9, 10, 12, 14, 17, 115, 119, 120
- Resampling, 12, 14, 15, 17
- rpart::predict.rpart(), 33
- rpart::rpart(), 33
  
- simSurv::simSurv(), 107
- surv.harrellC, 14
- surv.kaplan, 112
- survAUC::AUC.cd(), 41
- survAUC::AUC.hc(), 52
- survAUC::AUC.sh(), 77
- survAUC::AUC.uno(), 84
- survAUC::BeggC(), 36
- survAUC::GHCI(), 45
- survAUC::Nagelk(), 68
- survAUC::OXS(), 70
- survAUC::sens.sh(), 81
- survAUC::sens.uno(), 88
- survAUC::spec.sh(), 79
- survAUC::spec.uno(), 86
- survAUC::UnoC(), 83
- survAUC::X0(), 90
- survival::coxph(), 31
- survival::predict.coxph(), 31
- survival::Surv, 128
- survival::survfit(), 32
- survival::survfit.coxph(), 31
  
- Task, 12, 14, 15, 17, 119, 120, 125
- TaskDens, 102, 104, 125, 128
- TaskGenerator, 106, 107
- TaskGenerators, 106, 108
- TaskGeneratorSimdens  
(mlr\_task\_generators\_simdens), 106
- TaskGeneratorSimsurv  
(mlr\_task\_generators\_simsurv), 107
- TaskRegr, 25, 26, 97, 98
- Tasks, 105
- TaskSurv, 26, 97, 98, 101–105, 109, 121, 122, 124, 126, 126
- tgen(), 106, 107



VectorDistribution, [124](#)

whas, [105](#), [129](#)