Package 'mlr3proba'

July 25, 2020

Title Probabilistic Supervised Learning for 'mlr3'

Version 0.2.0

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

License LGPL-3

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

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

Depends R (>= 3.1.0)

Imports checkmate, data.table, distr6 (>= 1.4.2), mlr3 (>= 0.3.0), mlr3misc (>= 0.1.7), mlr3pipelines, paradox (>= 0.1.0), Rcpp (>= 1.0.4), R6, survival

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

LinkingTo Rcpp

VignetteBuilder knitr

ByteCompile true

Encoding UTF-8

LazyData true

NeedsCompilation yes

RoxygenNote 7.1.0.9000

RdMacros mlr3misc

Collate 'LearnerDens.R' 'LearnerDensHistogram.R' 'LearnerDensKDE.R' 'LearnerProbreg.R' 'LearnerProbregGaussian.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' 'PipeOpProbRegr.R' 'PipeOpSurvAvg.R'
'PredictionDens.R' 'PredictionProbreg.R' 'PredictionRegr.R'
'PredictionSurv.R' 'RcppExports.R' 'TaskDens.R'
'TaskDens_faithful.R' 'TaskDens_precip.R'
'TaskGeneratorSimdens.R' 'TaskGeneratorSimsurv.R'
'TaskProbreg.R' 'TaskSurv.R' 'TaskSurv_lung.R'
'TaskSurv_rats.R' 'TaskSurv_unemployment.R' 'assertions.R'
'cindex.R' 'crank_compositor.R' 'distr_compositor.R'
'helpers.R' 'histogram.R' 'integrated_scores.R' 'pecs.R'
'plot.R' 'probregr_compose.R' 'surv_average.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@ucl.ac.uk>

Repository CRAN

Date/Publication 2020-07-25 21:10:02 UTC

R topics documented:

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

Description

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

Author(s)

Maintainer: Raphael Sonabend <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

crankcompositor *Compose a Crank Predict Type for Survival Learners*

Description

This is a wrapper around the PipeOpCrankCompositor pipe operation, which simplifies graph creation.

Usage

```
crankcompositor(
  learner,
  method = c("mean", "median", "mode"),
  which = NULL,
  response = FALSE,
  param_vals = list()
)
```

Arguments

learner	LearnerSurv object for which a crank is composed (or over-written)
method	One of mean, mode, or median; abbreviations allowed. Used to determine how crank is estimated from the predicted distr. Default is mean.
which	If method = "mode" then specifies which mode to use if multi-modal, default is the first.
response	If TRUE then the response predict type is imputed with the same values as crank.
param_vals	Additional parameters to pass to the learner.

Details

For full details see PipeOpCrankCompositor.

Value

mlr3pipelines::GraphLearner

Examples

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

task = tgen("simsurv")$generate(20)
cox.crank = crankcompositor(
  learner = lrn("surv.coxph"),
  method = "median")
```

```
resample(task, cox.crank, rsmp("cv", folds = 2))$predictions()

## End(Not run)
```

`distrcompositor` *Compose a Distr Predict Type for Survival Learners*

Description

This is a wrapper around the PipeOpDistrCompositor pipe operation, which simplifies graph creation.

Usage

```
distrcompositor(
  learner,
  estimator = c("kaplan", "nelson"),
  form = c("aft", "ph", "po"),
  overwrite = FALSE,
  param_vals = list()
)
```

Arguments

`learner`	LearnerSurv object for which a `distr` is composed (or over-written).
`estimator`	One of `kaplan` (default) or `nelson`, corresponding to the Kaplan-Meier and Nelson-Aalen estimators respectively. Used to estimate the baseline survival distribution. Abbreviations allowed.
`form`	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. 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.
`param_vals`	Additional parameters to pass to the `learner`.

Details

For full details see PipeOpDistrCompositor.

Value

mlr3pipelines::GraphLearner

Examples

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

task = tgen("simsurv")$generate(20)
cox.distr = distrcompositor(
  learner = lrn("surv.coxph"),
  estimator = "kaplan",
  form = "aft")

resample(task, cox.distr, rsmp("cv", folds = 2))$predictions()

## End(Not run)
```

`LearnerDens` *Density Learner*

Description

This Learner specializes Learner for density estimation problems:

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

Super class

`mlr3::Learner` -> `LearnerDens`

Methods

Public methods:

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

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

Usage:

```
LearnerDens$new(
  id,
  param_set = ParamSet$new(),
  predict_types = "cdf",
  feature_types = character(),
  properties = character(),
```

```
  data_formats = "data.table",
  packages = character(),
  man = NA_character_
)
```

Arguments:

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

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

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

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

`properties (character())`
: Set of properties of the Learner. Must be a subset of `mlr_reflections$learner_properties`.
The following properties are currently standardized and understood by learners in **mlr3**:
  - `"missings"`: The learner can handle missing values in the data.
  - `"weights"`: The learner supports observation weights.
  - `"importance"`: The learner supports extraction of importance scores, i.e. comes with an $importance() extractor function (see section on optional extractors in Learner).
  - `"selected_features"`: The learner supports extraction of the set of selected features, i.e. comes with a $selected_features() extractor function (see section on optional extractors in Learner).
  - `"oob_error"`: The learner supports extraction of estimated out of bag error, i.e. comes with a `oob_error()` extractor function (see section on optional extractors in Learner).

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

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

`man (character(1))`
: String in the format [pkg]::[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Learner: `LearnerSurv`

Examples

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

# get a specific learner from mlr_learners:
mlr_learners$get("dens.hist")
lrn("dens.hist")
```

LearnerSurv *Survival Learner*

Description

This Learner specializes Learner for survival problems:

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

Super class

`mlr3::Learner` -> `LearnerSurv`

Methods

Public methods:

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

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

Usage:

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

Arguments:

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

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

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

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

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

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

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

`man (character(1))`
String in the format [pkg]::[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Learner: `LearnerDens`

Examples

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

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

MeasureDens *Density Measure*

Description

This measure specializes Measure for survival problems.

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

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

Super class

mlr3::Measure -> MeasureDens

Methods

Public methods:

- MeasureDens$new()

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

Usage:

```
MeasureDens$new(
  id,
  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 `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.

`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_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.

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.

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.

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: `lp`

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::aidsim511>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_au` `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_a` `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 δ is the observed censoring indicator from the test data, H_i is the predicted cumulative hazard, and t_i is the observed survival time.

The standard error is given by

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

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

Dictionary

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

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

Meta Information

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

Super classes

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

Active bindings

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

Methods

Public methods:

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

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

Usage:

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

Arguments:

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

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Van Houwelingen 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.

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_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 calibration survival measures: mlr_measures_surv.calib_beta

Other distr survival measures: mlr_measures_surv.grafSE, mlr_measures_surv.graf, mlr_measures_surv.intloglos
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 β coefficient is equal to 1.

Assumes fitted model is Cox PH.

Dictionary

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

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

Meta Information

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

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.

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_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 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.

See Also

Other survival measures: `mlr_measures_surv.beggC`, `mlr_measures_surv.calib_alpha`, `mlr_measures_surv.calib_b`
`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_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 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_`

`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.

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.

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

See Also

Other survival measures: mlr_measures_surv.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.intlo
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: `lp`

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.

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.grafSE,
mlr_measures_surv.graf, mlr_measures_surv.harrellC, mlr_measures_surv.hung_auc, mlr_measures_surv.intlo
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.harrellC,
mlr_measures_surv.unoC

Other lp survival measures: mlr_measures_surv.beggC, mlr_measures_surv.calib_beta, mlr_measures_surv.chambl
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_r

`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.

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_a
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 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:

```
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.

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.intlo
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.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.intlogl`
`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: lp

Super classes

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

Methods

Public methods:

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

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

Usage:

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

Arguments:

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

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

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

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

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.intlogl
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 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.chambl
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_r

---

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.

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_au`
`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.

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_au
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_au
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_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.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_au
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_au
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_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.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_au
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_au`
`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_au
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: lp

Super classes

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

Methods

Public methods:

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

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

Usage:

```
MeasureSurvNagelkR2$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Nagelkerke 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_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_au
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_tr
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.chambl
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_r

`mlr_measures_surv.oquigley_r2`
O'Quigley, Xu, and Stare's R2 Survival Measure

Description

Calls `survAUC::OXS()`.

Assumes Cox PH model specification.

Details

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

Dictionary

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

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

Meta Information

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

Super classes

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

Methods

Public methods:

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

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

Usage:

```
MeasureSurvOQuigleyR2$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

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

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_au
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_tr
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.chambl
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_r

---

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_au
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_au`
`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_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.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.

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.

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_au`
`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_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.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: `lp`

Super classes

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

Active bindings

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

Methods

Public methods:

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

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

Usage:

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

Arguments:

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

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

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

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_au
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_tr
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.chambl
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_

`mlr_measures_surv.song_tnr`

Song and Zhou's TNR Survival Measure

Description

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

Assumes Cox PH model specification.

`times` and `lp_thresh` are arbitrarily set to `0` to prevent crashing, these should be further specified.

Details

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

Dictionary

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

```
MeasureSurvSongTNR$new()
mlr_measures$get("surv.song_tnr")
msr("surv.song_tnr")
```

Meta Information

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

Super classes

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

Active bindings

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

Methods

Public methods:

- `MeasureSurvSongTNR$new()`
- `MeasureSurvSongTNR$clone()`

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

Usage:

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

Arguments:

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

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Song 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_au`
`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_tr`
`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.chambl`
`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_r`

`mlr_measures_surv.song_tpr` *Song and Zhou's TPR Survival Measure*

Description

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

Assumes Cox PH model specification.

`times` and `lp_thresh` are arbitrarily set to `0` to prevent crashing, these should be further specified.

Details

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

Dictionary

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

```
MeasureSurvSongTPR$new()
mlr_measures$get("surv.song_tpr")
msr("surv.song_tpr")
```

Meta Information

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

Super classes

`mlr3::Measure` -> `mlr3proba::MeasureSurv` -> `mlr3proba::MeasureSurvIntegrated` -> `mlr3proba::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_au
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_tn
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.chambl
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_

`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.

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_au
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::MeasureSurvAUC`
-> `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 fort-Year Survivors With Censored Regression Models." *Journal of the American Statistical Association*, **102**(478), 527–537. doi: 10.1198/016214507000000149.

See Also

Other survival measures: mlr_measures_surv.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_au
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_t
mlr_measures_surv.uno_tpr, mlr_measures_surv.xu_r2

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

Other lp survival measures: mlr_measures_surv.beggC, mlr_measures_surv.calib_beta, mlr_measures_surv.chambl
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_tnr, mlr_measures_surv.uno_tpr,
mlr_measures_surv.xu_r2

---

mlr_measures_surv.uno_tnr

Uno's TNR Survival Measure

Description

Calls survAUC::spec.uno().

Assumes random censoring.

times and lp_thresh are arbitrarily set to 0 to prevent crashing, these should be further specified.

Details

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

Dictionary

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

```
MeasureSurvUnoTNR$new()
mlr_measures$get("surv.uno_tnr")
msr("surv.uno_tnr")
```

Meta Information

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

Super classes

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

Active bindings

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

Methods

Public methods:

- MeasureSurvUnoTNR$new()
- MeasureSurvUnoTNR$clone()

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

Usage:

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

Arguments:

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

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

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Uno H, Cai T, Tian L, Wei LJ (2007). "Evaluating Prediction Rules fort-Year Survivors With Censored Regression Models." *Journal of the American Statistical Association*, **102**(478), 527–537. doi: 10.1198/016214507000000149.

See Also

Other survival measures: mlr_measures_surv.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_au
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_a
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.chambl
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::MeasureSurvAUC
-> 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 fort-Year Survivors With Censored Regression Models." *Journal of the American Statistical Association*, **102**(478), 527–537. doi: 10.1198/016214507000000149.

See Also

Other survival measures: mlr_measures_surv.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_au
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_a
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.chambl
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::XO().

Assumes Cox PH model specification.

Details

All measures implemented from **survAUC** should be used with care, we are aware of problems in implementation that sometimes cause fatal errors in R. In future updates 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.

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_au
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_a
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.chambl
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 PredictionSurvs by connecting `PipeOpSurvAvg` to multiple PipeOpLearner outputs.

The resulting prediction will aggregate any predict types that are contained within all inputs. Any predict types missing from at least one input will be set to `NULL`. These are aggregated as follows:

- `"response"`, `"crank"`, and `"lp"` are all a weighted average from the incoming predictions.
- `"distr"` is a distr6::VectorDistribution containing distr6::MixtureDistributions.

Weights can be set as a parameter; if none are provided, defaults to equal weights for each prediction.

Format

R6Class inheriting from mlr3pipelines::PipeOp.

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.

Fields

Only fields inherited from PipeOpEnsemble/PipeOp.

Methods

Only methods inherited from PipeOpEnsemble/PipeOp.

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, default `"survavg"`.

`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.

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_tasks_faithful *Old Faithful Eruptions Density Task*

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_lung *Lung Cancer Survival Task*

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 *Annual Precipitation Density Task*

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 *Rats Survival Task*

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_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 LearnerSurvs.

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	(PredictionSurv or list of LearnerSurvs)
measure	(character(1)) Either "graf" for MeasureSurvGraf, or "logloss" for MeasureSurvIntLogloss
times	(numeric()) If provided then either a vector of time-points to evaluate measure or a range of time-points.
n	(integer()) If times is missing or given as a range, then n provide number of time-points to evaluate measure over.
eps	(numeric()) Small error value to pass to MeasureSurvIntLogloss to prevent errors resulting from a log(0) calculation.
...	Additional arguments.
task	(TaskSurv)
row_ids	(integer()) Passed to Learner$predict.
newdata	(data.frame()) If not missing Learner$predict_newdata is called instead of Learner$predict.

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`.

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.

Fields

Only fields inherited from PipeOp.

Methods

Only fields inherited from PipeOp.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpCrankCompositor`

Methods

Public methods:

- `PipeOpCrankCompositor$new()`
- `PipeOpCrankCompositor$train_internal()`
- `PipeOpCrankCompositor$predict_internal()`
- `PipeOpCrankCompositor$clone()`

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

Usage:

```
PipeOpCrankCompositor$new(
  id = "crankcompose",
  param_vals = list(method = "mean")
)
```

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 `train_internal()`: train_internal Internal `train` function, will be moved to `private` in a near-future update, should be ignored.

Usage:

```
PipeOpCrankCompositor$train_internal(inputs)
```

Arguments:

`inputs` Ignore.

Method `predict_internal()`: predict_internal Internal `predict` function, will be moved to `private` in a near-future update, should be ignored.

Usage:

```
PipeOpCrankCompositor$predict_internal(inputs)
```

Arguments:

`inputs` Ignore.

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

mlr3pipelines::PipeOp and crankcompositor

Other survival compositors: `PipeOpDistrCompositor`, `PipeOpProbregrCompositor`

Examples

```
## Not run:
library(mlr3)
library(mlr3pipelines)
set.seed(1)

# Three methods to predict a `crank` from `surv.rpart`
task = tgen("simsurv")$generate(30)

# Method 1 - Train and predict separately then compose
learn = lrn("surv.coxph")$train(task)$predict(task)
poc = po("crankcompose", param_vals = list(method = "mean"))
poc$predict(list(learn))

# Method 2 - Create a graph manually
gr = Graph$new()$
  add_pipeop(po("learner", lrn("surv.coxph")))$
```

```
  add_pipeop(po("crankcompose"))$
  add_edge("surv.coxph", "crankcompose")
gr$train(task)
gr$predict(task)

# Method 3 - Syntactic sugar: Wrap the learner in a graph
cox.crank = crankcompositor(
  learner = lrn("surv.coxph"),
  method = "median")
resample(task, cox.crank, rsmp("cv", folds = 2))$predictions()

## End(Not run)
```

`PipeOpDistrCompositor` *PipeOpDistrCompositor*

Description

Estimates (or 'composes') a survival distribution from a predicted baseline `distr` and a `crank` or `lp` from two PredictionSurvs.

Compositor Assumptions:

- The baseline `distr` is a discrete estimator, e.g. surv.kaplan.
- The composed `distr` is of a linear form
- 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:

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

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

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

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.**

Fields

Only fields inherited from PipeOp.

Methods

Only methods inherited from PipeOp.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpDistrCompositor`

Methods

Public methods:

- `PipeOpDistrCompositor$new()`
- `PipeOpDistrCompositor$train_internal()`
- `PipeOpDistrCompositor$predict_internal()`
- `PipeOpDistrCompositor$clone()`

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

Usage:

```
PipeOpDistrCompositor$new(
  id = "distrcompose",
  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 `train_internal()`**:** train_internal Internal `train` function, will be moved to `private` in a near-future update, should be ignored.

Usage:

```
PipeOpDistrCompositor$train_internal(inputs)
```

Arguments:

`inputs` Ignore.

Method `predict_internal()`**:** predict_internal Internal `predict` function, will be moved to `private` in a near-future update, should be ignored.

Usage:

```
PipeOpDistrCompositor$predict_internal(inputs)
```

Arguments:

`inputs` Ignore.

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

mlr3pipelines::PipeOp and distrcompositor

Other survival compositors: `PipeOpCrankCompositor`, `PipeOpProbregrCompositor`

Examples

```
## Not run:
library(mlr3)
library(mlr3pipelines)
set.seed(42)

# Three methods to transform the cox ph predicted `distr` to an
#  accelerated failure time model
task = tgen("simsurv")$generate(30)

# Method 1 - Train and predict separately then compose
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))

# Method 2 - Create a graph manually
gr = Graph$new()$
  add_pipeop(po("learner", lrn("surv.kaplan")))$
  add_pipeop(po("learner", lrn("surv.coxph")))$
  add_pipeop(po("distrcompose"))$
  add_edge("surv.kaplan", "distrcompose", dst_channel = "base")$
  add_edge("surv.coxph", "distrcompose", dst_channel = "pred")
gr$train(task)
gr$predict(task)

# Method 3 - Syntactic sugar: Wrap the learner in a graph.
cox.distr = distrcompositor(
  learner = lrn("surv.coxph"),
  estimator = "kaplan",
  form = "aft")
cox.distr$train(task)$predict(task)

## End(Not run)
```

`PipeOpProbregrCompositor` *PipeOpProbregrCompositor*

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()`:

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

Input and Output Channels

PipeOpProbregrCompositor has one input channel named "input", which takes NULL during training and PredictionSurv during prediction.

PipeOpProbregrCompositor 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.

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.

Fields

Only fields inherited from PipeOp.

Methods

Only fields inherited from PipeOp.

Super class

mlr3pipelines::PipeOp -> PipeOpProbregrCompositor

Methods

Public methods:

- PipeOpProbregrCompositor$new()
- PipeOpProbregrCompositor$train_internal()
- PipeOpProbregrCompositor$predict_internal()
- PipeOpProbregrCompositor$clone()

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

Usage:

```
PipeOpProbregrCompositor$new(
  id = "probregr_compose",
  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 `train_internal()`**:** train_internal Internal `train` function, will be moved to `private` in a near-future update, should be ignored.

Usage:

```
PipeOpProbregrCompositor$train_internal(inputs)
```

Arguments:

`inputs` Ignore.

Method `predict_internal()`**:** predict_internal Internal `predict` function, will be moved to `private` in a near-future update, should be ignored.

Usage:

```
PipeOpProbregrCompositor$predict_internal(inputs)
```

Arguments:

`inputs` Ignore.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

mlr3pipelines::PipeOp and crankcompositor

Other survival compositors: `PipeOpCrankCompositor`, `PipeOpDistrCompositor`

Examples

```
## Not run:
library(mlr3)
library(mlr3pipelines)
set.seed(1)

# Three methods to predict a `crank` from `surv.rpart`
task = tsk("boston_housing")
```

```
# Method 1 - Train and predict separately then compose
learn = lrn("regr.featureless", predict_type = "se")
pred = learn$train(task)$predict(task)
poc = po("probregr_compose")
poc$predict(list(pred))

# Method 2 - Create a graph manually
gr = Graph$new()$
  add_pipeop(po("learner", lrn("regr.featureless", predict_type = "se")))$
  add_pipeop(po("probregr_compose"))$
  add_edge("regr.featureless", "probregr_compose")
gr$train(task)
gr$predict(task)

# Method 3 - Syntactic sugar: Wrap the learner in a graph
feat_distr = probregr_compose(
  learner = lrn("regr.featureless", predict_type = "se"),
  dist = "Logistic")
resample(task, feat_distr, rsmp("cv", folds = 2))$predictions()

## End(Not run)
```

`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	(LearnerSurv)
task	(TaskSurv)
fun	(character) Passed to `distr6::plot.VectorDistribution`

row_ids	(integer()) Passed to Learner$predict
newdata	(data.frame()) If not missing Learner$predict_newdata is called instead of Learner$predict.
...	Additional arguments passed to distr6::plot.VectorDistribution

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: `PredictionRegr`, `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))
```

PredictionRegr *Prediction Object for Regression*

Description

This object wraps the predictions returned by a learner of class LearnerRegr, i.e. the predicted response and standard error.

This masks the mlr3::PredictionRegr in order to include the `distr` predict type.

Super class

`mlr3::Prediction` -> `PredictionRegr`

Active bindings

`response` (`numeric()`)
Access the stored predicted response.

`se` (`numeric()`)
Access the stored standard error.

`distr` (VectorDistribution)
Access the stored predicted survival distribution.

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

Methods

Public methods:

- `PredictionRegr$new()`

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

Usage:

```
PredictionRegr$new(
  task = NULL,
  row_ids = task$row_ids,
  truth = task$truth(),
  response = NULL,
  se = NULL,
  distr = NULL
)
```

Arguments:

`task` (TaskRegr)
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.

`response` (`numeric()`)
: Vector of numeric response values. One element for each observation in the test set.

`se` (`numeric()`)
: Numeric vector of predicted standard errors. One element for each observation in the test set.

`distr` (VectorDistribution)
: VectorDistribution from **distr6**. Each individual distribution in the vector represents the random variable 'survival time' for an individual observation.

See Also

Other Prediction: `PredictionDens`, `PredictionSurv`

Examples

```
library(mlr3)
task = tsk("boston_housing")
learner = lrn("regr.featureless", predict_type = "se")
p = learner$train(task)$predict(task)
p$predict_types
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 β 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, PredictionRegr

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))
```

`probregr_compose` *Create a Distr Predict Type for Regression Learners*

Description

This is a wrapper around the PipeOpProbregrCompositor pipe operation, which simplifies graph creation.

Usage

```
probregr_compose(learner, dist = "Normal", param_vals = list())
```

Arguments

`learner`	LearnerSurv object for which a `distr` is composed.
`dist`	Location-scale distribution to use for composition. Current possibilities are "Cauchy", "Gumbel", "Laplace", "Logistic", "Normal (default).
`param_vals`	Additional parameters to pass to the `learner`.

Details

For full details see PipeOpProbregrCompositor.

Value

mlr3pipelines::GraphLearner

Examples

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

task = tsk("boston_housing")
feat_distr = probregr_compose(
  learner = lrn("regr.featureless", predict_type = "se"),
  dist = "Logistic")
```

```
resample(task, feat_distr, rsmp("cv", folds = 2))$predictions()

## End(Not run)
```

`surv_averager` *Average Survival Predictions*

Description

This is a wrapper around the PipeOpSurvAvg pipe operation, which simplifies graph creation.

Usage

```
surv_averager(learners, param_vals = list())
```

Arguments

`learners`	`(list())` List of LearnerSurvs to average.
`param_vals`	`(list())` Parameters, including weights, to pass to PipeOpSurvAvg.

Details

For full details see PipeOpSurvAvg.

Value

mlr3pipelines::GraphLearner

Examples

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

task = tgen("simsurv")$generate(5)
avg = surv_averager(
  learners = lrns(c("surv.kaplan", "surv.coxph")),
  param_vals = list(weights = c(0.1, 0.9))
 )
avg$train(task)$predict(task)

## End(Not run)
```

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()
```

Index