Package: mlr3cmprsk (via r-universe)

May 11, 2026

Title Competing Risks Machine Learning for 'mlr3'

Version 0.0.5

Description Provides a unified interface for right-censored competing risks tasks in 'mlr3'.

License MIT + file LICENSE

Depends mlr3 (>= 1.4.0), R (>= 3.6.0)

Imports checkmate (>= 2.3.2), cmprsk, data.table (>= 1.17.8), mlr3misc (>= 0.19.0), paradox (>= 1.0.1), R6, riskRegression, survdistr (>= 0.0.3), survival

Suggests lgr, mirai, testthat (>= 3.0.0)

ByteCompile true

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE, r6 = TRUE)

RoxygenNote 7.3.3

Collate 'LearnerCompRisks.R' 'aaa.R' 'LearnerCompRisksAalenJohansen.R' 'LearnerCompRisksFineGray.R' 'MeasureCompRisks.R' 'MeasureCompRisksAUC.R' 'MeasureCompRisksBrierScore.R' 'PredictionCompRisks.R' 'PredictionDataCompRisks.R' 'TaskCompRisks.R' 'Task_pbc.R' 'as_prediction_cmprsk.R' 'as_task_cmprisk.R' 'assertions.R' 'bibentries.R' 'default_fallback.R' 'helpers.R' 'zzz.R'

Config/pak/sysreqs cmake make libicu-dev libuv1-dev

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

Date/Publication 2026-04-11 13:04:27 UTC

RemoteUrl https://github.com/mlr-org/mlr3cmprsk

RemoteRef v0.0.5

RemoteSha de51006a1e417469a34d365763b201110e2966cf

Contents

`mlr3cmprsk-package` *mlr3cmprsk: Competing Risks Machine Learning for 'mlr3'*

Description

Provides a unified interface for right-censored competing risks tasks in 'mlr3'.

Author(s)

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

Other contributors:

- Andrzej Galecki <agalecki@umich.edu> (ORCID) [contributor]

`as_prediction_cmprsk` *Convert to a Competing Risk Prediction*

Description

Convert object to a PredictionCompRisks.

Usage

```
as_prediction_cmprsk(x, ...)

## S3 method for class 'PredictionCompRisks'
as_prediction_cmprsk(x, ...)

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

Arguments

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

Value

PredictionCompRisks.

Examples

```
library(mlr3)
task = tsk("pbc")
learner = lrn("cmprsk.aalen")
learner$train(task)
p = learner$predict(task)

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

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

---

as_task_cmprsk *Convert to a Competing Risks Task*

Description

Convert object to a competing risks task (TaskCompRisks).

Usage

```
as_task_cmprsk(x, ...)

## S3 method for class 'TaskCompRisks'
as_task_cmprsk(x, clone = FALSE, ...)

## S3 method for class 'data.frame'
as_task_cmprsk(
  x,
  time = "time",
  event = "event",
  id = deparse(substitute(x)),
  ...
)
```

```
## S3 method for class 'DataBackend'
as_task_cmprsk(
  x,
  time = "time",
  event = "event",
  id = deparse(substitute(x)),
  ...
)
```

Arguments

x	(any) Object to convert, e.g. a `data.frame()`.
...	(any) Additional arguments.
clone	(`logical(1)`) If `TRUE`, ensures that the returned object is not the same as the input `x`.
time	(`character(1)`) Name of the column for outcome time.
event	(`character(1)`) Name of column giving that holds the event indicator. 0 corresponds to censoring, values > 0 correspond to different competing events.
id	(`character(1)`) Id for the new task. Defaults to the (deparsed and substituted) name of `x`.

`LearnerCompRisks` *Competing Risks Learner*

Description

This Learner specializes Learner for competing risks problems:

- `task_type` is set to `"cmprsk"`
- Creates Predictions of class PredictionCompRisks.
- The only currently available option for `predict_types` is `"cif"`, which represents the predicted **cumulative incidence function** for each observation in the test set.

Super class

`mlr3::Learner` -> `LearnerCompRisks`

Methods

Public methods:

- LearnerCompRisks$new()
- LearnerCompRisks$clone()

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

Usage:

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

Arguments:

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

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

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

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

`properties (character())`
: Set of properties of the Learner (see initialization method `$new()`. Must be a subset of `mlr_reflections$learner_properties`.

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

`label (character(1))`
: Label for the new instance.

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

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

# get a specific learner from mlr_learners:
mlr_learners$get("cmprsk.aalen")
lrn("cmprsk.aalen")
```

`MeasureCompRisks` *Competing Risks Measure*

Description

This measure specializes Measure for competing risk problems.

- `task_type` is set to `"cmprsk"`.
- `predict_type` is set to `"cif"`.

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

Super class

`mlr3::Measure` -> `MeasureCompRisks`

Methods

Public methods:

- `MeasureCompRisks$new()`
- `MeasureCompRisks$clone()`

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

Usage:

```
MeasureCompRisks$new(
  id,
  param_set = ps(),
  range,
  minimize = NA,
  average = "macro",
  aggregator = NULL,
  properties = character(),
  predict_type = "cif",
  predict_sets = "test",
  task_properties = character(),
  packages = character(),
  label = NA_character_,
  man = NA_character_
)
```

Arguments:

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

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

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

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

`average (character(1))`
How to average multiple Predictions from a ResampleResult.
The default, `"macro"`, calculates the individual performances scores for each Prediction and then uses the function defined in `$aggregator` to average them to a single number.
If set to `"micro"`, the individual Prediction objects are first combined into a single new Prediction object which is then used to assess the performance. The function in `$aggregator` is not used in this case.

`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.

`predict_sets (character())`
Prediction sets to operate on, used in `aggregate()` to extract the matching `predict_sets` from the ResampleResult. Multiple predict sets are calculated by the respective Learner during resample()/benchmark(). Must be a non-empty subset of `{"train", "test"}`. If multiple sets are provided, these are first combined to a single prediction object. Default is `"test"`.

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

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

`label (character(1))`
Label for the new instance.

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Default competing risks measure: `cmprsk.auc`

`mlr_learners_cmprsk.aalen`

Aalen Johansen Competing Risks Learner

Description

This learner estimates the Cumulative Incidence Function (CIF) for competing risks using the empirical Aalen-Johansen (AJ) estimator.

Transition probabilities to each competing event are computed from the training data via the `survival::survfit.formula()` function. Predictions are made at all **unique event times (across all causes)** observed in the training set.

Dictionary

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

```
mlr_learners$get("cmprsk.aalen")
lrn("cmprsk.aalen")
```

Meta Information

- Task type: "cmprsk"
- Predict Types: "cif"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3cmprsk**, **survival**

Parameters

Empty ParamSet

Super classes

`mlr3::Learner` -> `mlr3cmprsk::LearnerCompRisks` -> `LearnerCompRisksAalenJohansen`

Active bindings

`native_model` (survival::survfit)
The fitted model.

Methods

Public methods:

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

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

Usage:

```
LearnerCompRisksAalenJohansen$new()
```

Method `importance()`**:** All features have a score of `0` for this learner. This method exists solely for compatibility with the `mlr3` ecosystem, as this learner is used as a fallback for other survival learners that require an `importance()` method.

Usage:

```
LearnerCompRisksAalenJohansen$importance()
```

Returns: Named `numeric()`.

Method `selected_features()`**:** Selected features are always the empty set for this learner. This method is implemented only for compatibility with the `mlr3` API, as this learner does not perform feature selection.

Usage:

```
LearnerCompRisksAalenJohansen$selected_features()
```

Returns: `character(0)`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Aalen, O O, Johansen, Soren (1978). "An empirical transition matrix for non-homogeneous Markov chains based on censored observations." *Scandinavian journal of statistics*, 141–150.

See Also

Other competing risk learners: `mlr_learners_cmprsk.fg`

Examples

```
# Define the Learner (Aalen-Johansen/AJ estimator)
learner = lrn("cmprsk.aalen")
learner

# Define a Task
task = tsk("pbc")

# Stratification based on event
task$set_col_roles(cols = "status", add_to = "stratum")

# Create train and test set
part = partition(task)

# Train the learner on the training set
learner$train(task, row_ids = part$train)
learner$native_model

# Make predictions for the test set
predictions = learner$predict(task, row_ids = part$test)
predictions

# Score the predictions
# AJ has random discriminative performance
predictions$score(msr("cmprsk.auc", time = 100))

# Prediction error (Brier score) at specific time point
# BS(t) => weighted mean score across causes (default)
predictions$score(msr("cmprsk.brier", time = 100))
```

---

mlr_learners_cmprsk.fg

Fine-Gray Competing Risks Learner

Description

Fine-Gray subdistribution hazards model for competing risks using `cmprsk::crr()`.

Details

The fitted model is an S3 object of class `"fine_gray"` that stores a cause-specific list of `crr` class models.

At prediction time, each cause-specific model is evaluated with `cmprsk::predict.crr()` and the resulting CIF matrices are aligned to a common time grid across causes using constant CIF interpolation. The time grid is the **unique event times (across all causes)** observed in the training set.

Time-interaction terms (via `cov2`) are not implemented.

Dictionary

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

```
mlr_learners$get("cmprsk.fg")
lrn("cmprsk.fg")
```

Meta Information

- Task type: "cmprsk"
- Predict Types: "cif"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3cmprsk**, **cmprsk**

Parameters

Id	Type	Default	Levels	Range
cengroup	numeric	-		$(-\infty, \infty)$
gtol	numeric	1e-06		$[0, \infty)$
maxiter	integer	10		$[0, \infty)$
init	untyped	-		-
variance	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3cmprsk::LearnerCompRisks` -> `LearnerCompRisksFineGray`

Methods

Public methods:

- `LearnerCompRisksFineGray$new()`
- `LearnerCompRisksFineGray$clone()`

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

Usage:

```
LearnerCompRisksFineGray$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Fine, P J, Gray, J R (1999). "A Proportional Hazards Model for the Subdistribution of a Competing Risk." *Journal of the American Statistical Association*, **94**(446), 496–509. doi:10.1080/01621459.1999.10474144.

See Also

Other competing risk learners: `mlr_learners_cmprsk.aalen`

Examples

```
# Define the Learner
learner = lrn("cmprsk.fg")
learner

# Define a Task
task = tsk("pbc")

# Subset task features as Fine-Gray model doesn't accept factors
# Encode factors with `mlr3pipelines::po("encode")` if needed
feats = c("age", "chol", "albumin", "ast", "bili", "protime")
task$select(feats)

# Stratification based on event
task$set_col_roles(cols = "status", add_to = "stratum")

# Create train and test set
part = partition(task)

# Train the learner on the training set
learner$train(task, row_ids = part$train)
learner$native_model

# Make predictions for the test set
predictions = learner$predict(task, row_ids = part$test)
predictions

# Score the predictions
# AUC(t = 100), weighted mean score across causes (default)
predictions$score(msr("cmprsk.auc", cause = "mean", time = 100))

# AUC(t = 100), with user-specified weights
predictions$score(msr("cmprsk.auc", cause = "mean", cause_weights = c(0.2, 0.8),
  time = 100))

# AUC(t = 100), 1st cause
predictions$score(msr("cmprsk.auc", cause = 1, time = 100))

# AUC(t = 100), 2nd cause
predictions$score(msr("cmprsk.auc", cause = 2, time = 100))

# Prediction error (Brier score) at specific time point
```

```
# BS(t = 100) => weighted mean score across causes (default)
predictions$score(msr("cmprsk.brier", time = 100))

# BS(t = 100), 1st cause
predictions$score(msr("cmprsk.brier", cause = 1, time = 100))

# BS(t = 100), 2nd cause
predictions$score(msr("cmprsk.brier", cause = 2, time = 100))
```

`mlr_measures_cmprsk.auc`

Blanche's AUC Competing Risks Measure

Description

Calculates the time-dependent ROC-AUC at a **specific time point**, as described in Blanche et al. (2013).

Details

By default, this measure returns a **cause-independent AUC(t)** score, calculated as a weighted average of the cause-specific AUCs. The weights correspond to the relative event frequencies of each cause, following Equation (7) in Heyard et al. (2020). User-supplied weights are also supported. Alternatively, users can obtain the **cause-specific AUC(t)** for any individual cause by specifying the `cause` parameter.

Calls `riskRegression::Score()` with:

- `metric = "auc"`
- `cens.method = "ipcw"`
- `cens.model = "km"`

Notes on the `riskRegression` implementation:

1. IPCW weights are estimated using the **test data only**, so smaller test sets may lead to less stable estimates.
2. No extrapolation is supported: if `time` exceeds the maximum observed time on the test data, an error is thrown.
3. The choice of `time` is critical: if, at that time, no events of a given cause have occurred and all predicted CIFs are zero, `riskRegression` will return `NaN` for that cause-specific AUC (and subsequently for the summary AUC).

Dictionary

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

```
mlr_measures$get("cmprsk.auc")
msr("cmprsk.auc")
```

Meta Information

- Task type: "cmprsk"
- Range: $[0, 1]$
- Minimize: FALSE
- Average: macro
- Required Prediction: "cif"
- Required Packages: **mlr3**, **mlr3cmprsk**, **riskRegression**

Parameters

Id	Type	Default	Range
cause	integer	-	$[1, \infty)$
cause_weights	untyped	NULL	-
time	numeric	NULL	$[0, \infty)$

Parameter details

- `cause` (`numeric(1)` | `"mean"`)
  Integer number indicating which cause to use. Default value is `"mean"` which returns an event-frequency weighted mean of the cause-specific AUCs.
- `cause_weights` (`numeric()` | `NULL`)
  Optional custom weights for `cause = "mean"`. If `NULL`, observed cause frequencies in the test data are used. The weights must be non-negative, sum to 1 and match the number of causes 1-1, i.e. first weight for first cause, second weight for second cause, etc.
- `time` (`numeric(1)`)
  Single time point at which to return the score. If `NULL`, the **median observed time point** from the test set is used.

Super classes

`mlr3::Measure` -> `mlr3cmprsk::MeasureCompRisks` -> `MeasureCompRisksAUC`

Methods

Public methods:

- `MeasureCompRisksAUC$new()`
- `MeasureCompRisksAUC$clone()`

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

Usage:

```
MeasureCompRisksAUC$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Blanche, Paul, Dartigues, Francois J, Jacqmin-Gadda, Helene (2013). "Estimating and comparing time-dependent areas under receiver operating characteristic curves for censored event times with competing risks." *Statistics in Medicine*, **32**(30), 5381–5397. ISSN 1097-0258, doi:10.1002/SIM.5958, https://onlinelibrary.wiley.com/doi/10.1002/sim.5958.

Spitoni, Claudia, Lammens, Valerie, Putter, Hein (2018). "Prediction errors for state occupation and transition probabilities in multi-state models." *Biometrical Journal*, **60**(1), 34–48. ISSN 0323-3847, doi:10.1002/BIMJ.201600191, https://doi.org/10.1002/BIMJ.201600191.

Heyard, Rachel, Timsit, Jean-Francois, Held, Leonhard (2020). "Validation of discrete time-to-event prediction models in the presence of competing risks." *Biometrical Journal*, **62**(3), 643–657. https://doi.org/10.1002/BIMJ.201800293.

Examples

```
# Define the Learner
learner = lrn("cmprsk.fg")
learner

# Define a Task
task = tsk("pbc")

# Subset task features as Fine-Gray model doesn't accept factors
# Encode factors with `mlr3pipelines::po("encode")` if needed
feats = c("age", "chol", "albumin", "ast", "bili", "protime")
task$select(feats)

# Stratification based on event
task$set_col_roles(cols = "status", add_to = "stratum")

# Create train and test set
part = partition(task)

# Train the learner on the training set
learner$train(task, row_ids = part$train)
learner$native_model

# Make predictions for the test set
predictions = learner$predict(task, row_ids = part$test)
predictions

# Score the predictions
# AUC(t = 100), weighted mean score across causes (default)
predictions$score(msr("cmprsk.auc", cause = "mean", time = 100))
```

```
# AUC(t = 100), with user-specified weights
predictions$score(msr("cmprsk.auc", cause = "mean", cause_weights = c(0.2, 0.8),
  time = 100))

# AUC(t = 100), 1st cause
predictions$score(msr("cmprsk.auc", cause = 1, time = 100))

# AUC(t = 100), 2nd cause
predictions$score(msr("cmprsk.auc", cause = 2, time = 100))

# Prediction error (Brier score) at specific time point
# BS(t = 100) => weighted mean score across causes (default)
predictions$score(msr("cmprsk.brier", time = 100))

# BS(t = 100), 1st cause
predictions$score(msr("cmprsk.brier", cause = 1, time = 100))

# BS(t = 100), 2nd cause
predictions$score(msr("cmprsk.brier", cause = 2, time = 100))
```

`mlr_measures_cmprsk.brier`

Brier Score Competing Risks Measure

Description

Calculates the competing risks prediction error (Brier score, BS) at a **specific time point**, using IPCW as described in Schopp et al. (2011).

Details

By default, this measure returns a **cause-independent BS(t)** score, calculated as a weighted average of the cause-specific Brier scores. The weights correspond to the relative event frequencies of each cause, following Equation (8) in Spitoni et al. (2018). User-supplied weights are also supported. Alternatively, users can obtain the **cause-specific** Brier score for any individual cause by specifying the `cause` parameter.

Calls `riskRegression::Score()` with:

- `metric = "brier"`
- `cens.method = "ipcw"`
- `cens.model = "km"`

Notes on the `riskRegression` implementation:

1. IPCW weights are estimated using the **test data only**, so smaller test sets may lead to less stable estimates.
2. No extrapolation is supported: if `time` exceeds the maximum observed time on the test data, an error is thrown.

Dictionary

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

```
mlr_measures$get("cmprsk.brier")
msr("cmprsk.brier")
```

Meta Information

- Task type: "cmprsk"
- Range: $[0, \infty)$
- Minimize: TRUE
- Average: macro
- Required Prediction: "cif"
- Required Packages: **mlr3**, **mlr3cmprsk**, **riskRegression**

Parameters

Id	Type	Default	Range
cause	integer	-	$[1, \infty)$
cause_weights	untyped	NULL	-
time	numeric	NULL	$[0, \infty)$

Parameter details

- `cause` (`numeric(1)` | `"mean"`)
  Integer number indicating which cause to use. Default value is `"mean"` which returns an event-frequency weighted mean of the cause-specific Brier scores.
- `cause_weights` (`numeric()` | `NULL`)
  Optional custom weights for `cause = "mean"`. If `NULL`, observed cause frequencies in the test data are used. The weights must be non-negative, sum to 1 and match the number of causes 1-1, i.e. first weight for first cause, second weight for second cause, etc. See Spitoni et al. (2018), Equation (8) for a similar weighting scheme.
- `time` (`numeric(1)`)
  Single time point at which to return the score. If `NULL`, the **median observed time point** from the test set is used.

Super classes

`mlr3::Measure` -> `mlr3cmprsk::MeasureCompRisks` -> `MeasureCompRisksBrierScore`

Methods

Public methods:

- `MeasureCompRisksBrierScore$new()`
- `MeasureCompRisksBrierScore$clone()`

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

Usage:

```
MeasureCompRisksBrierScore$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Schoop, Roland, Beyersmann, Jan, Schumacher, Martin, Binder, Harald (2011). "Quantifying the predictive accuracy of time-to-event models in the presence of competing risks." *Biometrical Journal*, **53**(1), 88–112. https://doi.org/10.1002/BIMJ.201000073.

Spitoni, Claudia, Lammens, Valerie, Putter, Hein (2018). "Prediction errors for state occupation and transition probabilities in multi-state models." *Biometrical Journal*, **60**(1), 34–48. ISSN 0323-3847, doi:10.1002/BIMJ.201600191, https://doi.org/10.1002/BIMJ.201600191.

Examples

```
# Define the Learner
learner = lrn("cmprsk.fg")
learner

# Define a Task
task = tsk("pbc")

# Subset task features as Fine-Gray model doesn't accept factors
# Encode factors with `mlr3pipelines::po("encode")` if needed
feats = c("age", "chol", "albumin", "ast", "bili", "protime")
task$select(feats)

# Stratification based on event
task$set_col_roles(cols = "status", add_to = "stratum")

# Create train and test set
part = partition(task)

# Train the learner on the training set
learner$train(task, row_ids = part$train)
learner$native_model
```

```
# Make predictions for the test set
predictions = learner$predict(task, row_ids = part$test)
predictions

# Score the predictions
# AUC(t = 100), weighted mean score across causes (default)
predictions$score(msr("cmprsk.auc", cause = "mean", time = 100))

# AUC(t = 100), with user-specified weights
predictions$score(msr("cmprsk.auc", cause = "mean", cause_weights = c(0.2, 0.8),
  time = 100))

# AUC(t = 100), 1st cause
predictions$score(msr("cmprsk.auc", cause = 1, time = 100))

# AUC(t = 100), 2nd cause
predictions$score(msr("cmprsk.auc", cause = 2, time = 100))

# Prediction error (Brier score) at specific time point
# BS(t = 100) => weighted mean score across causes (default)
predictions$score(msr("cmprsk.brier", time = 100))

# BS(t = 100), 1st cause
predictions$score(msr("cmprsk.brier", cause = 1, time = 100))

# BS(t = 100), 2nd cause
predictions$score(msr("cmprsk.brier", cause = 2, time = 100))
```

`mlr_tasks_pbc` *Primary Biliary Cholangitis Competing Risks Task*

Description

A competing risks task for the pbc data set.

Format

R6::R6Class inheriting from TaskCompRisks.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function tsk():

```
mlr_tasks$get("pbc")
tsk("pbc")
```

Meta Information

- Task type: “cmprsk”
- Dimensions: 276x19
- Properties: -
- Has Missings: FALSE
- Target: “time”, “status”
- Features: “age”, “albumin”, “alk.phos”, “ascites”, “ast”, “bili”, “chol”, “copper”, “edema”, “hepato”, “platelet”, “protime”, “sex”, “spiders”, “stage”, “trig”, “trt”

Pre-processing

- Removed column id.
- Kept only complete cases (no missing values).
- Column age has been converted to integer.
- Columns trt, stage, hepato, edema and ascites have been converted to factors.
- Column trt has levels Dpenicillmain and placebo instead of 1 and 2.
- Column status has 0 for censored, 1 for transplant and 2 for death.
- Column time as been converted from days to months.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Dictionary of Tasks: mlr3::mlr_tasks
- as.data.table(mlr_tasks) for a table of available Tasks in the running session

Other Task: TaskCompRisks

PredictionCompRisks *Prediction Object for Competing Risks*

Description

This object stores the predictions returned by a learner of class LearnerCompRisks.

The task_type is set to "cmprsk".

For accessing survival and hazard functions, as well as other complex methods from a LearnerCompRisks object is not possible atm.

Super class

mlr3::Prediction -> PredictionCompRisks

Active bindings

`truth (Surv)`
: True (observed) outcome.

`cif (list())`
: Access the stored CIFs.

Methods

Public methods:

- `PredictionCompRisks$new()`
- `PredictionCompRisks$clone()`

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

Usage:

```
PredictionCompRisks$new(
  task = NULL,
  row_ids = task$row_ids,
  truth = task$truth(),
  cif = NULL,
  check = TRUE
)
```

Arguments:

`task` (TaskCompRisks)
: Task, used to extract defaults for `row_ids` and `truth`.

`row_ids (integer())`
: Row ids of the predicted observations, i.e. the row ids of the test set.

`truth (survival::Surv())`
: True (observed) response.

`cif (list())`
: A `list` of two or more `matrix` objects. Each matrix represents a different competing event and it stores the **Cumulative Incidence function** for each test observation. In each matrix, rows represent observations and columns time points. The names of the `list` must correspond to the competing event names (`task$cmp_events`).

`check (logical(1))`
: If `TRUE`, performs argument checks and predict type conversions.

Details: The `cif` input currently is a list of CIF matrices.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3)
task = tsk("pbc")
learner = lrn("cmprsk.aalen")
part = partition(task)
p = learner$train(task, part$train)$predict(task, part$test)
p

# CIF list: 1 matrix (obs x times) per competing event
names(p$cif) # competing events
# CIF matrix for competing event 1 (first 5 test observations and 20 time points)
p$cif[["1"]][1:5, 1:20]
# CIF matrix for competing event 2 (first 5 test observations and 20 time points)
p$cif[["2"]][1:5, 1:20]

# data.table conversion
tab = as.data.table(p)
tab$CIF[[1]] # for first test observation, list of CIF vectors
```

`TaskCompRisks` *Competing Risks Task*

Description

This task extends mlr3::Task and mlr3::TaskSupervised to handle survival problems with **competing risks**. The target variable consists of survival times and an event indicator, which must be a non-negative integer in the set $(0, 1, 2, ..., K)$. 0 represents censored observations, while other integers correspond to distinct competing events. Every row corresponds to one subject/observation.

Predefined tasks are stored in mlr3::mlr_tasks.

The `task_type` is set to `"cmprsk"`.

Note: Currently only right-censoring is supported.

Super classes

`mlr3::Task` -> `mlr3::TaskSupervised` -> `TaskCompRisks`

Active bindings

`cens_type (character(1))`
: Returns the type of censoring.

  Currently, only the `"right"` censoring type is fully supported. The API might change in the future to support left and interval censoring.

`cmp_events (character(1))`
: Returns the names of the competing events.

Methods

Public methods:

- TaskCompRisks$new()
- TaskCompRisks$truth()
- TaskCompRisks$formula()
- TaskCompRisks$times()
- TaskCompRisks$event()
- TaskCompRisks$unique_events()
- TaskCompRisks$unique_times()
- TaskCompRisks$unique_event_times()
- TaskCompRisks$aalen_johansen()
- TaskCompRisks$cens_prop()
- TaskCompRisks$filter()
- TaskCompRisks$clone()

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

Usage:

```
TaskCompRisks$new(
  id,
  backend,
  time = "time",
  event = "event",
  label = NA_character_
)
```

Arguments:

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

`backend` (mlr3::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 outcome time.

`event` (`character(1)`)
: Name of column giving that holds the event indicator. 0 corresponds to censoring, values > 0 correspond to different competing events.

`label` (`character(1)`)
: Label for the new instance.

Details: Only right-censoring competing risk tasks are currently supported.

Method `truth()`**:** True response for specified `row_ids`. This is the multi-state format using Surv with the `event` target column as a `factor`: `Surv(time, as.factor(event))`
Defaults to all rows with role `"use"`.

Usage:

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

Arguments:

`rows (integer())`
Row indices.

Returns: `survival::Surv()`.

Method `formula()`**:** Creates a formula for competing risk models with `survival::Surv()` on the LHS (left hand side).

Usage:

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

Arguments:

`rhs` If `NULL`, RHS (right hand side) is `"."`, otherwise RHS is `"rhs"`.

Returns: `stats::formula()`.

Method `times()`**:** Returns the (unsorted) outcome times.

Usage:

```
TaskCompRisks$times(rows = NULL)
```

Arguments:

`rows (integer())`
Row indices.

Returns: `numeric()`

Method `event()`**:** Returns the event indicator.

Usage:

```
TaskCompRisks$event(rows = NULL)
```

Arguments:

`rows (integer())`
Row indices.

Returns: `integer()`

Method `unique_events()`**:** Returns the unique events (excluding censoring).

Usage:

```
TaskCompRisks$unique_events(rows = NULL)
```

Arguments:

`rows (integer())`
Row indices.

Returns: `integer()`

Method `unique_times()`**:** Returns the sorted unique outcome times.

Usage:

```
TaskCompRisks$unique_times(rows = NULL)
```

Arguments:

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

Returns: `numeric()`

Method `unique_event_times()`: Returns the sorted unique event outcome times (by any cause).

Usage:

```
TaskCompRisks$unique_event_times(rows = NULL)
```

Arguments:

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

Returns: `numeric()`

Method `aalen_johansen()`: Calls `survival::survfit()` to calculate the Aalen–Johansen estimator.

Usage:

```
TaskCompRisks$aalen_johansen(strata = NULL, rows = NULL, ...)
```

Arguments:

`strata` (`character()`)
Stratification variables to use.

`rows` (`integer()`)
Subset of row indices.

`...` (any)
Additional arguments passed down to `survival::survfit.formula()`.

Returns: survival::survfit.object.

Method `cens_prop()`: Returns the **proportion of censoring** for this competing risks task. By default, this is returned for all observations, otherwise only the specified ones (`rows`).

Usage:

```
TaskCompRisks$cens_prop(rows = NULL)
```

Arguments:

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

Returns: `numeric()`

Method `filter()`: Subsets the task, keeping only the rows specified via row ids `rows`. This operation mutates the task in-place.

Usage:

```
TaskCompRisks$filter(rows = NULL)
```

Arguments:

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

Returns: Returns the object itself, but modified **by reference.**

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

Other Task: `mlr_tasks_pbc`

Examples

```
library(mlr3)
task = tsk("pbc")

# meta data
task$target_names # target is always (time, status) for right-censoring tasks
task$feature_names
task$formula()

# survival data
task$truth() # survival::Surv() object
task$times() # (unsorted) times
task$event() # event indicators (0 = censored, >0 = different causes)
task$unique_times() # sorted unique times
task$unique_event_times() # sorted unique event times (from any cause)
task$aalen_johansen(strata = "sex") # Aalen-Johansen estimator

# proportion of censored observations across all dataset
task$cens_prop()
```

Index