# Package: discSurv (via r-universe)

September 12, 2024

**Version** 2.0.0

**Title** Discrete Time Survival Analysis

**Date** 2022-03-02

**Author** Thomas Welchowski <welchow@imbie.meb.uni-bonn.de> and Moritz Berger <moritz.berger@imbie.uni-bonn.de> and David Koehler <koehler@imbie.uni-bonn.de> and Matthias Schmid <matthias.schmid@imbie.uni-bonn.de>

**Maintainer** Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

**Description** Provides data transformations, estimation utilities, predictive evaluation measures and simulation functions for discrete time survival analysis.

**Depends** R (>= 3.5.0), treeClust

**Imports** functional, mvtnorm, mgcv, data.table, Rdpack, VGAM, geepack, rpart, ranger, mvnfast

**RdMacros** Rdpack

**Suggests** Matrix, matrixcalc, numDeriv, caret, Ecdat, pec, survival, nnet

**Encoding** UTF-8

**License** GPL-3

**LazyLoad** yes

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2022-03-02 15:40:02 UTC

## Contents

---

`discSurv-package` *Discrete Survival Analysis*

---

## Description

Includes functions for data transformations, estimation, evaluation and simulation of discrete survival analysis. The most important functions are listed below:

- `contToDisc`: Discretizes continuous time variable into a specified grid of censored data for discrete survival analysis.

- dataLong: Transform data from short format into long format for discrete survival analysis and right censoring.
- dataLongCompRisks: Transforms short data format to long format for discrete survival modelling in the case of competing risks with right censoring.
- dataLongTimeDep: Transforms short data format to long format for discrete survival modelling of single event analysis with right censoring.
- cIndex: Calculates the concordance index for discrete survival models (independent measure of time).
- dataLongSubDist: Converts the data to long format suitable for applying discrete subdistribution hazard modelling (competing risks).

## Details

"DataShort" format is defined as data without repeated measurements. "DataSemiLong" format consists of repeated measurements, but there are gaps between the discrete time intervals. "DataLong" format is expanded to include all time intervals up to the last observation per individual.

| | |
|---|---|
| Package: | discSurv |
| Type: | Package |
| Version: | 2.0.0 |
| Date: | 2022-03-02 |
| License: | GPL-3 |

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

Moritz Berger <moritz.berger@imbie.uni-bonn.de>

David Koehler <koehler@imbie.uni-bonn.de>

Matthias Schmid <matthias.schmid@imbie.uni-bonn.de>

## References

Berger M, Schmid M (2018). "Semiparametric regression for discrete time-to-event data." *Statistical Modelling*, **18**, 322–345.

Berger M, Welchowski T, Schmitz-Valckenberg S, Schmid M (2019). "A classification tree approach for the modeling of competing risks in discrete time." *Advances in Data Analysis and Classification*, **13**, 965-990.

Berger M, Schmid M, Welchowski T, Schmitz-Valckenberg S, Beyersmann J (2020). "Subdistribution Hazard Models for Competing Risks in Discrete Time." *Biostatistics*, **21**, 449-466.

Schmid M, Tutz G, Welchowski T (2018). "Discrimination Measures for Discrete Time-to-Event Predictions." *Econometrics and Statistics*, **7**, 153-164.

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

adjDevResid *Adjusted Deviance Residuals in short format*

## Description

Calculates the adjusted deviance residuals for arbitrary prediction models. The adjusted deviance residuals should be approximately normal distributed, in the case of a well fitting model.

## Usage

```
adjDevResid(dataLong, hazards)
```

## Arguments

| | |
|---|---|
| dataLong | Data set in long format ("class data.frame"). |
| hazards | Estimated discrete hazards of the data in long format("numeric vector"). Hazard rates are probabilities and therefore restricted to the interval [0, 1]. |

## Value

- Output List with objects:
  - AdjDevResid Adjusted deviance residuals as numeric vector
- Input A list of given argument input values (saved for reference)

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

Tutz G (2012). *Regression for Categorical Data*. Cambridge University Press.

## See Also

intPredErr, predErrCurve

## Examples

```
library(survival)

# Transform data to long format
heart[, "stop"] <- ceiling(heart[, "stop"])
set.seed(0)
Indizes <- sample(unique(heart$id), 25)
randSample <- heart[unlist(sapply(1:length(Indizes),
function(x) which(heart$id == Indizes[x]))),]
```

```
heartLong <- dataLongTimeDep(dataSemiLong = randSample,
timeColumn = "stop", eventColumn = "event", idColumn = "id", timeAsFactor = FALSE)

# Fit a generalized, additive model and predict discrete hazards on data in long format
library(mgcv)
gamFit <- gam(y ~ timeInt + surgery + transplant + s(age), data = heartLong, family = "binomial")
hazPreds <- predict(gamFit, type = "response")

# Calculate adjusted deviance residuals
devResiduals <- adjDevResid(dataLong = heartLong, hazards = hazPreds)$Output$AdjDevResid
devResiduals
```

---

`calibrationPlot` *Calibration Plots*

---

## Description

Calibration plot based on predictions. Overall root mean squared error (RMSE) of predicted and observed discrete hazards is calculated.

## Usage

```
calibrationPlot(
  testPreds,
  testDataLong,
  weights = NULL,
  K = 10,
  event = "e1",
  ...
)
```

## Arguments

| | |
|---|---|
| `testPreds` | Predictions on the validation data with model fitted on training data ("numeric vector"). |
| `testDataLong` | Validation data set in long format ("class data.frame"). |
| `weights` | optional vector of weights ("numeric vector"). The length of weights must be equal to the number of observations of the validation data set. |
| `K` | Number of subsets for plotting ("integer vector"). |
| `event` | Column names of the event to be considered for plotting (only in case of cause-specific hazards) ("character vector"). |
| `...` | Additional arguments passed to `plot`. |

## Value

Calibration plot

## Author(s)

Moritz Berger <moritz.berger@imbie.uni-bonn.de>
https://www.imbie.uni-bonn.de/personen/dr-moritz-berger/

## References

Berger M, Schmid M (2018). "Semiparametric regression for discrete time-to-event data." *Statistical Modelling*, **18**, 322–345.

Heyard R, Timsit J, Held L, COMBACTE-MAGNET,consortium (2019). "Validation of discrete time-to-event prediction models in the presence of competing risks." *Biometrical Journal*, **62**, 643-657.

Berger M, Schmid M (2020). "Assessing the calibration of subdistribution hazard models in discrete time." *arXiv:2001.11240*.

## See Also

estRecal, dataLong, dataLongCompRisks, dataLongSubDist

## Examples

```
####################
# Data preprocessing

# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
selectInd1 <- 1:100
selectInd2 <- 101:200
trainSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd1], ]
valSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd2], ]

####################
# One event

# Convert to long format
trainSet_long <- dataLong(dataShort = trainSet, timeColumn = "spell", eventColumn = "censor1")
valSet_long <- dataLong(dataShort = valSet, timeColumn = "spell", eventColumn = "censor1")

# Estimate continuation ratio model with logit link
glmFit <- glm(formula = y ~ timeInt + age + logwage, data = trainSet_long, family = binomial())

# Calculate predicted hazards
predHazards <- predict(glmFit, newdata = valSet_long, type = "response")

# Calibration plot
calibrationPlot(predHazards, testDataLong = valSet_long)
```

```
############################
# Two cause specific hazards

# Convert to long format
trainSet_long <- dataLongCompRisks(dataShort = trainSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"))
valSet_long <- dataLongCompRisks(dataShort = valSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"))

# Estimate continuation ratio model with logit link
vglmFit <- VGAM::vglm(formula = cbind(e0, e1, e2) ~ timeInt + age + logwage, data = trainSet_long,
family = VGAM::multinomial(refLevel = "e0"))

# Calculate predicted hazards
predHazards <- VGAM::predictvglm(vglmFit, newdata = valSet_long, type = "response")

# Calibration plots
calibrationPlot(predHazards, testDataLong = valSet_long)
calibrationPlot(predHazards, testDataLong = valSet_long, event = "e2")

##############################
# Subdistribution hazards model

# Convert to long format
trainSet_long <- dataLongSubDist(dataShort = trainSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"), eventFocus = "censor1")
valSet_long <- dataLongSubDist(dataShort = valSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"), eventFocus = "censor1")

# Estimate continuation ratio model with logit link
glmFit <- glm(formula = y ~ timeInt + age + logwage, data = trainSet_long,
family = binomial(), weights = trainSet_long$subDistWeights)

# Calculate predicted  hazards
predHazards <- predict(glmFit, newdata = valSet_long, type = "response")

# Calibration plot
calibrationPlot(predHazards, testDataLong = valSet_long, weights = valSet_long$subDistWeights)
```

---

cIndex *Concordance index*

---

## Description

Calculates the concordance index for discrete survival models, which does not depend on time. This is the probability that, for a pair of randomly chosen comparable samples, the sample with the higher risk prediction will experience an event before the other sample or belongs to a higher binary class.

## Usage

```
cIndex(marker, testTime, testEvent, trainTime, trainEvent)
```

## Arguments

| | |
|---|---|
| marker | Gives the predicted values of the linear predictor of a regression model ("numeric vector"). May also be on the response scale. |
| testTime | New time intervals in the test data ("integer vector"). |
| testEvent | Event indicators in the test data ("binary vector"). |
| trainTime | Time intervals in the training data ("integer vector"). |
| trainEvent | Event indicators in the training data ("binary vector"). |

## Value

Value of discrete concordance index between zero and one ("numeric vector").

## Note

It is assumed that all time points up to the last observed interval [a_q-1, a_q) are available.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Schmid M, Tutz G, Welchowski T (2018). "Discrimination Measures for Discrete Time-to-Event Predictions." *Econometrics and Statistics*, **7**, 153-164.

Uno H, Cai T, Tian L, Wei LJ (2012). "Evaluating Prediction Rules fort-Year Survivors With Censored Regression Models." *Journal of the American Statistical Association*, **102**, 527-537.

Heagerty PJ, Zheng Y (2005). "Survival Model Predictive Accuracy and ROC Curves." *Biometrics*, **61**, 92-105.

## Examples

```
################################################
# Example with unemployment data and prior fitting

library(Ecdat)
library(caret)
library(mgcv)
data(UnempDur)
summary(UnempDur$spell)
# Extract subset of data
set.seed(635)
IDsample <- sample(1:dim(UnempDur)[1], 100)
UnempDurSubset <- UnempDur [IDsample, ]
```

```
set.seed(-570)
TrainingSample <- sample(1:100, 75)
UnempDurSubsetTrain <- UnempDurSubset [TrainingSample, ]
UnempDurSubsetTest <- UnempDurSubset [-TrainingSample, ]

# Convert to long format
UnempDurSubsetTrainLong <- dataLong(dataShort = UnempDurSubsetTrain,
timeColumn = "spell", eventColumn = "censor1")

# Estimate gam with smooth baseline
gamFit <- gam(formula = y ~ s(I(as.numeric(as.character(timeInt)))) +
s(age) + s(logwage), data = UnempDurSubsetTrainLong, family = binomial())
gamFitPreds <- predict(gamFit, newdata = cbind(UnempDurSubsetTest,
timeInt = UnempDurSubsetTest$spell))

# Evaluate C-Index based on short data format
cIndex(marker = gamFitPreds,
testTime = UnempDurSubsetTest$spell,
testEvent = UnempDurSubsetTest$censor1,
trainTime = UnempDurSubsetTrain$spell,
trainEvent = UnempDurSubsetTrain$censor1)

####################################
# Example National Wilm's Tumor Study

library(survival)
head(nwtco)
summary(nwtco$rel)

# Select subset
set.seed(-375)
Indices <- sample(1:dim(nwtco)[1], 500)
nwtcoSub <- nwtco [Indices, ]

# Convert time range to 30 intervals
intLim <- quantile(nwtcoSub$edrel, prob = seq(0, 1, length.out = 30))
intLim [length(intLim)] <- intLim [length(intLim)] + 1
nwtcoSubTemp <- contToDisc(dataShort = nwtcoSub, timeColumn = "edrel", intervalLimits = intLim)
nwtcoSubTemp$instit <- factor(nwtcoSubTemp$instit)
nwtcoSubTemp$histol <- factor(nwtcoSubTemp$histol)
nwtcoSubTemp$stage <- factor(nwtcoSubTemp$stage)

# Split in training and test sample
set.seed(-570)
TrainingSample <- sample(1:dim(nwtcoSubTemp)[1], round(dim(nwtcoSubTemp)[1]*0.75))
nwtcoSubTempTrain <- nwtcoSubTemp [TrainingSample, ]
nwtcoSubTempTest <- nwtcoSubTemp [-TrainingSample, ]

# Convert to long format
nwtcoSubTempTrainLong <- dataLong(dataShort = nwtcoSubTempTrain,
timeColumn = "timeDisc", eventColumn = "rel", timeAsFactor=TRUE)

# Estimate glm
```

```
inputFormula <- y ~ timeInt + histol + instit + stage
glmFit <- glm(formula = inputFormula, data = nwtcoSubTempTrainLong, family = binomial())
linPreds <- predict(glmFit, newdata = cbind(nwtcoSubTempTest,
timeInt = factor(nwtcoSubTempTest$timeDisc, levels=levels(nwtcoSubTempTrainLong$timeInt))))

# Evaluate C-Index based on short data format
cIndex(marker = linPreds,
testTime = as.numeric(as.character(nwtcoSubTempTest$timeDisc)),
testEvent = nwtcoSubTempTest$rel,
trainTime = as.numeric(as.character(nwtcoSubTempTrain$timeDisc)),
trainEvent = nwtcoSubTempTrain$rel)
```

---

`cIndexCompRisks` *Discrete concordance index for competing risks*

---

## Description

Estimates the discrete concordance index in the case of competing risks.

## Usage

```
cIndexCompRisks(markers, testTime, testEvents, trainTime, trainEvents)
```

## Arguments

| | |
|---|---|
| `markers` | Predictions on the test data with model fitted on training data ("numeric matrix"). Predictions are stored in the rows and the number of columns equal to the number of events. |
| `testTime` | New time intervals in the test data ("integer vector"). |
| `testEvents` | New event indicators (0 or 1) in the test data ("binary matrix"). Number of columns are equal to the number of events. |
| `trainTime` | Time intervals in the training data ("integer vector"). |
| `trainEvents` | Event indicators (0 or 1) in the training data ("binary matrix"). Number of columns are equal to the number of events. |

## Value

Value of discrete concordance index between zero and one ("numeric vector").

## Note

It is assumed that all time points up to the last observed interval [a_q-1, a_q) are available.

## Author(s)

Moritz Berger <moritz.berger@imbie.uni-bonn.de>
https://www.imbie.uni-bonn.de/personen/dr-moritz-berger/

## References

Heyard R, Timsit J, Held L, COMBACTE-MAGNET,consortium (2019). "Validation of discrete time-to-event prediction models in the presence of competing risks." *Biometrical Journal*, **62**, 643-657.

## See Also

cIndex

## Examples

```
#################################################
# Example with unemployment data and prior fitting

library(Ecdat)
data(UnempDur)
summary(UnempDur$spell)
# Extract subset of data
set.seed(635)
IDsample <- sample(1:dim(UnempDur)[1], 100)
UnempDurSubset <- UnempDur [IDsample, ]
set.seed(-570)
TrainingSample <- sample(1:100, 75)
UnempDurSubsetTrain <- UnempDurSubset [TrainingSample, ]
UnempDurSubsetTest <- UnempDurSubset [-TrainingSample, ]

# Convert to long format
UnempDurSubsetTrainLong <- dataLongCompRisks(dataShort = UnempDurSubsetTrain, timeColumn = "spell",
eventColumns = c("censor1", "censor4"), timeAsFactor = TRUE)

# Estimate continuation ratio model with logit link
vglmFit <- VGAM::vglm(formula = cbind(e0, e1, e2) ~ timeInt + age + logwage,
data = UnempDurSubsetTrainLong, family=VGAM::multinomial(refLevel = "e0"))

gamFitPreds <- VGAM::predictvglm(vglmFit , newdata = cbind(UnempDurSubsetTest,
timeInt = as.factor(UnempDurSubsetTest$spell)))

# Evaluate C-Index based on short data format
cIndexCompRisks(markers = gamFitPreds,
testTime = UnempDurSubsetTest$spell,
testEvents = UnempDurSubsetTest[, c("censor1", "censor4")],
trainTime = UnempDurSubsetTrain$spell,
trainEvents = UnempDurSubsetTrain[, c("censor1", "censor4")])
```

---

compRisksGEE *GEE model for discrete competing risks*

---

## Description

Estimates generalized estimation equation model for each competing event separately. Dependence within person IDs is accounted for by assuming a working covariance structure.

## Usage

```
compRisksGEE(
  datShort,
  dataTransform = "dataLongCompRisks",
  corstr = "independence",
  formulaVariable = ~timeInt,
  ...
)

## S3 method for class 'dCRGEE'
predict(object, newdata, ...)
```

## Arguments

| | |
|---|---|
| datShort | Original data set in short format with each row corresponding to one independent observation("class data.frame"). |
| dataTransform | Specification of the data transformation function from short to long format("character vector"). There are two available options: Without time dependent covariates ("dataLongCompRisks") and with time dependent covariates ("dataLongCompRisksTimeDep"). The default is set to the former. |
| corstr | Assumption of correlation structure ("character vector"). The following are permitted: '"independence"', '"exchangeable"', '"ar1"', '"unstructured"' and '"userdefined". |
| formulaVariable | Specifies the right hand side of the regression formula ("class formula"). The default is to use the discrete time variable, which corresponds to a covariate free hazard. It is recommended to always include the discrete time variable "timeInt". |
| ... | Additional arguments to data transformation (compRisksGEE) or prediction function (predict). Preprocessing function argument responseAsFactor has to be set to FALSE (Default). |
| object | Discrete time competing risks GEE model prediction model ("class dCRGEE"). |
| newdata | ("class data.set") New data set to be used for prediction (class data.frame). |

## Details

Variables in argument *formulaVariable* need to be separated by "+ ". For example if the two variables *timeInt* and *X1* should be included the formula would be "~ timeInt + X1". The variable *timeInt* is constructed before estimation of the model.

## Value

Returns an object of class "geeglm".

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Lee M, Feuer EJ, Fine JP (2018). "On the analysis of discrete time competing risks data." *Biometrics*, **74**, 1468-1481.

## See Also

covarGEE, dataLongCompRisks, dataLongCompRisksTimeDep, geeglm

## Examples

```
# Example with unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
SubUnempDur <- UnempDur [1:100, ]

# Estimate GEE models for all events
estGEE <- compRisksGEE(datShort = SubUnempDur, dataTransform = "dataLongCompRisks",
corstr = "independence", formulaVariable =~ timeInt + age + ui + logwage * ui,
eventColumns = c("censor1", "censor2", "censor3", "censor4"), timeColumn = "spell")
names(estGEE)
estGEE[[1]]

# Predictions
SubUnempDurLong <- dataLongCompRisks(dataShort = SubUnempDur,
eventColumns = c("censor1", "censor2", "censor3", "censor4"), timeColumn = "spell")
preds <- predict(estGEE, newdata = SubUnempDurLong)
head(preds)
```

---

# contToDisc *Continuous to Discrete Transformation*

---

## Description

Discretizes continuous time variable into a specified grid of censored data for discrete survival analysis. It is a data preprocessing step, before the data can be extendend in long format and further analysed with discrete survival models.

## Usage

```
contToDisc(
  dataShort,
  timeColumn,
  intervalLimits,
  equi = FALSE,
  timeAsFactor = FALSE
)
```

## Arguments

| | |
|---|---|
| dataShort | Original data in short format ("class data.frame"). |
| timeColumn | Name of the column of discrete survival times ("character vector"). |
| intervalLimits | Right interval borders ("numeric vector"), e. g. if the intervals are [0, a_1), [a_1, a_2), [a_2, a_max), then intervalLimits = c(a_1, a_2, a_max) |
| equi | Specifies if argument *intervalLimits* should be interpreted as number of equidistant intervals ("logical vector"). |
| timeAsFactor | Specifies if the computed discrete time intervals should be converted to a categorical variable ("logical vector"). Default is FALSE. In the default settings the discret time intervals are treated as quantitative ("numeric vector"). |

## Value

Gives the data set expanded with a first column "timeDisc". This column includes the discrete time intervals ("class factor").

## Note

In discrete survival analysis the survival times have to be categorized in time intervals. Therefore this function is required, if there are observed continuous survival times.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

## See Also

dataLong, dataLongTimeDep, dataLongCompRisks

## Examples

```
# Example copenhagen stroke study data
library(pec)
data(cost)
head(cost)

# Convert observed times to months
# Right borders of intervals [0, a_1), [a_1, a_2), ... , [a_{\max-1}, a_{\max})
IntBorders <- 1:ceiling(max(cost$time)/30)*30

# Select subsample
subCost <- cost [1:100, ]
CostMonths <- contToDisc(dataShort=subCost, timeColumn = "time", intervalLimits = IntBorders)
head(CostMonths)

# Select subsample giving number of equidistant intervals
CostMonths <- contToDisc(dataShort = subCost, timeColumn = "time", intervalLimits = 10, equi = TRUE)
head(CostMonths)
```

---

# covarGEE *GEE covariance of all events for discrete competing risks*

---

## Description

Estimates covariance of estimated parameters of all competing events generalized estimation equation models using sandwich approach.

## Usage

```
covarGEE(modelEst)
```

## Arguments

| | |
|---|---|
| modelEst | Discrete time competing risks GEE model prediction model ("class dCRGEE"). |

## Value

Returns symmetric matrix of rows and columns dimension "number of competing risks" * "number of regression parameters" ("numeric matrix").

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Lee M, Feuer EJ, Fine JP (2018). "On the analysis of discrete time competing risks data." *Biometrics*, **74**, 1468-1481.

## See Also

compRisksGEE, dataLongCompRisks, dataLongCompRisksTimeDep, geeglm

## Examples

```
# Example with unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
SubUnempDur <- UnempDur [1:100, ]

# Estimate GEE models for all events
estGEE <- compRisksGEE(datShort = SubUnempDur, dataTransform = "dataLongCompRisks",
corstr = "independence", formulaVariable =~ timeInt + age + ui + logwage * ui,
eventColumns = c("censor1", "censor2", "censor3", "censor4"), timeColumn = "spell")

## Not run:
# Estimate covariance matrix of estimated parameters and competing events
estCovar <- covarGEE(modelEst=estGEE)
estCovar

# Covariances of estimated parameters of one event equal the diagonal blocks
lengthParameters <- length(estGEE[[1]]$coefficients)
noCompEvents <- length(estGEE)
meanAbsError <- rep(NA, noCompEvents)
for( k in 1:noCompEvents ){

  relInd <- (1 + (k-1) * lengthParameters) : (k * lengthParameters)
  meanAbsError[k] <- mean(abs(estCovar[relInd, relInd] - estGEE[[k]]$geese$vbeta))

}
mean(meanAbsError)
# -> Covariance estimates within each event are equal to diagonal blocks in
# complete covariance matrix with very small differences due to numerical accuracy.

## End(Not run)
```

---

# crash2 *Crash 2 competing risk data*

---

## Description

Adapted version of the crash2 trial data as availlable in the package Hmisc. Both death or survival and main cause of death are included. Death times are discretized into days. Included covariates are sex and age of patient, elapsed time between injury and hospitalization, type of injury, systolic blood pressure, heart rate, respiratory rate, central capillary refill time and total glascow coma score. #'

- Column "time" is time until death or hospital discharge/transfer in weeks.
- Column "Status" denotes type of death
  - bleeding
  - head injury
  - vascular occlusion
  - multi organ failure
  - other
  - NA if patient is not dead (see "statusSE")
- Column "statusSE" denotes death or discharge/transfer from hospital
  - 0 - Transfer/Discharge
  - 1 - Death
- Column "sex" denotes sex of the patient.
- Column "age" denotes age of the patient in years.
- Column "injurytime" gives time in hours between injury and hospitalization.
- Column "injurytype" denotes type of injury, one in
  - blunt
  - penetrating
  - blunt and penetrating
- Column "sbp" denotes systolic blood pressure in mmHg.
- Column "rr" denotes respiratory rate per minute.
- Column "cc" denotes central capillary refill time in seconds.
- Column "hr" denotes heart rate per minute.
- Column "gcs" denotes total Glascow Coma Score.

## Usage

```
data(crash2)
```

## Author(s)

David Koehler <koehler@imbie.uni-bonn.de>

## Source

getHdata

dataCensoring *Data Censoring Transformation for short formats*

## Description

Function for transformation of discrete survival times in censoring encoding. The original data is expanded to include the censoring process. Alternatively the long data format can also be augmented. With the new generated variable "yCens", the discrete censoring process can be analyzed instead of the discrete survival process. In discrete survival analysis this information is used to constructs weights for predictive evaluation measures. It is applicable in single event survival analysis.

## Usage

```
dataCensoring(dataShort, eventColumns, timeColumn, shortFormat = TRUE)
```

## Arguments

| | |
|---|---|
| dataShort | Original data set in short format ("class data.frame"). |
| eventColumns | Name of event columns ("character vector"). The event columns have to be in binary format. If the sum of all events equals zero in a row, then this observation is interpreted as censored. |
| timeColumn | Name of column with discrete time intervals ("character vector"). |
| shortFormat | Is the supplied data set *dataShort* not preprocessed with function dataLong() ("logical vector")? Default is TRUE. If shortFormat=FALSE then it is assumed that the data set was augmented with function dataLong(). |

## Value

Original data set as argument *dataShort*, but with added censoring process as first variable in column "yCens".

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

Fahrmeir L (2005). "Discrete Survival-Time Models." In *Encyclopedia of Biostatistics*, chapter Survival Analysis. John Wiley \& Sons.

Thompson Jr. WA (1977). "On the Treatment of Grouped Observations in Life Studies." *Biometrics*, **33**, 463-470.

## See Also

contToDisc, dataLong, dataLongTimeDep, dataLongCompRisks

## Examples

```
library(pec)
data(cost)
head(cost)
IntBorders <- 1:ceiling(max(cost$time)/30)*30
subCost <- cost [1:100, ]

# Convert from days to months
CostMonths <- contToDisc(dataShort=subCost, timeColumn="time", intervalLimits=IntBorders)
head(CostMonths)

# Generate censoring process variable in short format
CostMonthsCensorShort <- dataCensoring (dataShort = CostMonths,
eventColumns = "status", timeColumn = "time", shortFormat = TRUE)
head(CostMonthsCensorShort)

################################
# Example with long data format
library(pec)
data(cost)
head(cost)
IntBorders <- 1:ceiling(max(cost$time)/30)*30
subCost <- cost [1:100, ]

# Convert from days to months
CostMonths <- contToDisc(dataShort = subCost, timeColumn = "time", intervalLimits = IntBorders)
head(CostMonths)

# Convert to long format based on months
CostMonthsLong <- dataLong(dataShort = CostMonths, timeColumn = "timeDisc", eventColumn = "status")
head(CostMonthsLong, 20)

# Generate censoring process variable
CostMonthsCensor <- dataCensoring (dataShort = CostMonthsLong, timeColumn = "timeInt",
shortFormat = FALSE)
head(CostMonthsCensor)
tail(CostMonthsCensor [CostMonthsCensor$obj==1, ], 10)
tail(CostMonthsCensor [CostMonthsCensor$obj==3, ], 10)
```

---

`dataLong` *Data Long Transformation*

---

## Description

Transform data from short format into long format for discrete survival analysis and right censoring. Data is assumed to include no time varying covariates, e. g. no follow up visits are allowed. It is assumed that the covariates stay constant over time, in which no information is available.

## Usage

```
dataLong(
  dataShort,
  timeColumn,
  eventColumn,
  timeAsFactor = FALSE,
  remLastInt = FALSE,
  aggTimeFormat = FALSE,
  lastTheoInt = NULL
)
```

## Arguments

| | |
|---|---|
| dataShort | Original data in short format ("class data.frame"). |
| timeColumn | Character giving the column name of the observed times. It is required that the observed times are discrete ("integer vector"). |
| eventColumn | Column name of the event indicator ("character vector"). It is required that this is a binary variable with 1=="event" and 0=="censored". |
| timeAsFactor | Should the time intervals be coded as factor ("logical vector")? Default is FALSE. In the default settings the column is treated as quantitative variable ("numeric vector"). |
| remLastInt | Should the last theoretical interval be removed in long format ("logical vector")? Default setting (FALSE) is no deletion. This is only important, if the short format data includes the last theoretic interval [a_q, Inf). There are only events in the last theoretic interval, so the discrete hazard is always one and these observations have to be excluded for estimation. |
| aggTimeFormat | Instead of the usual long format, should every observation have all time intervals ("logical vector")? Default is standard long format (FALSE). In the case of nonlinear risk score models, the time effect has to be integrated out before these can be applied to the C-index. |
| lastTheoInt | Gives the number of the last theoretic interval ("integer vector"). Only used, if argument *aggTimeFormat* is set to TRUE. |

## Details

If the data has continuous survival times, the response may be transformed to discrete intervals using function contToDisc. If the data set has time varying covariates the function dataLongTimeDep should be used instead. In the case of competing risks and no time varying covariates see function dataLongCompRisks.

## Value

Original data.frame with three additional columns:

- obj Index of persons as integer vector
- timeInt Index of time intervals (factor)

- y Response in long format as binary vector. 1=="event happens in period timeInt" and zero otherwise. If argument *responseAsFactor* is set to TRUE, then responses will be coded as factor in one column.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

Matthias Schmid <matthias.schmid@imbie.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

Fahrmeir L (2005). "Discrete Survival-Time Models." In *Encyclopedia of Biostatistics*, chapter Survival Analysis. John Wiley \& Sons.

Thompson Jr. WA (1977). "On the Treatment of Grouped Observations in Life Studies." *Biometrics*, **33**, 463-470.

## See Also

contToDisc, dataLongTimeDep, dataLongCompRisks

## Examples

```
# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
subUnempDur <- UnempDur [1:100, ]
head(subUnempDur)

# Convert to long format
UnempLong <- dataLong (dataShort = subUnempDur, timeColumn = "spell", eventColumn = "censor1")
head(UnempLong, 20)

# Is there exactly one observed event of y for each person?
splitUnempLong <- split(UnempLong, UnempLong$obj)
all(sapply(splitUnempLong, function (x) sum(x$y))==subUnempDur$censor1) # TRUE

# Second example: Acute Myelogenous Leukemia survival data
library(survival)
head(leukemia)
leukLong <- dataLong(dataShort = leukemia, timeColumn = "time",
eventColumn = "status", timeAsFactor=TRUE)
head(leukLong, 30)

# Estimate discrete survival model
estGlm <- glm(formula = y ~ timeInt + x, data=leukLong, family = binomial())
summary(estGlm)
```

```
# Estimate survival curves for non-maintained chemotherapy
newDataNonMaintained <- data.frame(timeInt = factor(1:161), x = rep("Nonmaintained"))
predHazNonMain <- predict(estGlm, newdata = newDataNonMaintained, type = "response")
predSurvNonMain <- cumprod(1-predHazNonMain)

# Estimate survival curves for maintained chemotherapy
newDataMaintained <- data.frame(timeInt = factor(1:161), x = rep("Maintained"))
predHazMain <- predict(estGlm, newdata = newDataMaintained, type = "response")
predSurvMain <- cumprod(1-predHazMain)

# Compare survival curves
plot(x = 1:50, y = predSurvMain [1:50], xlab = "Time", ylab = "S(t)", las = 1,
type = "l", main = "Effect of maintained chemotherapy on survival of leukemia patients")
lines(x = 1:161, y = predSurvNonMain, col = "red")
legend("topright", legend = c("Maintained chemotherapy", "Non-maintained chemotherapy"),
col = c("black", "red"), lty = rep(1, 2))
# The maintained therapy has clearly a positive effect on survival over the time range

##############################################
# Simulation
# Single event in case of right-censoring

# Simulate multivariate normal distribution
library(discSurv)
library(mvnfast)
set.seed(-1980)
X <- mvnfast::rmvn(n = 1000, mu = rep(0, 10), sigma = diag(10))


# Specification of discrete hazards with 11 theoretical intervals
betaCoef <- seq(-1, 1, length.out = 11)[-6]
timeInt <- seq(-1, 1, length.out = 10)
linPred <- c(X %*% betaCoef)
hazTimeX <- cbind(sapply(1:length(timeInt),
                  function(x) exp(linPred+timeInt[x]) / (1+exp(linPred+timeInt[x])) ), 1)


# Simulate discrete survival and censoring times in 10 observed intervals
discT <- rep(NA, dim(hazTimeX)[1])
discC <- rep(NA, dim(hazTimeX)[1])
for( i in 1:dim(hazTimeX)[1] ){

 discT[i] <- sample(1:11, size = 1, prob = estMargProb(haz=hazTimeX[i, ]))
 discC[i] <- sample(1:11, size = 1, prob = c(rep(1/11, 11)))
}


# Calculate observed times, event indicator and specify short data format
eventInd <- discT <= discC
obsT <- ifelse(eventInd, discT, discC)
eventInd[obsT == 11] <- 0
obsT[obsT == 11] <- 10
```

```
simDatShort <- data.frame(obsT = obsT, event = as.numeric(eventInd), X)


# Convert data to discrete data long format
simDatLong <- dataLong(dataShort = simDatShort, timeColumn = "obsT", eventColumn = "event",
timeAsFactor=TRUE)


# Estimate discrete-time continuation ratio model
formSpec <- as.formula(paste("y ~ timeInt + ",
                          paste(paste("X", 1:10, sep=""), collapse = " + "), sep = ""))
modelFit <- glm(formula = formSpec, data = simDatLong, family = binomial(link = "logit"))
summary(modelFit)


# Compare estimated to true coefficients
coefModel <- coef(modelFit)
MSE_covariates <- mean((coefModel[11:20]-timeInt)^2)
MSE_covariates
# -> Estimated coefficients are near true coefficients
```

---

`dataLongCompRisks` *Data Long Competing Risks Transformation*

---

## Description

Transforms short data format to long format for discrete survival modelling in the case of competing risks with right censoring. It is assumed that the covariates are not time varying.

## Usage

```
dataLongCompRisks(
  dataShort,
  timeColumn,
  eventColumns,
  eventColumnsAsFactor = FALSE,
  timeAsFactor = FALSE,
  aggTimeFormat = FALSE,
  lastTheoInt = NULL,
  responseAsFactor = FALSE
)
```

## Arguments

| | |
|---|---|
| `dataShort` | Original data in short format ("class data.frame"). |
| `timeColumn` | Character giving the column name of the observed times ("character vector"). It is required that the observed times are discrete ("integer vector"). |

`eventColumns` Character vector giving the column names of the event indicators (excluding censoring column)("character vector"). It is required that all events are binary encoded. If the sum of all event indicators is zero, then this is interpreted as a censored observation. Alternatively a column name of a factor representing competing events can be given. In this case the argument *eventColumnsAsFactor* has to be set TRUE and the first level is assumed to represent censoring.

`eventColumnsAsFactor`
Should the argument *eventColumns* be interpreted as column name of a factor variable ("logical vector")? Default is FALSE.

`timeAsFactor` Should the time intervals be coded as factor ("logical vector")? Default is FALSE. In the default settings the discrete time variable are treated as quantitative.

`aggTimeFormat` Instead of the usual long format, should every observation have all time intervals ("logical vector")? Default is standard long format. In the case of nonlinear risk score models, the time effect has to be integrated out before these can be applied to the C-index.

`lastTheoInt` Gives the number of the last theoretic interval ("integer vector"). Only used, if *aggTimeFormat* is set to TRUE.

`responseAsFactor`
Should the response columns be given as factor ("logical vector")? Default is FALSE.

## Details

It is assumed, that only one event happens at a specific time point (competing risks). Either the observation is censored or one of the possible events takes place.

In contrast to continuous survival (see e. g. `Surv`) the start and stop time notation is not used here. In discrete time survival analysis the only relevant information is to use the stop time. Start time does not matter, because all discrete intervals need to be included in the long data set format to ensure consistent estimation. It is assumed that the supplied data set *dataShort* contains all repeated measurements of each cluster (e. g. persons). For further information see example *Start-stop notation*.

## Value

Original data set in long format with additional columns

- obj Gives identification number of objects (row index in short format) (integer)
- timeInt Gives number of discrete time intervals (factor)
- responses Columns with dimension count of events + 1 (censoring)
  - e0 No event (observation censored in specific interval)
  - e1 Indicator of first event, 1 if event takes place and 0 otherwise
  - ... ...
  - ek Indicator of last k-th event, 1 if event takes place and zero otherwise

If argument responseAsFactor=TRUE, then responses will be coded as factor in one column.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

Steele F, Goldstein H, Browne W (2004). "A general multilevel multistate competing risks model for event history data, with an application to a study of contraceptive use dynamics." *Statistical Modelling*, **4**, 145-159.

Narendranathan W, Stewart MB (1993). "Modelling the Probability of Leaving Unemployment: Competing Risks Models with Flexible Base-Line Hazards." *Journal of the Royal Statistical Society Series C*, **42**, 63-83.

## See Also

contToDisc, dataLongTimeDep, dataLongCompRisksTimeDep

## Examples

```
# Example with unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
SubUnempDur <- UnempDur [1:100, ]

# Convert competing risk data to long format
SubUnempDurLong <- dataLongCompRisks (dataShort = SubUnempDur, timeColumn = "spell",
eventColumns = c("censor1", "censor2", "censor3", "censor4"))
head(SubUnempDurLong, 20)

# Fit multinomial logit model with VGAM package
# with one coefficient per response
library(VGAM)
multLogitVGM <- vgam(cbind(e0, e1, e2, e3, e4) ~ timeInt + ui + age + logwage,
                     family = multinomial(refLevel = 1),
                     data = SubUnempDurLong)
coef(multLogitVGM)

# Alternative: Use nnet
# Convert response to factor
rawResponseMat <- SubUnempDurLong[, c("e0", "e1", "e2", "e3", "e4")]
NewFactor <- factor(unname(apply(rawResponseMat, 1, function(x) which(x == 1))),
                     labels = colnames(rawResponseMat))

# Include recoded response in data
SubUnempDurLong <- cbind(SubUnempDurLong, NewResp = NewFactor)

# Construct formula of mlogit model
```

```
mlogitFormula <- formula(NewResp ~ timeInt + ui + age + logwage)

# Fit multinomial logit model
# with one coefficient per response
library(nnet)
multLogitNNET <- multinom(formula = mlogitFormula, data = SubUnempDurLong)
coef(multLogitNNET)

##########################################################
# Simulation
# Cause specific competing risks in case of right-censoring
# Discrete subdistribution hazards model

# Simulate covariates as multivariate normal distribution
library(mvnfast)
set.seed(1980)
X <- mvnfast::rmvn(n = 1000, mu = rep(0, 4), sigma = diag(4))

# Specification of two discrete cause specific hazards with four intervals
# Event 1
theoInterval <- 4
betaCoef_event1 <- seq(-1, 1, length.out = 5)[-3]
timeInt_event1 <- seq(0.1, -0.1, length.out = theoInterval-1)
linPred_event1 <- c(X %*% betaCoef_event1)
# Event 2
betaCoef_event2 <- seq(-0.5, 0.5, length.out = 5)[-3]
timeInt_event2 <- seq(-0.1, 0.1, length.out = theoInterval-1)
linPred_event2 <- c(X %*% betaCoef_event2)
# Discrete cause specific hazards in last theoretical interval
theoHaz_event1 <- 0.5
theoHaz_event2 <- 0.5

haz_event1_X <- cbind(sapply(1:length(timeInt_event1),
                function(x) exp(linPred_event1 + timeInt_event1[x]) /
                (1 + exp(linPred_event1 + timeInt_event1[x]) +
                exp(linPred_event2 + timeInt_event2[x])) ), theoHaz_event1)

haz_event2_X <- cbind(sapply(1:length(timeInt_event2),
                function(x) exp(linPred_event2 + timeInt_event2[x]) /
                (1 + exp(linPred_event1 + timeInt_event1[x]) +
                exp(linPred_event2 + timeInt_event2[x]) ) ), theoHaz_event2)
allCauseHaz_X <- haz_event1_X + haz_event2_X


pT_X <- t(sapply(1:dim(allCauseHaz_X)[1], function(i) estMargProb(allCauseHaz_X[i, ]) ))


pR_T_X_event1 <- haz_event1_X / (haz_event1_X + haz_event2_X)


survT <- sapply(1:dim(pT_X)[1], function(i) sample(x = 1:(length(timeInt_event1) + 1),
                                                   size = 1, prob = pT_X[i, ]) )
censT <- sample(x = 1:(length(timeInt_event1)+1), size = dim(pT_X)[1],
```

```
                prob = rep(1/(length(timeInt_event1) + 1), (length(timeInt_event1) + 1)),
                replace = TRUE)


  obsT <- ifelse(survT <= censT, survT, censT)
  obsEvent <- rep(0, length(obsT))
  obsEvent <- sapply(1:length(obsT),
                   function(i) if(survT[i] <= censT[i]){
                     return(sample(x = c(1, 2), size=1,
                     prob = c(pR_T_X_event1[i, obsT[i]  ],
                     1 - pR_T_X_event1[i, obsT[i]  ]) ))
                   } else{

                     return(0)
                   }
  )


  # Recode last interval to censored
  lastInterval <- obsT == theoInterval
  obsT[lastInterval] <- theoInterval - 1
  obsEvent[lastInterval] <- 0
  obsT <- factor(obsT)
  obsEvent <- factor(obsEvent)

  datShort <- data.frame(event = factor(obsEvent), time = obsT, X)
  datLong <- dataLongCompRisks(dataShort = datShort, timeColumn = "time",
                               eventColumns = "event", responseAsFactor = TRUE,
                               eventColumnsAsFactor = TRUE, timeAsFactor = TRUE)


  # Estimate discrete cause specific hazard model
  library(VGAM)
  estModel <- vglm(formula=responses ~ timeInt + X1 + X2 + X3 + X4, data=datLong,
                   family = multinomial(refLevel = 1))


  # Mean squared errors per event
  coefModels <- coef(estModel)
  mean((coefModels[seq(7, length(coefModels), 2)] - betaCoef_event1)^2) # Event 1
  mean((coefModels[seq(8, length(coefModels), 2)] - betaCoef_event2)^2) # Event 2
  # -> Estimated coefficients are near true coefficients for each event type
```

---

## dataLongCompRisksTimeDep

*Data Long Competing Risks Time Dependent Covariates Transformation*

---

## Description

Transforms short data format to long format for discrete survival modelling in the case of competing risks with right censoring. Covariates may vary over time.

## Usage

```
dataLongCompRisksTimeDep(
  dataSemiLong,
  timeColumn,
  eventColumns,
  eventColumnsAsFactor = FALSE,
  idColumn,
  timeAsFactor = FALSE,
  responseAsFactor = FALSE
)
```

## Arguments

| | |
|---|---|
| `dataSemiLong` | Original data in semi-long format ("class data.frame"). |
| `timeColumn` | Character giving the column name of the observed times("logical vector"). It is required that the observed times are discrete ("integer vector"). |
| `eventColumns` | Character vector giving the column names of the event indicators (excluding censoring column)("character vector"). It is required that all events are binary encoded. If the sum of all event indicators is zero, then this is interpreted as a censored observation. Alternatively a column name of a factor representing competing events can be given. In this case the argument *eventColumnsAsFactor* has to be set TRUE and the first level is assumed to represent censoring. |
| `eventColumnsAsFactor` | Should the argument eventColumns be intepreted as column name of a factor variable ("logical vector")? Default is FALSE. |
| `idColumn` | Name of column of identification number of persons as character("character vector"). |
| `timeAsFactor` | Should the time intervals be coded as factor ("logical vector")? Default is FALSE. In the default settings the discrete time intervals are treated as quantitative ("numeric vector"). |
| `responseAsFactor` | Should the response columns be given as factor ("logical vector")? Default is FALSE. |

## Details

There may be some intervals, where no additional information on the covariates is observed (e. g. observed values in interval one and three but two is missing). In this case it is assumed, that the values from the last observation stay constant over time until a new measurement was done.

In contrast to continuous survival (see e. g. `Surv`) the start and stop time notation is not used here. In discrete time survival analysis the only relevant information is to use the stop time. Start time does not matter, because all discrete intervals need to be included in the long data set format to

ensure consistent estimation. It is assumed that the supplied data set *dataSemiLong* contains all repeated measurements of each cluster in semi-long format (e. g. persons). For further information see example *Start-stop notation*.

## Value

Original data set in long format with additional columns

- obj Gives identification number of objects (row index in short format) (integer)
- timeInt Gives number of discrete time intervals (factor)
- responses Columns with dimension count of events + 1 (censoring)
  - e0 No event (observation censored in specific interval)
  - e1 Indicator of first event, 1 if event takes place and 0 otherwise
  - ... ...
  - ek Indicator of last k-th event, 1 if event takes place and 0 otherwise

If argument responseAsFactor=TRUE, then responses will be coded as factor in one column.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Fahrmeir L (2005). "Discrete Survival-Time Models." In *Encyclopedia of Biostatistics*, chapter Survival Analysis. John Wiley \& Sons.

Thompson Jr. WA (1977). "On the Treatment of Grouped Observations in Life Studies." *Biometrics*, **33**, 463-470.

## See Also

contToDisc, dataLong, dataLongCompRisks

## Examples

```
# Example Primary Biliary Cirrhosis data
library(survival)
pbcseq_example <- pbcseq

# Convert to months
pbcseq_example$day <- ceiling(pbcseq_example$day/30) + 1
names(pbcseq_example)[7] <- "month"
pbcseq_example$status <- factor(pbcseq_example$status)

# Convert to long format for time varying effects
pbcseq_exampleLong <- dataLongCompRisksTimeDep(dataSemiLong = pbcseq_example, timeColumn = "month",
eventColumns = "status", eventColumnsAsFactor = TRUE, idColumn = "id",
timeAsFactor = TRUE)
head(pbcseq_exampleLong)
```

```
#####################
# Start-stop notation

library(survival)
?pbcseq

# Choose subset of patients
subsetID <- unique(pbcseq$id)[1:100]
pbcseq_mod <- pbcseq[pbcseq$id %in% subsetID, ]

# Convert to start stop notation
pbcseq_mod_split <- split(pbcseq_mod, pbcseq_mod$id)
pbcseq_mod_split <- lapply(1:length(pbcseq_mod_split), function(x) {

 cbind(pbcseq_mod_split[[x]],
 start_time=c(0, pbcseq_mod_split[[x]][ - dim(pbcseq_mod_split[[x]])[1], "day"]),
 stop_time=pbcseq_mod_split[[x]][, "day"])

})
pbcseq_mod <- do.call(rbind, pbcseq_mod_split)

# Convert stop time to months
intervalDef <- c(quantile(pbcseq_mod$stop_time, probs = seq(0.1, 0.9, by=0.1)), Inf)
names(pbcseq_mod)
pbcseq_mod <- contToDisc(dataShort = pbcseq_mod, timeColumn = "stop_time",
                         intervalLimits = intervalDef, equi = FALSE)
pbcseq_mod$status <- factor(pbcseq_mod$status)

# Conversion to data long format
pbcseq_mod_long <- dataLongCompRisksTimeDep(dataSemiLong = pbcseq_mod, timeColumn = "timeDisc",
                                            eventColumns = "status",
                                           idColumn = "id",
                                            eventColumnsAsFactor = TRUE,
                                           responseAsFactor = TRUE,
                                           timeAsFactor = TRUE)
head(pbcseq_mod_long)
```

---

dataLongMultiSpell *Data long transformation for multi spell analysis*

---

## Description

Transform data from short format into long format for discrete multi spell survival analysis and right censoring.

## Usage

```
dataLongMultiSpell(
```

```
  dataSemiLong,
  timeColumn,
  eventColumn,
  idColumn,
  timeAsFactor = FALSE,
  spellAsFactor = FALSE
)
```

## Arguments

| | |
|---|---|
| dataSemiLong | Original data in semi-long format ("class data.frame"). |
| timeColumn | Character giving the column name of the observed times. It is required that the observed times are discrete ("character vector"). |
| eventColumn | Column name of the event status ("character vector"). The events can take multiple values on a discrete scale (0, 1, 2, ...) and repetition of events is allowed (integer vector or class factor). It is assumed that the number zero corresponds to censoring and all number > 0 represent the observed states between transitions. |
| idColumn | Name of column of identification number of persons as character("character vector"). |
| timeAsFactor | Should the time intervals be coded as factor ("logical vector")? Default is FALSE. In the default settings the discrete time intervals are treated as quantitative ("numeric vector"). |
| spellAsFactor | Should the spells be coded as factor ("logical vector")? Default is not to use factor. If the argument is false, the column is coded as numeric. |

## Details

If the data has continuous survival times, the response may be transformed to discrete intervals using function contToDisc. The discrete time variable needs to be strictly increasing for each person, because otherwise the order of the events is not distinguishable. Here is an example data structure in short format prior augmentation with three possible states: \ idColumn=1, 1, ... , 1, 2, 2, ... , n \ timeColumn= t_ID1_1 < t_ID1_1 < ... < t_ID1_k, t_ID2_1 < t_ID2_2 < ... < t_ID2_k, ... \ eventColumn = 0, 1, ... , 2, 1, 0, ... , 0

The starting state of each individual is assumed to given with time interval equals zero. For example in an illness-death model with three states ("healthy", "illness", "death") if an individual was healthy at the beginning of the study this has to be encoded with discrete time interval set to zero and event state "healthy".

## Value

Original data.frame with three additional columns:

- obj Index of persons as integer vector
- timeInt Index of time intervals (factor or integer vector)
- spell The spell gives the actual state of each individual within a given discrete interval.

- e0 Response transition in long format as binary vector. Column *e0* represents censoring. If *e0* is coded one in the in the last observed time interval *timeInt* of a person, then this observation was censored.
- e1 Response in long format as binary vector. The column *e1* represents the transition to the first event state.
- eX Response in long format as binary vector. The column *eX* represents the transition to the last event state out of the set of possible states "1, 2, 3, ..., X".
- ... Expanded columns of original data set.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

Fahrmeir L (2005). "Discrete Survival-Time Models." In *Encyclopedia of Biostatistics*, chapter Survival Analysis. John Wiley \& Sons.

Thompson Jr. WA (1977). "On the Treatment of Grouped Observations in Life Studies." *Biometrics*, **33**, 463-470.

## See Also

contToDisc, dataLongTimeDep, dataLongCompRisks, dataLongCompRisks

## Examples

```
################################
# Example with unemployment data
data(unempMultiSpell)

# Select subsample of first 500 persons
unempSub <- unempMultiSpell[unempMultiSpell$id %in% 1:250,]

# Expansion from semi-long to long format
unempLong <- dataLongMultiSpell(dataSemiLong=unempSub, timeColumn = "year",
                                eventColumn="spell", idColumn="id",
                                spellAsFactor=TRUE, timeAsFactor=FALSE)

head(unempLong, 25)

# Fit discrete multi-state model regression model
library(VGAM)

model <- vgam(cbind(e0, e1, e2, e3, e4) ~ 0 + s(timeInt) + age:spell,
data = unempLong, family = multinomial(refLevel="e0"))

############################
```

```
# Example with artificial data

# Seed specification
set.seed(-2578)

# Construction of data set
# Censoring and three possible states (0, 1, 2, 3)
# Discrete time intervals (1, 2, ... , 10)
# Noninfluential variable x ~ N(0, 1)
datFrame <- data.frame(
 ID = c(rep(1, 6), rep(2, 4), rep(3, 3), rep(4, 2), rep(5, 4),
      rep(6, 5), rep(7, 7), rep(8, 8)),
 time = c(c(0, 2, 5, 6, 8, 10), c(0, 1, 6, 7), c(0, 9, 10), c(0, 6), c(0, 2, 3, 4),
        c(0, 3, 4, 7, 9), c(0, 2, 3, 5, 7, 8, 10), c(0, 1, 3, 4, 6, 7, 8, 9) ),
 state = c(c(2, 1, 3, 2, 1, 0), c(3, 1, 2, 2), c(2, 2, 1), c(1, 2), c(3, 2, 2, 0),
         c(1, 3, 2, 1, 3), c(1, 1, 2, 3, 2, 1, 3), c(3, 2, 3, 2, 1, 1, 2, 3) ),
 x = rnorm(n=6+4+3+2+4+5+7+8) )

# Transformation to long format
datFrameLong <- dataLongMultiSpell(dataSemiLong=datFrame, timeColumn="time",
                                   eventColumn="state", idColumn="ID",
                                   spellAsFactor=TRUE)
head(datFrameLong, 25)
library(VGAM)
cRm <- vglm(cbind(e0, e1, e2, e3) ~ 0 + timeInt + x:spell,
data = datFrameLong, family = "multinomial")
summary(cRm)
```

---

dataLongSubDist *Data Matrix and Weights for Discrete Subdistribution Hazard Models*

---

## Description

Generates the augmented data matrix and the weights required for discrete subdistribution hazard modeling with right censoring.

## Usage

```
dataLongSubDist(
  dataShort,
  timeColumn,
  eventColumns,
  eventColumnsAsFactor = FALSE,
  eventFocus,
  timeAsFactor = FALSE,
  aggTimeFormat = FALSE,
  lastTheoInt = NULL
)
```

## Arguments

| | |
|---|---|
| dataShort | Original data in short format ("class data.frame"). |
| timeColumn | Character specifying the column name of the observed event times ("logical vector"). It is required that the observed times are discrete ("integer vector"). |
| eventColumns | Character vector specifying the column names of the event indicators (excluding censoring events) ("logical vector"). It is required that a 0-1 coding is used for all events. The algorithm treats row sums of zero of all event columns as censored. |
| eventColumnsAsFactor | Should the argument *eventColumns* be interpreted as column name of a factor variable ("logical vector")? Default is FALSE. |
| eventFocus | Column name of the event of interest (type 1 event) ("character vector"). |
| timeAsFactor | Logical indicating whether time should be coded as a factor in the augmented data matrix("logical vector"). If FALSE, a numeric coding will be used. |
| aggTimeFormat | Instead of the usual long format, should every observation have all time intervals? ("logical vector") Default is standard long format. In the case of nonlinear risk score models, the time effect has to be integrated out before these can be applied to the C-index. |
| lastTheoInt | Gives the number of the last theoretic interval ("integer vector"). Only used, if aggTimeFormat==TRUE. |

## Details

This function sets up the augmented data matrix and the weights that are needed for weighted maximum likelihood (ML) estimation of the discrete subdistribution model proposed by Berger et al. (2018). The model is a discrete-time extension of the original subdistribution model proposed by Fine and Gray (1999).

## Value

Data frame with additional column "subDistWeights". The latter column contains the weights that are needed for fitting a weighted binary regression model, as described in Berger et al. (2018). The weights are calculated by a life table estimator for the censoring event.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Berger M, Schmid M, Welchowski T, Schmitz-Valckenberg S, Beyersmann J (2020). "Subdistribution Hazard Models for Competing Risks in Discrete Time." *Biostatistics*, **21**, 449-466.

Fine JP, Gray RJ (2012). "A Proportional Hazards Model for the Subdistribution of a Competing Risk." *Journal of the American Statistical Association*, **94**, 496-509.

## See Also

dataLong

## Examples

```
################################
# Example with unemployment data
library(Ecdat)
data(UnempDur)

# Generate subsample, reduce number of intervals to k = 5
SubUnempDur <- UnempDur [1:500, ]
SubUnempDur$time <- as.numeric(cut(SubUnempDur$spell, c(0,4,8,16,28)))

# Convert competing risks data to long format
# The event of interest is re-employment at full job
SubUnempDurLong <- dataLongSubDist (dataShort=SubUnempDur, timeColumn = "time",
eventColumns=c("censor1", "censor2", "censor3"), eventFocus="censor1")
head(SubUnempDurLong)

# Fit discrete subdistribution hazard model with logistic link function
logisticSubDistr <- glm(y ~ timeInt + ui + age + logwage,
                    family=binomial(), data = SubUnempDurLong,
                    weights = SubUnempDurLong$subDistWeights)
summary(logisticSubDistr)

#######################################
# Simulation
# Discrete subdistribution hazards model

# Simulate covariates as multivariate normal distribution
library(mvnfast)
set.seed(1980)
X <- mvnfast::rmvn(n = 1000, mu = rep(0, 4), sigma = diag(4))

# Specification of two discrete cause specific hazards with four intervals
# Event 1
theoInterval <- 4
betaCoef_event1 <- seq(-1, 1, length.out = 5)[-3]
timeInt_event1 <- seq(0.1, -0.1, length.out = theoInterval-1)
linPred_event1 <- c(X %*% betaCoef_event1)
# Event 2
betaCoef_event2 <- seq(-0.5, 0.5, length.out = 5)[-3]
timeInt_event2 <- seq(-0.1, 0.1, length.out = theoInterval-1)
linPred_event2 <- c(X %*% betaCoef_event2)
# Discrete cause specific hazards in last theoretical interval
theoHaz_event1 <- 0.5
theoHaz_event2 <- 0.5

# Derive discrete all cause hazard
haz_event1_X <- cbind(sapply(1:length(timeInt_event1),
                             function(x) exp(linPred_event1 + timeInt_event1[x]) /
                               (1 + exp(linPred_event1 + timeInt_event1[x]) +
                               exp(linPred_event2 + timeInt_event2[x])) ),
                      theoHaz_event1)
```

```
haz_event2_X <- cbind(sapply(1:length(timeInt_event2),
                             function(x) exp(linPred_event2 + timeInt_event2[x]) /
                               (1 + exp(linPred_event1 + timeInt_event1[x]) +
                               exp(linPred_event2 + timeInt_event2[x]) ) ),
                      theoHaz_event2)
allCauseHaz_X <- haz_event1_X + haz_event2_X

# Derive discrete cumulative incidence function of event 1 given covariates
p_T_event1_X <- haz_event1_X * cbind(1, (1-allCauseHaz_X)[, -dim(allCauseHaz_X)[2]])
cumInc_event1_X <-  t(sapply(1:dim(p_T_event1_X)[1], function(x) cumsum(p_T_event1_X[x, ])))

# Calculate all cause probability P(T=t | X)
pT_X <- t(sapply(1:dim(allCauseHaz_X)[1], function(i) estMargProb(allCauseHaz_X[i, ]) ))

# Calculate event probability given time interval P(R=r | T=t, X)
pR_T_X_event1 <- haz_event1_X / (haz_event1_X + haz_event2_X)

# Simulate discrete survival times
survT <- sapply(1:dim(pT_X)[1], function(i) sample(x = 1:(length(timeInt_event1)+1),
                                                   size = 1, prob = pT_X[i, ]) )
censT <- sample(x = 1:(length(timeInt_event1)+1), size = dim(pT_X)[1],
                prob = rep(1/(length(timeInt_event1) + 1), (length(timeInt_event1) + 1)),
                replace = TRUE)

# Calculate observed times
obsT <- ifelse(survT <= censT, survT, censT)
obsEvent <- rep(0, length(obsT))
obsEvent <- sapply(1:length(obsT),
                   function(i) if(survT[i] <= censT[i]){
                     return(sample(x = c(1, 2), size = 1,
                     prob = c(pR_T_X_event1[i, obsT[i]  ],
                     1 - pR_T_X_event1[i, obsT[i]  ]) ))
                   } else{

                     return(0)
                   }
)

# Recode last interval to censored
lastInterval <- obsT == theoInterval
obsT[lastInterval] <- theoInterval-1
obsEvent[lastInterval] <- 0
obsT <- factor(obsT)
obsEvent <- factor(obsEvent)

# Data preparation
datShort <- data.frame(event = factor(obsEvent), time=obsT, X)

# Conversion to long data format
datLongSub <- dataLongSubDist(dataShort = datShort, timeColumn = "time",
                      eventColumns = "event", eventFocus = 1, eventColumnsAsFactor = TRUE)

# Estimate discrete subdistribution hazard model
```

```
estSubModel <- glm(formula = y ~ timeInt + X1 + X2 + X3 + X4, data = datLongSub,
                   family = binomial(link = "logit"), weights = datLongSub$subDistWeights)

# Predict cumulative incidence function of first event
predSubHaz1 <- predict(estSubModel, newdata = datLongSub[datLongSub$obj == 2, ], type = "response")
mean(((1 - estSurv(predSubHaz1)) - cumInc_event1_X[2, 1:3])^2)
```

---

`dataLongTimeDep` *Data Long Time Dependent Covariates*

---

## Description

Transforms short data format to long format for discrete survival modelling of single event analysis with right censoring. Covariates may vary over time.

## Usage

```
dataLongTimeDep(
  dataSemiLong,
  timeColumn,
  eventColumn,
  idColumn,
  timeAsFactor = FALSE
)
```

## Arguments

| | |
|---|---|
| `dataSemiLong` | Original data in semi-long format ("class data.frame"). |
| `timeColumn` | Character giving the column name of the observed times ("character vector"). It is required that the observed times are discrete ("integer vector"). |
| `eventColumn` | Column name of the event indicator ("character vector"). It is required that this is a binary variable with 1=="event" and 0=="censored". |
| `idColumn` | Name of column of identification number of persons ("character vector"). |
| `timeAsFactor` | Should the time intervals be coded as factor ("logical vector")? Default is FALSE. In case of default settings the discrete time intervals are treated as quantitative ("numeric vector"). |

## Details

There may be some intervals, where no additional information on the covariates is observed (e. g. observed values in interval one and three but two is missing). In this case it is assumed, that the values from the last observation stay constant over time until a new measurement was done.

In contrast to continuous survival (see e. g. `Surv`) the start and stop time notation is not used here. In discrete time survival analysis the only relevant information is to use the stop time. Start time does not matter, because all discrete intervals need to be included in the long data set format to

ensure consistent estimation. It is assumed that the supplied data set "dataSemiLong" contains all repeated measurements of each cluster in semi-long format (e. g. persons). For further information see example *Start-stop notation*.

## Value

Original data in long format with three additional columns:

- obj Index of persons as integer vector
- timeInt Index of time intervals (factor)
- y Response in long format as binary vector. 1=="event happens in period timeInt" and zero otherwise

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

Fahrmeir L (2005). "Discrete Survival-Time Models." In *Encyclopedia of Biostatistics*, chapter Survival Analysis. John Wiley \& Sons.

Thompson Jr. WA (1977). "On the Treatment of Grouped Observations in Life Studies." *Biometrics*, **33**, 463-470.

## See Also

contToDisc, dataLong, dataLongCompRisks

## Examples

```
# Example Primary Biliary Cirrhosis data
library(survival)
dataSet1 <- pbcseq

# Only event death is of interest
dataSet1$status [dataSet1$status == 1] <- 0
dataSet1$status [dataSet1$status == 2] <- 1
table(dataSet1$status)

# Convert to months
dataSet1$day <- ceiling(dataSet1$day/30) + 1
names(dataSet1) [7] <- "month"

# Convert to long format for time varying effects
pbcseqLong <- dataLongTimeDep (dataSemiLong = dataSet1, timeColumn = "month",
eventColumn = "status", idColumn = "id")
pbcseqLong [pbcseqLong$obj == 1, ]
```

```
#####################
# Start-stop notation

library(survival)
?survival::heart

# Assume that time was measured on a discrete scale.
# Discrete interval lengths are assumed to vary.
intervalLimits <- quantile(heart$stop, probs = seq(0.1, 1, by=0.1))
intervalLimits[length(intervalLimits)] <- intervalLimits[length(intervalLimits)] + 1
heart_disc <- contToDisc(dataShort = heart, timeColumn = "stop",
intervalLimits = intervalLimits, equi = FALSE)
table(heart_disc$timeDisc)

# Conversion to long format
heart_disc_long <- dataLongTimeDep(dataSemiLong = heart_disc, timeColumn = "timeDisc",
eventColumn = "event", idColumn = "id")
head(heart_disc_long)
```

---

devResid *Deviance Residuals*

---

## Description

Computes the root of the deviance residuals for evaluation of performance in discrete survival analysis.

## Usage

```
devResid(dataLong, hazards)
```

## Arguments

| | |
|---|---|
| dataLong | Original data in long format ("class data.frame"). The correct format can be specified with data preparation, see e. g. dataLong. |
| hazards | Estimated discrete hazards of the data in long format("numeric vector"). Discrete discrete hazards are probabilities and therefore restricted to the interval [0, 1]. |

## Value

- Output List with objects:
  - DevResid Square root of deviance residuals as numeric vector.
- Input A list of given argument input values (saved for reference)

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

Tutz G (2012). *Regression for Categorical Data*. Cambridge University Press.

## See Also

adjDevResid, predErrCurve

## Examples

```
library(survival)

# Transform data to long format
heart[, "stop"] <- ceiling(heart[, "stop"])
set.seed(0)
Indizes <- sample(unique(heart$id), 25)
randSample <- heart[unlist(sapply(1:length(Indizes),
function(x) which(heart$id == Indizes[x]))),]
heartLong <- dataLongTimeDep(dataSemiLong = randSample,
timeColumn = "stop", eventColumn = "event", idColumn = "id", timeAsFactor = FALSE)

# Fit a generalized, additive model and predict discrete hazards on data in long format
library(mgcv)
gamFit <- gam(y ~ timeInt + surgery + transplant + s(age), data = heartLong, family = "binomial")
hazPreds <- predict(gamFit, type = "response")

# Calculate the deviance residuals
devResiduals <- devResid (dataLong = heartLong, hazards = hazPreds)$Output$DevResid

# Compare with estimated normal distribution
plot(density(devResiduals),
main = "Empirical density vs estimated normal distribution",
las = 1, ylim = c(0, 0.5))
tempFunc <- function (x) dnorm(x, mean = mean(devResiduals), sd = sd(devResiduals))
curve(tempFunc, xlim = c(-10, 10), add = TRUE, col = "red")
# The empirical density seems like a mixture distribution,
# but is not too far off in with values greater than 3 and less than 1
```

---

# estCumInz *Estimates Cumulative Incidence Function for Discrete Time Competing Risks Models*

---

## Description

Estimates the cumulative incidence function of a discrete time competing risks model given covariates P(T <= t, event = k | x).

## Usage

```
estCumInz(hazards, eventFocus)
```

## Arguments

| | |
|---|---|
| hazards | Estimated discrete hazard rates of all events ("numeric matrix"). Each column represents one event. The first column is assumed to contain the censoring case and the discrete hazards should only vary over time in each row. |
| eventFocus | Column that represent the discrete hazards of the primary event ("integer vector"). |

## Details

The covariates set is required to be constant across rows.

## Value

Returns cumulative incidence function of the primary event. If argument *nonparCI* is set to TRUE, then a list is returned: The first element includes the cumulative incidence function. The second list element contains the lower and the third list element the upper bound of the pointwise confidence intervals.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Lee M, Feuer EJ, Fine JP (2018). "On the analysis of discrete time competing risks data." *Biometrics*, **74**, 1468-1481.

## See Also

compRisksGEE, dataLongCompRisks, dataLongCompRisksTimeDep, geeglm,

## Examples

```
# Example with unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
SubUnempDur <- UnempDur [1:100, ]

# Estimate GEE models for all events
estGEE <- compRisksGEE(datShort = SubUnempDur, dataTransform = "dataLongCompRisks",
corstr = "independence", formulaVariable =~ timeInt + age + ui + logwage * ui,
eventColumns = c("censor1", "censor2", "censor3", "censor4"), timeColumn = "spell")

# Estimate hazards of all events given the covariates of third person
```

```
SubUnempDurLong <- dataLongCompRisks(dataShort = SubUnempDur,
eventColumns = c("censor1", "censor2", "censor3", "censor4"), timeColumn = "spell")
preds <- predict(estGEE, subset(SubUnempDurLong, obj == 3))

# Estimate cumulative incidence function
cumInzGEE <- estCumInz(preds, eventFocus = 2)
cumInzGEE
```

---

`estMargProb` *Estimated Marginal Probabilities*

---

## Description

Estimates the marginal probability P(T=t|x) based on estimated discrete hazards. The discrete hazards may or may not depend on covariates. The covariates have to be equal across all estimated hazard rates. Therefore the given discrete hazards should only vary over time.

## Usage

```
estMargProb(hazards)
```

## Arguments

| | |
|---|---|
| `hazards` | Estimated discrete hazards ("numeric vector") |

## Details

The argument *hazards* must be given for all intervals [a_0, a_1), [a_1, a_2), ..., [a_q-1, a_q), [a_q, Inf).

## Value

Estimated marginal probabilities ("numeric vector")

## Note

It is assumed that all time points up to the last interval [a_q, Inf) are available. If not already present, these can be added manually.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

## See Also

estSurv

## Examples

```
# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
subUnempDur <- UnempDur [1:100, ]

# Convert to long format
UnempLong <- dataLong(dataShort = subUnempDur, timeColumn = "spell", eventColumn = "censor1")
head(UnempLong)

# Estimate binomial model with logit link
Fit <- glm(formula = y ~ timeInt + age + logwage, data = UnempLong, family = binomial())

# Estimate discrete survival function given age, logwage of first person
hazard <- predict(Fit, newdata = subset(UnempLong, obj == 1), type = "response")

# Estimate marginal probabilities given age, logwage of first person
MarginalProbCondX <- estMargProb (c(hazard, 1))
MarginalProbCondX
sum(MarginalProbCondX)==1 # TRUE: Marginal probabilities must sum to 1!
```

---

# estMargProbCompRisks *Estimated Marginal Probabilities for Competing Risks*

---

## Description

Estimates the marginal probability P(T = t,R = r|x) based on estimated discrete hazards of a competing risks model. The discrete hazards may or may not depend on covariates. The covariates have to be equal across all estimated discrete hazards. Therefore the given discrete hazards should only vary over time.

## Usage

```
estMargProbCompRisks(hazards)
```

## Arguments

| | |
|---|---|
| hazards | Estimated discrete hazards ("numeric matrix"). Discrete hazards of each time interval are stored in the rows and the number of columns equal to the number of events. |

## Details

The argument *hazards* must be given for all intervals [a_0, a_1), [a_1, a_2), ..., [a_q-1, a_q), [a_q, Inf).

## Value

Estimated marginal probabilities ("numeric matrix")

## Note

It is assumed that all time points up to the last interval [a_q, Inf) are available. If not already present, these can be added manually. In competing risk settings the marginal probabilities of the last theoretical interval depend on the assumptions on the discrete hazards in the last theoretical interval. However the estimation of all previous discrete intervals is not affected by those assumptions.

## Author(s)

Moritz Berger <moritz.berger@imbie.uni-bonn.de>
https://www.imbie.uni-bonn.de/personen/dr-moritz-berger/

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

## See Also

estMargProb

## Examples

```
# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
subUnempDur <- UnempDur [1:100, ]

# Convert to long format
UnempLong <- dataLongCompRisks(dataShort = subUnempDur, timeColumn = "spell",
eventColumns = c("censor1", "censor4"))
head(UnempLong)

# Estimate continuation ratio model with logit link
vglmFit <- VGAM::vglm(formula = cbind(e0, e1, e2) ~ timeInt + age + logwage, data = UnempLong,
family = VGAM::multinomial(refLevel = "e0"))

# Estimate discrete survival function given age, logwage of first person
hazards <- VGAM::predictvglm(vglmFit, newdata = subset(UnempLong, obj == 1), type = "response")[,-1]

# Estimate marginal probabilities given age, logwage of first person
# Example 1
```

```
# Assumption: Discrete hazards in last theoretical interval are equal for both event types
MarginalProbCondX <- estMargProbCompRisks(rbind(hazards, 0.5))
MarginalProbCondX
all.equal(sum(MarginalProbCondX), 1) # TRUE: Marginal probabilities must sum to 1!

# Example 2
# Assumption: Discrete hazards in last theoretical interval are event1=, event2=
MarginalProbCondX2 <- estMargProbCompRisks(rbind(hazards, c(0.75, 0.25)))
MarginalProbCondX2
all.equal(sum(MarginalProbCondX2), 1) # TRUE: Marginal probabilities must sum to 1!

# Compare marginal probabilities given X
all.equal(MarginalProbCondX[1:5, ], MarginalProbCondX2[1:5, ])
all.equal(MarginalProbCondX[6, ], MarginalProbCondX2[6, ])
```

---

`estRecal` *Logistic recalibration based on linear predictors*

---

## Description

Fits a logistic recalibration model to independent test data. It updates the intercept and slope parameters. It is assumed that the factor levels of time are equal in both training and validation data. Time dependent covariates, discrete cause specific competing risks and subdistribution hazards are also supported.

## Usage

```
estRecal(testLinPred, testDataLong, weights = NULL)
```

## Arguments

| | |
|---|---|
| `testLinPred` | Calculated linear predictor on the validation data with model fitted on training data ("numeric vector"). |
| `testDataLong` | Validation data set in long format ("class data.frame"). |
| `weights` | Weights used in estimation of the logistic recalibration model ("numeric vector"). Default is no weighting (NULL). |

## Details

Updates estimated hazards of discrete survival models to better adapt to a different environment. If there are substantial environment changes the predicted probabilities will differ between two environments. Logistic recalibration may be used to improve the calibration of predicted probabilities by incorperating information from the existing model and data from the environment. This approach works for any survival prediction model with one event that provides linear predictors.

## Value

Continuation ratio model that calibrates estimated discrete hazards to new validation environment ("class glm, lm").

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Heyard R, Timsit J, Held L, COMBACTE-MAGNET,consortium (2019). "Validation of discrete time-to-event prediction models in the presence of competing risks." *Biometrical Journal*, **62**, 643-657.

## See Also

(Calibration plots links) dataLong, dataLongTimeDep, dataLongCompRisks, dataLongCompRisksTimeDep, dataLongSubDist

## Examples

```
####################
# Data preprocessing

# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
selectInd1 <- 1:100
selectInd2 <- 101:200
trainSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd1], ]
valSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd2], ]

####################
# One event

# Convert to long format
trainSet_long <- dataLong(dataShort = trainSet, timeColumn = "spell", eventColumn = "censor1")
valSet_long <- dataLong(dataShort = valSet, timeColumn = "spell", eventColumn = "censor1")

# Estimate continuation ratio model with logit link
glmFit <- glm(formula = y ~ timeInt + age + logwage, data = trainSet_long, family = binomial())

# Calculate linear predictors on validation set
linPred <- predict(glmFit, newdata = valSet_long, type = "link")

# Estimate logistic recalibration model
recalModel <- estRecal(testLinPred = linPred, testDataLong = valSet_long)
summary(recalModel)
```

```
# Calibration plots
hazOrg <- predict(glmFit, newdata = valSet_long, type = "response")
hazRecal <- predict(recalModel, newdata = data.frame(linPred), type = "response")

# Before logistic recalibration
calibrationPlot(hazOrg, testDataLong = valSet_long)
# After logistic recalibration
calibrationPlot(hazRecal, testDataLong = valSet_long)

############################
# Two cause specific hazards
library(VGAM)

# Convert to long format
trainSet_long <- dataLongCompRisks(dataShort = trainSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"))
valSet_long <- dataLongCompRisks(dataShort = valSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"))

# Estimate continuation ratio model with logit link
vglmFit <- VGAM::vglm(formula = cbind(e0, e1, e2) ~ timeInt + age + logwage, data = trainSet_long,
family = VGAM::multinomial(refLevel = "e0"))

# Calculate linear predictors on training and test set
linPred <- VGAM::predictvglm(vglmFit, newdata = valSet_long, type = "link")

# Estimate logistic recalibration model
recalModel <- estRecal(testLinPred = linPred, testDataLong = valSet_long)
recalModel

# Calibration plots
hazOrg <- predict(vglmFit, newdata = valSet_long, type = "response")
predDat <- as.data.frame(linPred)
names(predDat) <- recalModel@misc$colnames.x[-1]
hazRecal <- predictvglm(recalModel, newdata = predDat, type = "response")

# Before logistic recalibration
calibrationPlot(hazOrg, testDataLong = valSet_long, event = "e1")
calibrationPlot(hazOrg, testDataLong = valSet_long, event = "e2")
# After logistic recalibration
calibrationPlot(hazRecal, testDataLong = valSet_long, event = "e1")
calibrationPlot(hazRecal, testDataLong = valSet_long, event = "e2")

##############################
# Subdistribution hazards model

# Convert to long format
trainSet_long <- dataLongSubDist(dataShort = trainSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"), eventFocus = "censor1")
valSet_long <- dataLongSubDist(dataShort = valSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"), eventFocus = "censor1")

# Estimate continuation ratio model with logit link
```

```
glmFit <- glm(formula = y ~ timeInt + age + logwage, data = trainSet_long,
family = binomial(), weights = trainSet_long$subDistWeights)

# Calculate linear predictors on training and test set
linPred <- predict(glmFit, newdata = valSet_long, type = "link")

# Estimate logistic recalibration model
recalModel <- estRecal(testLinPred = linPred, testDataLong = valSet_long,
weights = valSet_long$subDistWeights)
recalModel

# Calibration plots
hazOrg <- predict(glmFit, newdata = valSet_long, type = "response",
weights = valSet_long$subDistWeights)
hazRecal <- predict(recalModel, newdata = data.frame(linPred), type = "response",
weights = valSet_long$subDistWeights)

# Before logistic recalibration
calibrationPlot(hazOrg, testDataLong = valSet_long,
weights=valSet_long$subDistWeights)
# After logistic recalibration
calibrationPlot(hazRecal, testDataLong = valSet_long,
weights=valSet_long$subDistWeights)
```

---

`estSurv` *Estimated Survival Function*

---

## Description

Estimates the survival function S(T = t|x) based on estimated hazard rates. The hazard rates may or may not depend on covariates. The covariates have to be equal across all estimated hazard rates. Therefore the given hazard rates should only vary over time.

## Usage

```
estSurv(haz)
```

## Arguments

| | |
|---|---|
| `haz` | Estimated hazard rates ("numeric vector") |

## Details

The argument *haz* must be given for all intervals [a_0, a_1), [a_1, a_2), ..., [a_q-1, a_q), [a_q, Inf).

## Value

Estimated probabilities of survival ("numeric vector")

## Note

It is assumed that all time points up to the last theoretical interval [a_q, Inf) are available. If not already present, these can be added manually.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

## See Also

estMargProb

## Examples

```
# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
subUnempDur <- UnempDur [1:100, ]

# Convert to long format
UnempLong <- dataLong(dataShort = subUnempDur, timeColumn = "spell", eventColumn = "censor1")
head(UnempLong)

# Estimate binomial model with logit link
Fit <- glm(formula = y ~ timeInt + age + logwage, data=UnempLong, family = binomial())

# Estimate discrete survival function given age, logwage of first person
hazard <- predict(Fit, newdata = subset(UnempLong, obj == 1), type = "response")
SurvivalFuncCondX <- estSurv(c(hazard, 1))
SurvivalFuncCondX
```

---

# estSurvCens *Estimated Survival Function of Censoring Process*

---

## Description

Estimates the marginal survival function G(T=t) of the censoring process based on a life table estimator. Compatible with single event and competing risks data.

## Usage

```
estSurvCens(dataShort, timeColumn, eventColumns)
```

## Arguments

| | |
|---|---|
| dataShort | Data in original short format ("class data.frame"). |
| timeColumn | Name of column with discrete time intervals ("character vector"). |
| eventColumns | Names of the event columns of dataShort("character vector"). In the competing risks case the event columns have to be in dummy encoding format ("numeric vector"). |

## Value

Named vector of estimated survival function of the censoring process for all time points except the last theoretical interval.

## Note

In the censoring survival function the last time interval [a_q, Inf) has the probability of zero.

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

## See Also

estSurv

## Examples

```
# Load unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
subUnempDur <- UnempDur [1:100, ]

######################
# Single event example

# Estimate censoring survival function G(t)
estG <- estSurvCens(dataShort = subUnempDur, timeColumn = "spell",
eventColumns = "censor1")
estG

########################
# Competing risks example

# Estimate censoring survival function G(t)
```

```
estG <- estSurvCens(dataShort = subUnempDur, timeColumn = "spell",
eventColumns = c("censor1", "censor2", "censor3", "censor4"))
estG
```

---

`estSurvCompRisks` *Estimated Survival Function for Competing Risks*

---

## Description

Computes the survival function S(T>tlx) based on estimated hazards of a competing risks model. The discrete hazards may or may not depend on covariates. The covariates have to be equal across all estimated hazards. Therefore the given discrete hazards should only vary over time.

## Usage

```
estSurvCompRisks(hazards)
```

## Arguments

| | |
|---|---|
| `hazards` | Estimated discrete hazards ("numeric matrix"). Discrete hazards of each time interval are stored in the rows and the number of columns equal to the number of events. |

## Details

The argument *hazards* must be given for all intervals [a_0, a_1), [a_1, a_2), ..., [a_q-1, a_q), [a_q, Inf).

## Value

Estimated survival probabilities ("numeric vector")

## Note

It is assumed that all time points up to the last interval [a_q, Inf) are available. If not already present, these can be added manually.

## Author(s)

Moritz Berger <moritz.berger@imbie.uni-bonn.de>
https://www.imbie.uni-bonn.de/personen/dr-moritz-berger/

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

## See Also

estSurv

## Examples

```
# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
subUnempDur <- UnempDur [1:100, ]

# Convert to long format
UnempLong <- dataLongCompRisks(dataShort = subUnempDur, timeColumn = "spell",
eventColumns = c("censor1", "censor4"))
head(UnempLong)

# Estimate continuation ratio model with logit link
vglmFit <- VGAM::vglm(formula = cbind(e0, e1, e2) ~ timeInt + age + logwage, data = UnempLong,
family = VGAM::multinomial(refLevel = "e0"))

# Estimate discrete survival function given age, logwage of first person
hazards <- VGAM::predictvglm(vglmFit, newdata = subset(UnempLong, obj == 1), type = "response")[,-1]
SurvivalFuncCondX <- estSurvCompRisks(rbind(hazards, 0.5))
SurvivalFuncCondX
```

---

# gumbel *Gumbel Link Function*

---

## Description

Constructs the link function with gumbel distribution in approriate format for use in generalized, linear models.

## Usage

```
gumbel()
```

## Details

Insert this function into a binary regression model

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

Matthias Schmid <matthias.schmid@imbie.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

## See Also

glm

## Examples

```
# Example with copenhagen stroke study
library(pec)
data(cost)
head(cost)

# Take subsample and convert time to months
costSub <- cost [1:50, ]
costSub$time <- ceiling(costSub$time/30)
costLong <- dataLong(dataShort = costSub, timeColumn = "time", eventColumn = "status",
timeAsFactor=TRUE)
gumbelModel <- glm(formula = y ~ timeInt + diabetes, data = costLong,
family = binomial(link = gumbel()))

# Estimate hazard given prevStroke and no prevStroke
hazPrevStroke <- predict(gumbelModel, newdata=data.frame(timeInt = factor(1:143),
diabetes = factor(rep("yes", 143), levels = c("no", "yes"))), type = "response")
hazWoPrevStroke <- predict(gumbelModel, newdata = data.frame(timeInt = factor(1:143),
diabetes=factor(rep("no", 143), levels = c("no", "yes"))), type = "response")

# Estimate survival function
SurvPrevStroke <- cumprod(1 - hazPrevStroke)
SurvWoPrevStroke <- cumprod(1 - hazWoPrevStroke)

# Example graphics of survival curves with and without diabetes
plot(x = 1:143, y = SurvWoPrevStroke, type = "l", xlab = "Months",
ylab = "S (t|x)", las = 1, lwd = 2, ylim = c(0,1))
lines(x = 1:143, y = SurvPrevStroke, col = "red", lwd = 2)
legend("topright", legend = c("Without diabetes", "Diabetes"),
lty = 1, lwd =2, col = c("black", "red"))
```

---

# intPredErr *Integrated prediction error*

---

## Description

Computes the integrated prediction error curve for discrete survival models.

## Usage

```
intPredErr(
  hazards,
  testTime,
  testEvent,
  trainTime,
  trainEvent,
  testDataLong,
  tmax = NULL
)
```

## Arguments

| | |
|---|---|
| hazards | Predicted discrete hazards in the test data ("numeric vector"). |
| testTime | Discrete time intervals in short format of the test set ("integer vector"). |
| testEvent | Events in short format in the test set ("binary vector"). |
| trainTime | Discrete time intervals in short format of the training data set ("integer vector"). |
| trainEvent | Events in short format in the training set ("binary vector"). |
| testDataLong | Test data in long format("class data.frame"). The discrete survival function is calculated based on the predicted hazards. It is assumed that the data was preprocessed with a function with prefix "dataLong", see e. g. dataLong, dataLongTimeDep. |
| tmax | Gives the maximum time interval for which prediction errors are calculated ("integer vector"). It must be smaller than the maximum observed time in the training data of the object produced by function. The default setting NULL means, that all observed intervals are used. |

## Value

Integrated prediction error ("numeric vector").

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

Gneiting T, Raftery AE (2007). "Strictly Proper Scoring Rules, Prediction, and Estimation." *Journal of the American Statistical Association*, **102**, 359-378.

## See Also

predErrCurve, aggregate

## Examples

```
##########################
# Example with cancer data

library(survival)
head(cancer)

# Data preparation and convertion to 30 intervals
cancerPrep <- cancer
cancerPrep$status <- cancerPrep$status-1
intLim <- quantile(cancerPrep$time, prob = seq(0, 1, length.out = 30))
intLim [length(intLim)] <- intLim [length(intLim)] + 1

# Cut discrete time in smaller number of intervals
cancerPrep <- contToDisc(dataShort = cancerPrep, timeColumn = "time", intervalLimits = intLim)

# Generate training and test data
set.seed(753)
TrainIndices <- sample (x = 1:dim(cancerPrep) [1], size = dim(cancerPrep) [1] * 0.75)
TrainCancer <- cancerPrep [TrainIndices, ]
TestCancer <- cancerPrep [-TrainIndices, ]
TrainCancer$timeDisc <- as.numeric(as.character(TrainCancer$timeDisc))
TestCancer$timeDisc <- as.numeric(as.character(TestCancer$timeDisc))

# Convert to long format
LongTrain <- dataLong(dataShort = TrainCancer, timeColumn = "timeDisc", eventColumn = "status",
timeAsFactor=FALSE)
LongTest <- dataLong(dataShort = TestCancer, timeColumn = "timeDisc", eventColumn = "status",
timeAsFactor=FALSE)
# Convert factors
LongTrain$timeInt <- as.numeric(as.character(LongTrain$timeInt))
LongTest$timeInt <- as.numeric(as.character(LongTest$timeInt))
LongTrain$sex <- factor(LongTrain$sex)
LongTest$sex <- factor(LongTest$sex)

# Estimate, for example, a generalized, additive model in discrete survival analysis
library(mgcv)
gamFit <- gam (formula = y ~ s(timeInt) + s(age) + sex + ph.ecog, data = LongTrain,
family = binomial())
summary(gamFit)

# 1. Specification of predicted discrete hazards
# Estimate survival function of each person in the test data
testPredHaz <- predict(gamFit, newdata = LongTest, type = "response")

# 2. Calculate integrated prediction error
intPredErr(hazards = testPredHaz,
testTime = TestCancer$timeDisc, testEvent = TestCancer$status,
trainTime = TrainCancer$timeDisc, trainEvent = TrainCancer$status,
testDataLong = LongTest)
```

`intPredErrCompRisks` *Integrated Prediction Error for Competing Risks*

## Description

Estimates integrated prediction errors of arbitrary prediction models in the case of competing risks.

## Usage

```
intPredErrCompRisks(
  testPreds,
  testDataShort,
  trainDataShort,
  timeColumn,
  eventColumns,
  tmax = NULL
)
```

## Arguments

| | |
|---|---|
| `testPreds` | Predictions on the test data with model fitted on training data. Must be a matrix, with the predictions in the rows and the number of columns equal to the number of events. |
| `testDataShort` | Test data in short format. |
| `trainDataShort` | Train data in short format. |
| `timeColumn` | Character giving the column name of the observed times. |
| `eventColumns` | Character vector giving the column names of the event indicators (excluding censoring column). |
| `tmax` | Gives the maximum time interval for which prediction errors are calculated. It must not be higher than the maximum observed time in the training data. |

## Value

Integrated prediction errors for each competing event. Matrix, with the integrated predictions in the rows and the number of columns equal to the number of events.

## Author(s)

Moritz Berger <moritz.berger@imbie.uni-bonn.de>
https://www.imbie.uni-bonn.de/personen/dr-moritz-berger/

## References

Heyard R, Timsit J, Held L, COMBACTE-MAGNET,consortium (2019). "Validation of discrete time-to-event prediction models in the presence of competing risks." *Biometrical Journal*, **62**, 643-657.

## See Also

predErrCompRisks, predErrCurve

## Examples

```
############################
# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
selectInd1 <- 1:200
selectInd2 <- 201:400
trainSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd1], ]
testSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd2], ]

# Convert to long format
trainSet_long <- dataLongCompRisks(dataShort=trainSet, timeColumn="spell",
eventColumns=c("censor1", "censor4"), timeAsFactor=TRUE)
tmax          <- max(trainSet$spell)
testSet_long <- dataLongCompRisks(dataShort=testSet, timeColumn="spell",
eventColumns=c("censor1", "censor4"), aggTimeFormat = TRUE, lastTheoInt=tmax,
timeAsFactor=TRUE)

# Estimate continuation ratio model with logit link
vglmFit <- VGAM::vglm(formula=cbind(e0, e1, e2) ~ timeInt + age + logwage,
data=trainSet_long, family=VGAM::multinomial(refLevel="e0"))

# Calculate predicted hazards
predHazards <- VGAM::predictvglm(vglmFit, newdata=testSet_long, type="response")

# Compute integrated prediction error
intPredErrCompRisks(testPreds=predHazards[,-1], testSet, trainSet, "spell",
c("censor1", "censor4"), tmax)
```

---

# lifeTable *Life Table Construction and Estimates*

---

## Description

Constructs a life table and estimates discrete hazards, survival functions, discrete cumulative hazards and their standard errors without covariates.

## Usage

```
lifeTable(dataShort, timeColumn, eventColumn, intervalLimits = NULL)
```

```
## S3 method for class 'discSurvLifeTable'
print(x, ...)
```

## Arguments

| | |
|---|---|
| dataShort | Original data in short format ("class data.frame"). |
| timeColumn | Name of the column with discrete survival times ("character vector"). |
| eventColumn | Gives the column name of the event indicator (1=observed, 0=censored) ("character vector"). |
| intervalLimits | Optional names of the intervals for each row, e. g. [a_0, a_1), [a_1, a_2), ..., [a_q-1, a_q) ("character vector") |
| x | Object of class "discSurvLifeTable"("class discSurvLifeTable") |
| ... | Additional arguments to the print function |

## Value

List containing an object of class "data.frame" with following columns

- n Number of individuals at risk in a given time interval (integer)
- events Observed number of events in a given time interval (integer)
- dropouts Observed number of dropouts in a given time interval (integer)
- atRisk Estimated number of individuals at risk, corrected by dropouts (numeric)
- hazard Estimated risk of death (without covariates) in a given time interval
- seHazard Estimated standard deviation of estimated hazard
- S Estimated survival curve
- seS Estimated standard deviation of estimated survival function
- cumHazard Estimated cumulative hazard function
- seCumHazard Estimated standard deviation of the estimated cumulative hazard function
- margProb Estimated marginal probability of event in time interval

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

Matthias Schmid <matthias.schmid@imbie.uni-bonn.de>

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

Lawless JF (2002). *Statistical Models and Methods for Lifetime Data, 2nd edition*. Wiley series in probability and statistics.

## Examples

```
# Example with unemployment data
library(Ecdat)
data(UnempDur)

# Extract subset of all persons smaller or equal the median of age
UnempDurSubset <- subset(UnempDur, age <= median(UnempDur$age))
LifeTabUnempDur <- lifeTable(dataShort = UnempDurSubset, timeColumn = "spell",
eventColumn = "censor1")
LifeTabUnempDur

# Example with monoclonal gammapothy data
library(survival)
head(mgus)

# Extract subset of mgus
subMgus <- mgus [mgus$futime<=median(mgus$futime), ]

# Transform time in days to intervals [0, 1), [1, 2), [2, 3), ... , [12460, 12461)
mgusInt <- subMgus
mgusInt$futime <- mgusInt$futime + 1
LifeTabGamma <- lifeTable(dataShort = mgusInt, timeColumn= "futime", eventColumn = "death")
head(LifeTabGamma$Output, 25)
plot(x = 1:dim(LifeTabGamma$Output)[1], y = LifeTabGamma$Output$hazard, type = "l",
xlab = "Time interval", ylab = "Hazard", las = 1,
main = "Life table estimated marginal discrete hazards")
```

---

# minNodePruning *Minimal Node Size Pruning*

---

## Description

Computes optimal minimal node size of a discrete survival tree from a given vector of possible node sizes by cross-validation. Laplace-smoothing can be applied to the estimated hazards.

## Usage

```
minNodePruning(
  formula,
  data,
  treetype = "rpart",
  splitruleranger = "hellinger",
  sizes,
  indexList,
  timeColumn,
  eventColumn,
  lambda = 1,
  logOut = FALSE
)
```

## Arguments

| | |
|---|---|
| formula | Model formula for tree fitting("class formula") |
| data | Discrete survival data in short format for which a survival tree is to be fitted ("class data.frame"). |
| treetype | Type of tree to be fitted ("character vector"). Possible values are "rpart" or "ranger". The default is to fit an rpart tree; when "ranger" is chosen, a ranger forest with a single tree is fitted. |
| splitruleranger | String specifying the splitting rule of the ranger tree("character vector"). Possible values are either "gini", "extratrees" or "hellinger". Default is "hellinger". |
| sizes | Vector of different node sizes to try ("integer vector"). Values should be non-negative. |
| indexList | List of data partitioning indices for cross-validation ("class list"). Each element represents the test indices of one fold ("integer vector"). |
| timeColumn | Character giving the column name of the observed times in the *data* argument ("character vector"). |
| eventColumn | Character giving the column name of the event indicator in the *data* argument ("character vector"). |
| lambda | Parameter for laplace-smoothing. A value of 0 corresponds to no laplace-smoothing ("numeric vector"). |
| logOut | Logical value ("logical vector"). If the argument is set to TRUE, then computation progress will be written to console. |

## Details

Computes the out-of-sample log likelihood for all data partitionings for each node size in *sizes* and returns the node size for which the log likelihood was minimal. Also returns an rpart tree with the optimal minimal node size using the entire data set.

## Value

A list containing the two items

- Optimal minimal node size - Node size with lowest out-of-sample log-likelihood
- tree - a tree object with type corresponding to *treetype* argument with the optimal minimal node size

## Examples

```
library(pec)
library(caret)
data(cost)
# Take subsample and convert time to years
cost$time <- ceiling(cost$time / 365)
costSub <- cost[1:50, ]
# Specify column names for data augmentation
timeColumn <- "time"
```

```
eventColumn <- "status"
# Create data partition for cross validation
indexList <- createFolds(costSub$status * max(costSub$time) + costSub$time, k = 5)
# specify function arguments and perform node size pruning
formula <- y ~ timeInt + prevStroke + age + sex
sizes <- 1:10
optiTree <- minNodePruning(formula, costSub, treetype = "rpart", sizes = sizes,
indexList = indexList, timeColumn =  timeColumn, eventColumn = eventColumn,
lambda = 1, logOut = TRUE)
```

---

`minNodePruningCompRisks` *Minimal Node Size Pruning in the Presence of Competing Risks*

---

## Description

Computes optimal minimal node size of a discrete survival tree from a given vector of possible node sizes by cross-validation. Laplace-smoothing can be applied to the estimated hazards.

## Usage

```
minNodePruningCompRisks(
  formula,
  data,
  treetype = "rpart",
  splitruleranger = "gini",
  sizes,
  indexList,
  timeColumn,
  eventColumns,
  lambda = 1,
  logOut = FALSE
)
```

## Arguments

| | |
|---|---|
| `formula` | Model formula for tree fitting("class formula") |
| `data` | Discrete survival data in short format for which a survival tree is to be fitted ("class data.frame"). |
| `treetype` | Type of tree to be fitted. Possible values are "rpart" or "ranger" ("character vector"). The default is to fit an rpart tree; when "ranger" is chosen, a ranger forest with a single tree is fitted. |
| `splitruleranger` | String specifying the splitting rule of the ranger tree ("character vector"). Possible values are either "gini" or "extratrees". Default is "gini". |
| `sizes` | Vector of different node sizes to try ("integer vector"). Values need to be non-negative. |

| | |
|---|---|
| indexList | List of data partitioning indices for cross-validation ("class list"). Each element represents the test indices of one fold ("integer vector"). |
| timeColumn | Character giving the column name of the observed times in the "data"-argument("character vector"). |
| eventColumns | Character vector giving the column names of the event indicators (excluding censoring column) in the "data"-argument("character vector"). |
| lambda | Parameter for laplace-smoothing. A value of 0 corresponds to no laplace-smoothing ("numeric vector"). |
| logOut | Logical value("logical vector"). If True, computation progress will be written to console. |

## Details

Computes the out-of-sample log likelihood for all data partitionings for each node size in *sizes* and returns the node size for which the log likelihood was minimal. Also returns an rpart tree with the optimal minimal node size using the entire data set.

## Value

A list containing the two items

- Optimal minimal node size - Node size with lowest out-of-sample log-likelihood
- tree - a tree object with type corresponding to *treetype* argument with the optimal minimal node size

## Examples

```
# Example unemployment data
library(Ecdat)
library(caret)
data(UnempDur)
# Select training and testing subsample
subUnempDur <- UnempDur[which(UnempDur$spell < 10),]
subUnempDur <- subUnempDur[1:250,]
#creating status variable for data partitioning
subUnempDur$status <- ifelse(subUnempDur$censor1, 1,
ifelse(subUnempDur$censor2, 2, ifelse(
subUnempDur$censor3, 3, ifelse(subUnempDur$censor4, 4, 0))))
indexList <- createFolds(subUnempDur$status*max(subUnempDur$spell) + subUnempDur$spell, k = 5)
# performing minimal node size pruning
formula <- responses ~ timeInt + age + logwage
sizes <- 1:10
timeColumn <- "spell"
eventColumns <- c("censor1", "censor2", "censor3","censor4")
optiTree <- minNodePruningCompRisks(formula, subUnempDur, treetype = "rpart", sizes = sizes,
indexList = indexList, timeColumn = timeColumn, eventColumns = eventColumns, lambda = 1,
logOut = TRUE)
```

plotCumInc *Plot Estimated Cumulative Incidence Function*

## Description

Generates a plot of an estimated cumulative incidence function P(T <= t, event=k | x) based on estimated hazards of a discrete competing risks model or a discrete subdistribution hazard model.

## Usage

```
plotCumInc(hazards, eventFocus = NULL, ...)
```

## Arguments

| | |
|---|---|
| hazards | Numeric matrix (where each column represents one event) or vector of estimated hazards("numeric matrix"). |
| eventFocus | Column that represent the primary event ("integer vector"). Only applicable in the case of competing risks. |
| ... | Further arguments passed to plot. |

## Author(s)

Moritz Berger <moritz.berger@imbie.uni-bonn.de>
https://www.imbie.uni-bonn.de/personen/dr-moritz-berger/

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

## See Also

estSurv, estCumInz, compRisksGEE

## Examples

```
# Example with unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
SubUnempDur <- UnempDur [1:100, ]

################################
# Competing risks model

# Estimate GEE models for all events
estGEE <- compRisksGEE(datShort = SubUnempDur, dataTransform = "dataLongCompRisks",
corstr = "independence", formulaVariable =~ timeInt + age + ui + logwage * ui,
```

```
eventColumns = c("censor1", "censor2", "censor3", "censor4"), timeColumn = "spell")

# Estimate hazards of all events given the covariates of third person
SubUnempDurLong <- dataLongCompRisks(dataShort = SubUnempDur,
eventColumns = c("censor1", "censor2", "censor3", "censor4"), timeColumn = "spell")
preds <- predict(estGEE, subset(SubUnempDurLong, obj == 3))

plotCumInc(preds, eventFocus = 3)


##############################
# Subdistribution hazards model

# Convert to long format
SubUnempDurLong <- dataLongSubDist(dataShort = SubUnempDur, timeColumn = "spell",
eventColumns = c("censor1", "censor2", "censor3", "censor4"), eventFocus = "censor1")

# Estimate continuation ratio model with logit link
glmFit <- glm(formula = y ~ timeInt + age + ui + logwage * ui, data = SubUnempDurLong,
family = binomial(), weights = SubUnempDurLong$subDistWeights)

# Estimated subdistribution hazard given the covariates of the third person
preds <- predict(glmFit, type = "response", newdata = subset(SubUnempDurLong, obj == 3))

plotCumInc(preds)
```

---

`plotSurv` *Plot Estimated Survival Function*

---

## Description

Generates a plot of an estimated survival function S(T>t|x) based on estimated discrete hazards.

## Usage

```
plotSurv(hazards, ...)
```

## Arguments

| | |
|---|---|
| `hazards` | Estimated discrete hazards ("numeric vector") |
| `...` | Further arguments passed to `plot`. |

## Author(s)

Moritz Berger <moritz.berger@imbie.uni-bonn.de>
https://www.imbie.uni-bonn.de/personen/dr-moritz-berger/

## References

Tutz G, Schmid M (2016). *Modeling discrete time-to-event data*. Springer Series in Statistics.

## See Also

estSurv

## Examples

```
# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
subUnempDur <- UnempDur [1:100, ]

# Convert to long format
UnempLong <- dataLong(dataShort = subUnempDur, timeColumn = "spell", eventColumn = "censor1")
head(UnempLong)

# Estimate binomial model with logit link
Fit <- glm(formula = y ~ timeInt + age + logwage, data = UnempLong, family = binomial())

# Estimate discrete survival function given age, logwage of first person
Tmax   <- max(subUnempDur$spell)
UnempEval <- dataLong(dataShort = UnempDur[1,], timeColumn = "spell", eventColumn = "censor1",
aggTimeFormat = TRUE, lastTheoInt = Tmax)
hazard <- predict(Fit, newdata = UnempEval, type = "response")

plotSurv(hazard)
```

---

# predErrCompRisks *Prediction Error Curves for Competing Risks*

---

## Description

Estimates prediction error curves for discrete survival competing risks models

## Usage

```
predErrCompRisks(
  testPreds,
  testDataShort,
  trainDataShort,
  timeColumn,
  eventColumns,
```

```
  tmax = NULL
)
```

## Arguments

| | |
|---|---|
| testPreds | Predictions on the test data with model fitted on training data ("numeric matrix"). Predictions are stored in the rows and the number of columns equal the number of events. |
| testDataShort | Test data in short format ("class data.frame"). |
| trainDataShort | Train data in short format ("class data.frame"). |
| timeColumn | Character giving the column name of the observed times("character vector"). |
| eventColumns | Character vector giving the column names of the event indicators (excluding censoring column) ("character vector"). |
| tmax | Gives the maximum time interval for which prediction errors are calculated ("integer vector"). It must not be higher than the maximum observed time in the training data. |

## Value

Calculated prediction errors for each competing event. Array with one matrix per competing event, with the predictions in the rows and the time points in the columns.

## Author(s)

Moritz Berger <moritz.berger@imbie.uni-bonn.de>
https://www.imbie.uni-bonn.de/personen/dr-moritz-berger/

## References

Heyard R, Timsit J, Held L, COMBACTE-MAGNET,consortium (2019). "Validation of discrete time-to-event prediction models in the presence of competing risks." *Biometrical Journal*, **62**, 643-657.

## See Also

intPredErrCompRisks, predErrCurve

## Examples

```
############################
# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
selectInd1 <- 1:200
selectInd2 <- 201:400
trainSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd1], ]
testSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd2], ]
```

```
# Convert to long format
trainSet_long <- dataLongCompRisks(dataShort=trainSet, timeColumn="spell",
eventColumns=c("censor1", "censor4"), timeAsFactor=TRUE)
tmax          <- max(trainSet$spell)
testSet_long <- dataLongCompRisks(dataShort=testSet, timeColumn="spell",
eventColumns=c("censor1", "censor4"), aggTimeFormat = TRUE, lastTheoInt=tmax,
timeAsFactor=TRUE)

# Estimate continuation ratio model with logit link
vglmFit <- VGAM::vglm(formula=cbind(e0, e1, e2) ~ timeInt + age + logwage,
data=trainSet_long, family=VGAM::multinomial(refLevel="e0"))

# Calculate predicted hazards
predHazards <- VGAM::predictvglm(vglmFit, newdata=testSet_long, type="response")

# Compute prediction error
predErrCompRisks(testPreds=predHazards[,-1], testSet, trainSet, "spell",
c("censor1", "censor4"), tmax)
```

---

`print.discSurvPredErrDisc` *Prediction Error Curves*

---

## Description

Estimates prediction error curves of arbitrary discrete survival prediction models. In prediction error curves the estimated and observed survival functions are compared adjusted by weights at given timepoints.

## Usage

```
## S3 method for class 'discSurvPredErrDisc'
print(x, ...)

## S3 method for class 'discSurvPredErrDisc'
plot(x, ...)

predErrCurve(
  timepoints,
  estSurvList,
  testTime,
  testEvent,
  trainTime,
  trainEvent
)
```

## Arguments

| | |
|---|---|
| x | Object of class "discSurvPredErrDisc"("class discSurvPredErrDisc") |
| ... | Additional arguments to S3 methods. |
| timepoints | Vector of the number of discrete time intervals ("integer vector"). |
| estSurvList | List of persons in the test data ("class list"). Each element contains a estimated survival functions of all given time points ("numeric vector"). |
| testTime | Discrete survival times in the test data ("numeric vector"). |
| testEvent | Univariate event indicator in the test data ("binary vector"). |
| trainTime | Numeric vector of discrete survival times in the training data ("numeric vector"). |
| trainEvent | Integer vector of univariate event indicator in the training data("integer vector"). |

## Details

The prediction error curves should be smaller than 0.25 for all time points, because this is equivalent to a random assignment error.

## Value

- List List with objects:
  - Output List with two components
    - predErr Numeric vector with estimated prediction error values. Names give the evaluation time point.
    - weights List of weights used in the estimation. Each list component gives the weights of a person in the test data.
  - Input A list of given argument input values (saved for reference)

## Author(s)

Thomas Welchowski <welchow@imbie.meb.uni-bonn.de>

## References

Van der Laan MJ, Robins JM (2003). *Unified Methods for Censored Longitudinal Data and Causality*. Springer Series in Statistics.

Gerds TA, Schumacher M (2006). "Consistent estimation of the expected Brier Score in general survival models with right-censored event times." *Biometrical Journal*, **48**, 1029-1040.

## See Also

gam

## Examples

```
# Example with cross validation and unemployment data
library(Ecdat)
library(mgcv)
data(UnempDur)
summary(UnempDur$spell)

# Extract subset of data
set.seed(635)
IDsample <- sample(1:dim(UnempDur)[1], 100)
UnempDurSubset <- UnempDur [IDsample, ]
head(UnempDurSubset)
range(UnempDurSubset$spell)

# Generate training and test data
set.seed(7550)
TrainIndices <- sample (x = 1:dim(UnempDurSubset) [1], size = 75)
TrainUnempDur <- UnempDurSubset [TrainIndices, ]
TestUnempDur <- UnempDurSubset [-TrainIndices, ]

# Convert to long format
LongTrain <- dataLong(dataShort = TrainUnempDur, timeColumn = "spell", eventColumn = "censor1")
LongTest <- dataLong(dataShort = TestUnempDur, timeColumn = "spell", eventColumn = "censor1")
# Convert factor to numeric for smoothing
LongTrain$timeInt <- as.numeric(as.character(LongTrain$timeInt))
LongTest$timeInt <- as.numeric(as.character(LongTest$timeInt))

#########################################################################
# Estimate a generalized, additive model in discrete survival analysis

gamFit <- gam (formula = y ~ s(timeInt) + age + logwage, data = LongTrain, family = binomial())

# Estimate survival function of each person in the test data
oneMinusPredHaz <- 1 - predict(gamFit, newdata = LongTest, type = "response")
predSurv <- aggregate(oneMinusPredHaz ~ obj, data = LongTest, FUN = cumprod)

# Prediction error in first interval
tryPredErrDisc1 <- predErrCurve (timepoints = 1,
estSurvList = predSurv [[2]], testTime = TestUnempDur$spell,
testEvent=TestUnempDur$censor1, trainTime = TrainUnempDur$spell,
 trainEvent=TrainUnempDur$censor1)
tryPredErrDisc1

# Prediction error of the 2. to 10. interval
tryPredErrDisc2 <- predErrCurve (timepoints = 2:10,
estSurvList = predSurv [[2]], testTime = TestUnempDur$spell,
testEvent = TestUnempDur$censor1, trainTime = TrainUnempDur$spell,
trainEvent = TrainUnempDur$censor1)
tryPredErrDisc2
plot(tryPredErrDisc2)

########################################
```

```
# Fit a random discrete survival forest

library(ranger)
LongTrainRF <- LongTrain
LongTrainRF$y <- factor(LongTrainRF$y)
rfFit <- ranger(formula = y ~ timeInt + age + logwage, data = LongTrainRF,
probability = TRUE)

# Estimate survival function of each person in the test data
oneMinusPredHaz <- 1 - predict(rfFit, data = LongTest)$predictions[, 2]
predSurv <- aggregate(oneMinusPredHaz ~ obj, data = LongTest, FUN = cumprod)

# Prediction error in first interval
tryPredErrDisc1 <- predErrCurve (timepoints = 1,
estSurvList = predSurv [[2]], testTime = TestUnempDur$spell,
testEvent = TestUnempDur$censor1, trainTime = TrainUnempDur$spell,
 trainEvent = TrainUnempDur$censor1)
tryPredErrDisc1

# Prediction error of the 2. to 10. interval
tryPredErrDisc2 <- predErrCurve (timepoints = 2:10,
estSurvList = predSurv [[2]], testTime = TestUnempDur$spell,
testEvent = TestUnempDur$censor1, trainTime = TrainUnempDur$spell,
trainEvent = TrainUnempDur$censor1)
tryPredErrDisc2
plot(tryPredErrDisc2)
```

---

`survTreeLaplaceHazard` *Laplace Hazards for a Competing Risk Survival Tree Object*

---

## Description

Predicts the laplace-smoothed hazards of discrete survival tree. Can be used for single-risk or competing risk discrete survival data.

## Usage

```
survTreeLaplaceHazard(treeModel, newdata, lambda)
```

## Arguments

| | |
|---|---|
| `treeModel` | Fitted tree object as generated by "rpart" ("class rpart"). |
| `newdata` | Data in long format for which hazards are to be computed. Must contain the same columns that were used for tree fitting("class data.frame"). |
| `lambda` | Smoothing parameter for laplace-smoothing. Must be a non-negative number. A value of 0 corresponds to no smoothing ("numeric vector"). |

## Value

A m by k matrix with m being the length of newdata and k being the number of classes in treeModel. Each row corresponds to the smoothed hazard of the respective observation.

## Examples

```
library(pec)
library(caret)
# Example data
data(cost)
# Convert time to years and select training and testing subsample
cost$time <- ceiling(cost$time/365)
costTrain <- cost[1:100, ]
costTest  <- cost[101:120, ]
# Convert to long format
timeColumn <- "time"
eventColumn <- "status"
costTrainLong <- dataLong(dataShort=costTrain, timeColumn = "time",
                          eventColumn = "status")
costTestLong  <- dataLong(dataShort=costTest, timeColumn = "time",
                          eventColumn = "status")
head(costTrainLong)
# Fit a survival tree
costTree <- rpart(formula = y ~ timeInt + prevStroke + age + sex, data = costTrainLong,
                  method = "class")
# Compute smoothed hazards for test data
predictedhazards <- survTreeLaplaceHazard(costTree, costTestLong, 1)
predictedhazards
```

---

# survTreeLaplaceHazardRanger

*Laplace Hazards for a Competing Risk Survival Tree Object*

---

## Description

Predicts the laplace-smoothed hazards of discrete survival data based on a survival tree from class "ranger". Currently only single-risk data is supported.

## Usage

```
survTreeLaplaceHazardRanger(treeModel, rangerdata, newdata, lambda)
```

## Arguments

| | |
|---|---|
| `treeModel` | Fitted tree object as generated by "ranger" ("class data.frame"). Must be a single ranger tree. |
| `rangerdata` | Original training data with which *treeModel* was fitted ("class data.frame"). Must be in long format. |

| | |
|---|---|
| newdata | Data in long format for which hazards are to be computed ("class data.frame"). Must contain the same columns that were used for tree fitting. |
| lambda | Smoothing parameter for laplace-smoothing ("class data.frame"). Must be a non-negative number. A value of zero corresponds to no smoothing. |

## Value

A m by k matrix with m being the length of newdata and k being the number of classes in *treeModel*. Each row corresponds to the smoothed hazard of the respective observation.

## Examples

```
library(pec)
library(caret)
library(ranger)
data(cost)
# Take subsample and convert time to years
cost$time <- ceiling(cost$time/365)
costSubTrain <- cost[1:50,]
costSubTest <- cost[51:70,]
# Specify column names for data augmentation
timeColumn<-"time"
eventColumn<-"status"
costSubTrainLong <- dataLong(costSubTrain, timeColumn, eventColumn)
costSubTestLong <- dataLong(costSubTest, timeColumn, eventColumn)
#create tree
formula <- y ~ timeInt + diabetes + prevStroke + age + sex
rangerTree <- ranger(formula, costSubTrainLong, num.trees = 1, mtry = 5,
classification = TRUE, splitrule = "hellinger", replace = FALSE,
sample.fraction = 1, max.depth = 5)
#compute laplace-smoothed hazards
laplHaz <- survTreeLaplaceHazardRanger(rangerTree, costSubTrainLong,
costSubTestLong, lambda = 1)
laplHaz
```

---

# unempMultiSpell *Multiple Spell employment data*

---

## Description

Subsample of 1000 persons from the national longitudinal survey of youth 1979 data. Included covariates are age, children, ethnicity, marital status and sex. The bivariate responses current state (spell) and discrete time interval (year) are the last two columns.

## Usage

```
data(unempMultiSpell)
```

## Details

- Column "id" is defined as identification number for each person.
- Column "age" represents the time-varying age of each person in years.
- Column "child" consists of values
  - 0 - No children
  - 1 - Individual has child/children
- Column "ethnicity" consists of values
  - 1 - Hispanic
  - 2 - Black
  - 3 - Other
- Column "marriage" consists of values
  - 1 - Never Married
  - 2 - Currently married
  - 3 - Other/Divorced
- Column "sex" consists of values
  - 1 - Male
  - 2 - Female
- Column "spell" represents the time-varying employment status of each person. Possible values are
  - 1 - Employed
  - 2 - Unemployed
  - 3 - Out of labor force
  - 4 - In active forces
  - 0 - Censored
- Column "year" represents the discrete time intervals in years.

## Author(s)

David Koehler <koehler@imbie.uni-bonn.de>

## Source

National Longitudinal Survey of Youth

`weightsLtoT` *Compute Subdistribution Weights*

## Description

Function to compute new subdistribution weights for a test data set based on the estimated censoring survival function from a learning data set

## Usage

```
weightsLtoT(
  dataShortTrain,
  dataShortTest,
  timeColumn,
  eventColumns,
  eventFocus
)
```

## Arguments

| | |
|---|---|
| `dataShortTrain` | Learning data in short format ("class data.frame"). |
| `dataShortTest` | Test data in short format ("class data.frame"). |
| `timeColumn` | Character specifying the column name of the observed event times ("character vector"). It is required that the observed times are discrete ("integer vector"). |
| `eventColumns` | Character vector specifying the column names of the event indicators ("logical vector")(excluding censoring events). It is required that a 0-1 coding is used for all events. The algorithm treats row sums of zero of all event columns as censored. |
| `eventFocus` | Column name of the event of interest, which corresponds to the type 1 event ("character vector"). |

## Value

Subdstribution weights for the test data in long format using the estimated censoring survival function from the learning data ("numeric vector"). The length of the vector is equal to the number of observations of the long test data.

## Author(s)

Moritz Berger <moritz.berger@imbie.uni-bonn.de>
https://www.imbie.uni-bonn.de/personen/dr-moritz-berger/

## References

Berger M, Schmid M, Welchowski T, Schmitz-Valckenberg S, Beyersmann J (2020). "Subdistribution Hazard Models for Competing Risks in Discrete Time." *Biostatistics*, **21**, 449-466.

## See Also

dataLongSubDist, calibrationPlot

## Examples

```
####################
# Data preprocessing

# Example unemployment data
library(Ecdat)
data(UnempDur)

# Select subsample
selectInd1 <- 1:100
selectInd2 <- 101:200
trainSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd1], ]
valSet <- UnempDur[which(UnempDur$spell %in% (1:10))[selectInd2], ]

# Convert to long format
trainSet_long <- dataLongSubDist(dataShort = trainSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"), eventFocus = "censor1")
valSet_long <- dataLongSubDist(dataShort = valSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"), eventFocus = "censor1")

# Compute new weights of the validation data set
valSet_long$subDistWeights <- weightsLtoT(trainSet, valSet, timeColumn = "spell",
eventColumns = c("censor1", "censor4"), eventFocus = "censor1")

# Estimate continuation ratio model with logit link
glmFit <- glm(formula = y ~ timeInt + age + logwage, data = trainSet_long,
family = binomial(), weights = trainSet_long$subDistWeights)

# Calculate predicted discrete hazards
predHazards <- predict(glmFit, newdata = valSet_long, type = "response")

# Calibration plot
calibrationPlot(predHazards, testDataLong = valSet_long, weights = valSet_long$subDistWeights)
```

# Index