Package ‘survcompare’

October 5, 2024

Title Nested Cross-Validation to Compare Cox-PH, Cox-Lasso, Survival Random Forests

Version 0.2.0

Date 2024-10-02

Description Performs repeated nested cross-validation for Cox Proportionate Hazards, Cox Lasso, Survival Random Forest, and their ensemble. Returns internally validated concordance index, time-dependent area under the curve, Brier score, calibration slope, and statistical testing of non-linear ensemble outperforming the baseline Cox model. In this, it helps researchers to quantify the gain of using a more complex survival model, or justify its redundancy. Equally, it shows the performance value of the non-linear and interaction terms, and may highlight the need of further feature transformation. Further details can be found in Shamsutdinova, Stamate, Roberts, & Stahl (2022) ``Combining Cox Model and Tree-Based Algorithms to Boost Performance and Preserve Interpretability for Health Outcomes'' <doi:10.1007/978-3-031-08337-2_15>, where the method is described as Ensemble 1.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.1), survival (>= 3.0)

Imports stats, timeROC, caret, glmnet, randomForestSRC

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Maintainer Diana Shamsutdinova <diana.shamsutdinova.github@gmail.com>

Config/testthat/edition 3

NeedsCompilation no

Author Diana Shamsutdinova [aut, cre] (<https://orcid.org/0000-0003-2434-3641>), Daniel Stahl [aut] (<https://orcid.org/0000-0001-7987-6619>)

Repository CRAN

Date/Publication 2024-10-05 17:00:01 UTC

Contents

`linear_beta` *Auxiliary function for simulatedata functions*

Description

Auxiliary function for simulatedata functions

Usage

```
linear_beta(df)
```

Arguments

`df`	data

`ml_hyperparams_srf` *Internal function for getting grid of hyperparameters for random or grid search of size = max_grid_size*

Description

Internal function for getting grid of hyperparameters for random or grid search of size = max_grid_size

Usage

```
ml_hyperparams_srf(
  mlparams = list(),
  p = 10,
  max_grid_size = 10,
  dftune_size = 1000,
  randomseed = NaN
)
```

Arguments

`mlparams`	list of params
`p`	number of predictors to detine mtry options
`max_grid_size`	grid size for tuning
`dftune_size`	size of the tuning data to define nodesize options
`randomseed`	randomseed to select the tuning grid

`print.survcompare` *Print survcompare object*

Description

Print survcompare object

Usage

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

Arguments

`x`	output object of the survcompare function
`...`	additional arguments to be passed

Value

x

print.survensemble_cv *Prints trained survensemble object*

Description

Prints trained survensemble object

Prints survensemble_cv object

Usage

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

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

Arguments

x	survensemble_cv object
...	additional arguments to be passed

Value

x

x

simulate_crossterms *Simulated sample with survival outcomes with non-linear and cross-term dependencies*

Description

Simulated sample with exponentially or Weibull distributed time-to-event; log-hazard depends non-linearly on risk factors, and includes cross-terms.

Usage

```
simulate_crossterms(
  N = 300,
  observe_time = 10,
  percentcensored = 0.75,
  randomseed = NULL,
  lambda = 0.1,
  distr = "Exp",
  rho_w = 1,
  drop_out = 0.3
)
```

Arguments

N	sample size, 300 by default
observe_time	study's observation time, 10 by default
percentcensored	expected number of non-events by observe_time, 0.75 by default (i.e. event rate is 0.25)
randomseed	random seed for replication
lambda	baseline hazard rate, 0.1 by default
distr	time-to-event distribution, "Exp" for exponential (default), "W" for Weibull
rho_w	shape parameter for Weibull distribution, 0.3 by default
drop_out	expected rate of drop out before observe_time, 0.3 by default

Value

data frame; "time" and "event" columns describe survival outcome; predictors are "age", "sex", "hyp", "bmi"

Examples

```
mydata <- simulate_crossterms()
head(mydata)
```

---

simulate_linear *Simulated sample with survival outcomes with linear dependencies*

Description

Simulated sample with exponentially or Weibull distributed time-to-event; log-hazard (lambda parameter) depends linearly on risk factors.

Usage

```
simulate_linear(
  N = 300,
  observe_time = 10,
  percentcensored = 0.75,
  randomseed = NULL,
  lambda = 0.1,
  distr = "Exp",
  rho_w = 1,
  drop_out = 0.3
)
```

Arguments

`N`	sample size, 300 by default
`observe_time`	study's observation time, 10 by default
`percentcensored`	expected number of non-events by observe_time, 0.75 by default (i.e. event rate is 0.25)
`randomseed`	random seed for replication
`lambda`	baseline hazard rate, 0.1 by default
`distr`	time-to-event distribution, "Exp" for exponential (default), "W" for Weibull
`rho_w`	shape parameter for Weibull distribution, 0.3 by default
`drop_out`	expected rate of drop out before observe_time, 0.3 by default

Value

data frame; "time" and "event" columns describe survival outcome; predictors are "age", "sex", "hyp", "bmi"

Examples

```
mydata <- simulate_linear()
head(mydata)
```

`simulate_nonlinear` *Simulated sample with survival outcomes with non-linear dependencies*

Description

Simulated sample with exponentially or Weibull distributed time-to-event; log-hazard (lambda parameter) depends non-linearly on risk factors.

Usage

```
simulate_nonlinear(
  N = 300,
  observe_time = 10,
  percentcensored = 0.75,
  randomseed = NULL,
  lambda = 0.1,
  distr = "Exp",
  rho_w = 1,
  drop_out = 0.3
)
```

Arguments

`N`	sample size, 300 by default
`observe_time`	study's observation time, 10 by default
`percentcensored`	expected number of non-events by observe_time, 0.75 by default (i.e. event rate is 0.25)
`randomseed`	random seed for replication
`lambda`	baseline hazard rate, 0.1 by default
`distr`	time-to-event distribution, "Exp" for exponential (default), "W" for Weibull
`rho_w`	shape parameter for Weibull distribution, 0.3 by default
`drop_out`	expected rate of drop out before observe_time, 0.3 by default

Value

data frame; "time" and "event" columns describe survival outcome; predictors are "age", "sex", "hyp", "bmi"

Examples

```
mydata <- simulate_nonlinear()
head(mydata)
```

---

summary.survcompare *Summary of survcompare results*

Description

Summary of survcompare results

Usage

```
## S3 method for class 'survcompare'
summary(object, ...)
```

Arguments

`object`	output object of the survcompare function
`...`	additional arguments to be passed

`summary.survensemble_cv` *Prints summary of a trained survensemble_cv object*

Description

Prints summary of a trained survensemble_cv object

Prints a summary of survensemble_cv object

Usage

```
## S3 method for class 'survensemble_cv'
summary(object, ...)

## S3 method for class 'survensemble_cv'
summary(object, ...)
```

Arguments

`object`	survensemble_cv object
`...`	additional arguments to be passed

Value

object

object

`survcompare` *Cross-validates and compares Cox Proportionate Hazards and Survival Random Forest models*

Description

The function performs a repeated nested cross-validation for

1. Cox-PH (survival package, survival::coxph) or Cox-Lasso (glmnet package, glmnet::cox.fit)
2. Survival Random Forest (randomForestSRC::rfsrc), or its ensemble with the Cox model (if use_ensemble =TRUE)

The same random seed for the train/test splits are used for all models to aid fair comparison; and the performance metrics are computed for the tree models including Harrel's c-index, time-dependent AUC-ROC, time-dependent Brier Score, and calibration slope. The statistical significance of the performance differences between Cox-PH and Cox-SRF Ensemble is tested and reported.

The function is designed to help with the model selection by quantifying the loss of predictive performance (if any) if Cox-PH is used instead of a more complex model such as SRF which can capture non-linear and interaction terms, as well as non-proportionate hazards. The difference in performance of the Ensembled Cox and SRF and the baseline Cox-PH can be viewed as quantification of the non-linear and cross-terms contribution to the predictive power of the supplied predictors.

The function is a wrapper for survcompare2(), for comparison of the CoxPH and SRF models, and an alternative way to do the same analysis is to run survcox_cv() and survsrf_cv(), then using survcompare2()

Cross-validates and compares Cox Proportionate Hazards and Survival Random Forest models

Usage

```
survcompare(
  df_train,
  predict_factors,
  fixed_time = NaN,
  randomseed = NaN,
  useCoxLasso = FALSE,
  outer_cv = 3,
  inner_cv = 3,
  tuningparams = list(),
  return_models = FALSE,
  repeat_cv = 2,
  ml = "SRF",
  use_ensemble = FALSE,
  max_grid_size = 10,
  suppresswarn = TRUE
)
```

Arguments

df_train	training data, a data frame with "time" and "event" columns to define the survival outcome
predict_factors	list of column names to be used as predictors
fixed_time	prediction time of interest. If NULL, 0.90th quantile of event times is used
randomseed	random seed for replication
useCoxLasso	TRUE / FALSE, for whether to use regularized version of the Cox model, FALSE is default
outer_cv	k in k-fold CV
inner_cv	k in k-fold CV for internal CV to tune survival random forest hyper-parameters
tuningparams	list of tuning parameters for random forest: 1) NULL for using a default tuning grid, or 2) a list("mtry"=c(...), "nodedepth" = c(...), "nodesize" = c(...))
return_models	TRUE/FALSE to return the trained models; default is FALSE, only performance is returned
repeat_cv	if NULL, runs once, otherwise repeats several times with different random split for CV, reports average of all

`ml`	this is currently for Survival Random Forest only ("SRF")
`use_ensemble`	TRUE/FALSE for whether to train SRF on its own, apart from the CoxPH->SRF ensemble. Default is FALSE as there is not much information in SRF itself compared to the ensembled version.
`max_grid_size`	number of random grid searches for model tuning
`suppresswarn`	TRUE/FALSE, TRUE by default

Value

outcome - cross-validation results for CoxPH, SRF, and an object containing the comparison results

Author(s)

Diana Shamsutdinova <diana.shamsutdinova.github@gmail.com>

Examples

```
df <-simulate_nonlinear(100)
predictors <- names(df)[1:4]
srf_params <- list("mtry" = c(2), "nodedepth"=c(25), "nodesize" =c(15))
mysurvcomp <- survcompare(df, predictors, tuningparams = srf_params, max_grid_size = 1)
summary(mysurvcomp)
```

---

survcompare2 — *Compares two cross-validated models using surv____cv functions of this package.*

Description

#' The two arguments are two cross-validated models, base and alternative, e.g., Cox Proportionate Hazards Model (or Cox LASSO), and Survival Random Forest, or DeepHit (if installed from GitHub, not in CRAN version). Please see examples below.

Both cross-validations should be done with the same random seed, number of repetitions (repeat_cv), outer_cv and inner_cv to ensure the models are compared on the same train/test splits.

Harrel's c-index,time-dependent AUC-ROC, time-dependent Brier Score, and calibration slopes are reported. The statistical significance of the performance differences is tested for the C-indeces.

The function is designed to help with the model selection by quantifying the loss of predictive performance (if any) if "alternative" is used instead of "base."

Usage

```
survcompare2(base, alternative)
```

Arguments

`base`	an object of type "survensemble_cv", for example, outcomes of survcox_cv, survsrf_cv, survsrfens_cv, survsrfstack_cv
`alternative`	an object of type "survensemble_cv", to compare to "base"

Value

outcome = list(data frame with performance results, fitted Cox models, fitted DeespSurv)

Examples

```
df <-simulate_nonlinear(100)
params <- names(df)[1:4]
cv1 <- survcox_cv(df, params, randomseed = 42, repeat_cv =1)
cv2 <- survsrf_cv(df, params, randomseed = 42, repeat_cv = 1)
survcompare2(cv1, cv2)
```

`survcoxlasso_train` *Trains CoxLasso, using cv.glmnet(s="lambda.min")*

Description

Trains CoxLasso, using cv.glmnet(s="lambda.min")

Usage

```
survcoxlasso_train(
  df_train,
  predict.factors,
  inner_cv = 5,
  fixed_time = NaN,
  retrain_cox = FALSE,
  verbose = FALSE
)
```

Arguments

`df_train`	data frame with the data, "time" and "event" should describe survival outcome
`predict.factors`	list of the column names to be used as predictors
`inner_cv`	k in k-fold CV for lambda tuning
`fixed_time`	not used here, for internal use
`retrain_cox`	whether to re-train coxph on non-zero predictors; FALSE by default
`verbose`	TRUE/FALSE prints warnings if no predictors in Lasso

Value

fitted CoxPH object with coefficient of CoxLasso or re-trained CoxPH with non-zero CoxLasso if retrain_cox = FALSE or TRUE

---

survcox_cv *Cross-validates Cox or CoxLasso model*

Description

Cross-validates Cox or CoxLasso model

Usage

```
survcox_cv(
  df,
  predict.factors,
  fixed_time = NaN,
  outer_cv = 3,
  repeat_cv = 2,
  randomseed = NaN,
  return_models = FALSE,
  inner_cv = 3,
  useCoxLasso = FALSE,
  suppresswarn = TRUE
)
```

Arguments

`df`	data frame with the data, "time" and "event" for survival outcome
`predict.factors`	list of predictor names
`fixed_time`	at which performance metrics are computed
`outer_cv`	k in k-fold CV, default 3
`repeat_cv`	if NULL, runs once, otherwise repeats CV
`randomseed`	random seed
`return_models`	TRUE/FALSE, if TRUE returns all CV objects
`inner_cv`	k in the inner loop of k-fold CV, default is 3; only used if CoxLasso is TRUE
`useCoxLasso`	TRUE/FALSE, FALSE by default
`suppresswarn`	TRUE/FALSE, TRUE by default

Value

list of outputs

Examples

```
df <- simulate_nonlinear()
coxph_cv <- survcox_cv(df, names(df)[1:4])
summary(coxph_cv)
```

---

survcox_predict — *Computes event probabilities from a trained cox model*

Description

Computes event probabilities from a trained cox model

Usage

```
survcox_predict(trained_model, newdata, fixed_time, interpolation = "constant")
```

Arguments

`trained_model`	pre-trained cox model of coxph class
`newdata`	data to compute event probabilities for
`fixed_time`	at which event probabilities are computed
`interpolation`	"constant" by default, can also be "linear", for between times interpolation for hazard rates

Value

returns matrix(nrow = length(newdata), ncol = length(fixed_time))

---

survcox_train — *Trains CoxPH using survival package, or trains CoxLasso (cv.glmnet, lambda.min), and then re-trains survival:coxph on non-zero predictors*

Description

Trains CoxPH using survival package, or trains CoxLasso (cv.glmnet, lambda.min), and then re-trains survival:coxph on non-zero predictors

Usage

```
survcox_train(
  df_train,
  predict.factors,
  fixed_time = NaN,
  useCoxLasso = FALSE,
  retrain_cox = FALSE,
  inner_cv = 5
)
```

Arguments

`df_train`	data, "time" and "event" should describe survival outcome
`predict.factors`	list of the column names to be used as predictors
`fixed_time`	target time, NaN by default; needed here only to re-align with other methods
`useCoxLasso`	TRUE or FALSE
`retrain_cox`	if useCoxLasso is TRUE, whether to re-train coxph on non-zero predictors, FALSE by default
`inner_cv`	k in k-fold CV for training lambda for Cox Lasso, only used for useCoxLasso = TRUE

Value

fitted CoxPH or CoxLasso model

---

survival_prob_km — *Calculates survival probability estimated by Kaplan-Meier survival curve Uses polynomial extrapolation in survival function space, using poly(n=3)*

Description

Calculates survival probability estimated by Kaplan-Meier survival curve Uses polynomial extrapolation in survival function space, using poly(n=3)

Usage

```
survival_prob_km(df_km_train, times, estimate_censoring = FALSE)
```

Arguments

`df_km_train`	event probabilities (!not survival)
`times`	times at which survival is estimated
`estimate_censoring`	FALSE by default, if TRUE, event and censoring is reversed (for IPCW calculations)

Value

vector of survival probabilities for time_point survsrfens_cv *Cross-validates predictive performance for SRF Ensemble*

Description

Cross-validates predictive performance for SRF Ensemble

Usage

```
survsrfens_cv(
  df,
  predict.factors,
  fixed_time = NaN,
  outer_cv = 3,
  inner_cv = 3,
  repeat_cv = 2,
  randomseed = NaN,
  return_models = FALSE,
  useCoxLasso = FALSE,
  tuningparams = list(),
  max_grid_size = 10,
  verbose = FALSE,
  suppresswarn = TRUE
)
```

Arguments

`df`	data frame with the data, "time" and "event" for survival outcome
`predict.factors`	list of predictor names
`fixed_time`	at which performance metrics are computed
`outer_cv`	number of folds in outer CV, default 3
`inner_cv`	number of folds for model tuning CV, default 3
`repeat_cv`	number of CV repeats, if NaN, runs once
`randomseed`	random seed
`return_models`	TRUE/FALSE, if TRUE returns all trained models
`useCoxLasso`	TRUE/FALSE, default is FALSE
`tuningparams`	if given, list of hyperparameters, list(mtry=c(), nodedepth=c(),nodesize=c()), otherwise a wide default grid is used
`max_grid_size`	number of random grid searches for model tuning
`verbose`	FALSE(default)/TRUE
`suppresswarn`	TRUE/FALSE, TRUE by default

Value

list of outputs

Examples

```
df <- simulate_nonlinear()
ens_cv <- survsrfens_cv(df, names(df)[1:4])
summary(ens_cv)
```

`survsrfens_predict` *Predicts event probability by a trained sequential ensemble of Survival Random Forest and CoxPH*

Description

Predicts event probability by a trained sequential ensemble of Survival Random Forest and CoxPH

Usage

```
survsrfens_predict(trained_model, newdata, fixed_time, extrapsurvival = TRUE)
```

Arguments

`trained_model`	a trained model, output of survsrfens_train()
`newdata`	new data for which predictions are made
`fixed_time`	time of interest, for which event probabilities are computed
`extrapsurvival`	if probabilities are extrapolated beyond trained times (constant)

Value

vector of predicted event probabilities

`survsrfens_train` *Fits an ensemble of Cox-PH and Survival Random Forest (SRF) with internal CV to tune SRF hyperparameters.*

Description

Details: the function trains Cox model, then adds its out-of-the-box predictions to Survival Random Forest as an additional predictor to mimic stacking procedure used in Machine Learning and reduce over-fitting. #' Cox model is fitted to .9 data to predict the rest .1 for each 1/10s fold; these out-of-the-bag predictions are passed on to SRF

Usage

```
survsrfens_train(
  df_train,
  predict.factors,
  fixed_time = NaN,
  inner_cv = 3,
  randomseed = NaN,
  tuningparams = list(),
  useCoxLasso = FALSE,
  max_grid_size = 10,
  var_importance_calc = FALSE,
  verbose = FALSE
)
```

Arguments

`df_train`	data, "time" and "event" should describe survival outcome
`predict.factors`	list of predictor names
`fixed_time`	time at which performance is maximized
`inner_cv`	number of cross-validation folds for hyperparameters' tuning
`randomseed`	random seed to control tuning including data splits
`tuningparams`	if given, list of hyperparameters, list(mtry=c(), nodedepth=c(),nodesize=c()), otherwise a wide default grid is used
`useCoxLasso`	if CoxLasso is used (TRUE) or not (FALSE, default)
`max_grid_size`	number of random grid searches for model tuning
`var_importance_calc`	if variable importance is computed
`verbose`	FALSE (default)/TRUE

Value

trained object of class survsrf_ens

survsrfstack_cv *Cross-validates stacked ensemble of the CoxPH and Survival Random Forest models*

Description

Cross-validates stacked ensemble of the CoxPH and Survival Random Forest models

Usage

```
survsrfstack_cv(
  df,
  predict.factors,
  fixed_time = NaN,
  outer_cv = 3,
  inner_cv = 3,
  repeat_cv = 2,
  randomseed = NaN,
  return_models = FALSE,
  useCoxLasso = FALSE,
  tuningparams = list(),
  max_grid_size = 10,
  verbose = FALSE,
  suppresswarn = TRUE
)
```

Arguments

df	data, "time" and "event" should describe survival outcome
predict.factors	list of predictor names
fixed_time	time at which performance is maximized
outer_cv	number of cross-validation folds for model validation
inner_cv	number of cross-validation folds for hyperparameters' tuning
repeat_cv	number of CV repeats, if NaN, runs once
randomseed	random seed to control tuning including data splits
return_models	TRUE/FALSE, if TRUE returns all CV objects
useCoxLasso	if CoxLasso is used (TRUE) or not (FALSE, default)
tuningparams	if given, list of hyperparameters, list(mtry=c(), nodedepth=c(),nodesize=c()), otherwise a wide default grid is used
max_grid_size	number of random grid searches for model tuning
verbose	FALSE(default)/TRUE
suppresswarn	TRUE/FALSE, TRUE by default

survsrfstack_predict *Predicts event probability by a trained stacked ensemble of Survival Random Forest and CoxPH*

Description

Predicts event probability by a trained stacked ensemble of Survival Random Forest and CoxPH

Usage

```
survsrfstack_predict(
  trained_object,
  newdata,
  fixed_time,
  predict.factors,
  extrapsurvival = TRUE
)
```

Arguments

`trained_object`	a trained model, output of survsrfstack_train()
`newdata`	new data for which predictions are made
`fixed_time`	time of interest, for which event probabilities are computed
`predict.factors`	list of predictor names
`extrapsurvival`	if probabilities are extrapolated beyond trained times (constant)

Value

vector of predicted event probabilities

survsrfstack_train *Trains the stacked ensemble of the CoxPH and Survival Random Forest*

Description

Trains the stacked ensemble of the CoxPH and Survival Random Forest

Usage

```
survsrfstack_train(
  df_train,
  predict.factors,
  fixed_time = NaN,
  inner_cv = 3,
  randomseed = NaN,
  useCoxLasso = FALSE,
  tuningparams = list(),
  max_grid_size = 10,
  verbose = FALSE
)
```

Arguments

`df_train`	data, "time" and "event" should describe survival outcome
`predict.factors`	list of predictor names
`fixed_time`	time at which performance is maximized
`inner_cv`	number of cross-validation folds for hyperparameters' tuning
`randomseed`	random seed to control tuning including data splits
`useCoxLasso`	if CoxLasso is used (TRUE) or not (FALSE, default)
`tuningparams`	if given, list of hyperparameters, list(mtry=c(), nodedepth=c(),nodesize=c()), otherwise a wide default grid is used
`max_grid_size`	number of random grid searches for model tuning
`verbose`	FALSE(default)/TRUE

Value

output = list(bestparams, allstats, model)

Examples

```
d <-simulate_nonlinear(100)
p<- names(d)[1:4]
tuningparams = list(
 "mtry" = c(5,10,15),
 "nodedepth" = c(5,10,15,20),
 "nodesize" =    c(20,30,50)
)
m_srf<- survsrf_train(d,p,tuningparams=tuningparams)
```

survsrf_cv — *Cross-validates Survival Random Forest*

Description

Cross-validates Survival Random Forest

Usage

```
survsrf_cv(
  df,
  predict.factors,
  fixed_time = NaN,
  outer_cv = 3,
  inner_cv = 3,
  repeat_cv = 2,
  randomseed = NaN,
  return_models = FALSE,
  tuningparams = list(),
  max_grid_size = 10,
  verbose = FALSE,
  suppresswarn = TRUE
)
```

Arguments

`df`	data, "time" and "event" should describe survival outcome
`predict.factors`	list of predictor names
`fixed_time`	time at which performance is maximized
`outer_cv`	number of cross-validation folds for model validation
`inner_cv`	number of cross-validation folds for hyperparameters' tuning
`repeat_cv`	number of CV repeats, if NaN, runs once
`randomseed`	random seed to control tuning including data splits
`return_models`	if all models are stored and returned
`tuningparams`	if given, list of hyperparameters, list(mtry=c(), nodedepth=c(),nodesize=c()), otherwise a wide default grid is used
`max_grid_size`	number of random grid searches for model tuning
`verbose`	FALSE(default)/TRUE
`suppresswarn`	TRUE/FALSE, TRUE by default

Value

list of outputs

Examples

```
df <- simulate_nonlinear()
srf_cv <- survsrf_cv(df, names(df)[1:4])
summary(srf_cv)
```

`survsrf_predict` *Predicts event probability by a trained Survival Random Forest*

Description

Predicts event probability by a trained Survival Random Forest

Usage

```
survsrf_predict(trained_model, newdata, fixed_time, extrapsurvival = TRUE)
```

Arguments

`trained_model`	a trained SRF model, output of survsrf_train(), or randomForestSRC::rfsrc()
`newdata`	new data for which predictions are made
`fixed_time`	time of interest for which event probabilities are computed
`extrapsurvival`	if probabilities are extrapolated beyond trained times (using probability of the lastest available time). Can be helpful for cross-validation of small data, where random split may cause the time of interest being outside of the training set.

Value

vector of predicted event probabilities

`survsrf_train` *Fits randomForestSRC, with tuning by mtry, nodedepth, and nodesize. Underlying model is by Ishwaran et al(2008) https://www.randomforestsrc.org/articles/survival.html Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS. Random survival forests. The Annals of Applied Statistics. 2008;2:841–60.*

Description

Fits randomForestSRC, with tuning by mtry, nodedepth, and nodesize. Underlying model is by Ishwaran et al(2008) https://www.randomforestsrc.org/articles/survival.html Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS. Random survival forests. The Annals of Applied Statistics. 2008;2:841–60.

Usage

```
survsrf_train(
  df_train,
  predict.factors,
  fixed_time = NaN,
  tuningparams = list(),
  max_grid_size = 10,
  inner_cv = 3,
  randomseed = NaN,
  verbose = TRUE
)
```

Arguments

`df_train`	data, "time" and "event" should describe survival outcome
`predict.factors`	list of predictor names
`fixed_time`	time at which performance is maximized
`tuningparams`	if given, list of hyperparameters, list(mtry=c(), nodedepth=c(),nodesize=c()), otherwise a wide default grid is used
`max_grid_size`	number of random grid searches for model tuning
`inner_cv`	number of cross-validation folds for hyperparameters' tuning
`randomseed`	random seed to control tuning including data splits
`verbose`	TRUE/FALSE, FALSE by default

Value

output = list(bestparams, allstats, model)

Examples

```
d <-simulate_nonlinear(100)
p<- names(d)[1:4]
tuningparams = list(
 "mtry" = c(5,10,15),
 "nodedepth" = c(5,10,15,20),
 "nodesize" =    c(20,30,50)
)
m_srf<- survsrf_train(d,p,tuningparams=tuningparams)
```

survsrf_tune *A repeated 3-fold CV over a hyperparameters grid*

Description

A repeated 3-fold CV over a hyperparameters grid

Usage

```
survsrf_tune(
  df_tune,
  predict.factors,
  repeat_tune = 1,
  fixed_time = NaN,
  tuningparams = list(),
  max_grid_size = 10,
  inner_cv = 3,
  randomseed = NaN
)
```

Arguments

df_tune	data
predict.factors	list of predictor names
repeat_tune	number of repeats
fixed_time	not used here, but for some models the time for which performance is optimized
tuningparams	if given, list of hyperparameters, list(mtry=c(), nodedepth=c(),nodesize=c()), otherwise a wide default grid is used
max_grid_size	number of random grid searches for model tuning
inner_cv	number of cross-validation folds for hyperparameter tuning
randomseed	to choose random subgroup of hyperparams

Value

output=list(cindex_ordered, bestparams)

survsrf_tune_single *Internal function for survsrf_tune(), performs 1 CV*

Description

Internal function for survsrf_tune(), performs 1 CV

Usage

```
survsrf_tune_single(
  df_tune,
  predict.factors,
  fixed_time = NaN,
  grid_hyperparams = c(),
  inner_cv = 3,
  randomseed = NaN,
  progressbar = FALSE
)
```

Arguments

`df_tune`	data
`predict.factors`	list of predictor names
`fixed_time`	predictions for which time are computed for c-index
`grid_hyperparams`	hyperparameters grid (or a default will be used)
`inner_cv`	number of folds for each CV
`randomseed`	randomseed
`progressbar`	FALSE(default)/TRUE

Value

output=list(grid, cindex, cindex_mean)

surv_brierscore *Calculates time-dependent Brier Score*

Description

Calculates time-dependent Brier Scores for a vector of times. Calculations are similar to that in: https://scikit-survival.readthedocs.io/en/stable/api/generated/sksurv.metrics.brier_score.html#sksurv.metrics.brier_score https://github.com/sebp/scikit-survival/blob/v0.19.0.post1/sksurv/metrics.py#L524-L644 The function uses IPCW (inverse probability of censoring weights), computed using the Kaplan-Meier survival function, where events are censored events from train data

Usage

```
surv_brierscore(
  y_predicted_newdata,
  df_brier_train,
  df_newdata,
  time_point,
  weighted = TRUE
)
```

Arguments

`y_predicted_newdata`	computed event probabilities (! not survival probabilities)
`df_brier_train`	train data
`df_newdata`	test data for which brier score is computed
`time_point`	times at which BS calculated
`weighted`	TRUE/FALSE for IPWC to use or not

Value

vector of time-dependent Brier Scores for all time_point

---

surv_validate — *Computes performance statistics for a survival data given the predicted event probabilities*

Description

Computes performance statistics for a survival data given the predicted event probabilities

Usage

```
surv_validate(
  y_predict,
  predict_time,
  df_train,
  df_test,
  weighted = TRUE,
  alpha = "logit"
)
```

Arguments

`y_predict`	probabilities of event by predict_time (matrix=observations x times)
`predict_time`	times for which event probabilities are given
`df_train`	train data, data frame
`df_test`	test data, data frame
`weighted`	TRUE/FALSE, for IPWC
`alpha`	calibration alpha as mean difference in probabilities, or in log-odds (from logistic regression, default)

Value

data.frame(T, AUCROC, Brier Score, Scaled Brier Score, C_score, Calib slope, Calib alpha)

Index