Package ‘survML’

July 23, 2025

Title Tools for Flexible Survival Analysis Using Machine Learning

Version 1.2.0

Description Statistical tools for analyzing time-to-event data using machine learning. Implements survival stacking for conditional survival estimation, standardized survival function estimation for current status data, and methods for algorithm-agnostic variable importance. See Wolock CJ, Gilbert PB, Simon N, and Carone M (2024) <doi:10.1080/10618600.2024.2304070>.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Depends SuperLearner (>= 2.0.28),

Imports Iso (>= 0.0.18.1), haldensify (>= 0.2.3), fdrtool (>= 1.2.17), ChernoffDist (>= 0.1.0), dplyr (>= 1.0.10), gtools (>= 3.9.5), mboost (>= 2.9.0), survival (>= 3.5.0), stats (>= 4.3.2), methods (>= 4.3.2)

Suggests knitr, rmarkdown, testthat (>= 3.0.0), ggplot2 (>= 3.4.0), gam (>= 1.22.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/cwolock/survML, https://cwolock.github.io/survML/

BugReports https://github.com/cwolock/survML/issues

NeedsCompilation no

Author Charles Wolock [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-3527-1102>), Avi Kenny [ctb] (ORCID: <https://orcid.org/0000-0002-9465-7307>)

Maintainer Charles Wolock <cwolock@gmail.com>

Repository CRAN

Date/Publication 2024-10-31 00:20:01 UTC

Contents

`crossfit_oracle_preds` *Generate cross-fitted oracle prediction function estimates*

Description

Generate cross-fitted oracle prediction function estimates

Usage

```
crossfit_oracle_preds(
  time,
  event,
  X,
  folds,
  nuisance_preds,
  pred_generator,
  ...
)
```

Arguments

`time`	`n x 1` numeric vector of observed follow-up times. If there is censoring, these are the minimum of the event and censoring times.
`event`	`n x 1` numeric vector of status indicators of whether an event was observed.
`X`	`n x p` data.frame of observed covariate values
`folds`	`n x 1` numeric vector of folds identifiers for cross-fitting

`nuisance_preds`	Named list of conditional event and censoring survival functions that will be used to estimate the oracle prediction function.
`pred_generator`	Function to be used to estimate oracle prediction function.
`...`	Additional arguments to be passed to `pred_generator`.

Value

Named list of cross-fitted oracle prediction estimates

`crossfit_surv_preds` *Generate cross-fitted conditional survival predictions*

Description

Generate cross-fitted conditional survival predictions

Usage

```
crossfit_surv_preds(time, event, X, newtimes, folds, pred_generator, ...)
```

Arguments

`time`	`n x 1` numeric vector of observed follow-up times. If there is censoring, these are the minimum of the event and censoring times.
`event`	`n x 1` numeric vector of status indicators of whether an event was observed.
`X`	`n x p` data.frame of observed covariate values
`newtimes`	Numeric vector of times on which to estimate the conditional survival functions
`folds`	`n x 1` numeric vector of folds identifiers for cross-fitting
`pred_generator`	Function to be used to estimate conditional survival function.
`...`	Additional arguments to be passed to `pred_generator`.

Value

Named list of cross-fitted conditional survival predictions currstatCIR *Estimate a survival function under current status sampling*

Description

Estimate a survival function under current status sampling

Usage

```
currstatCIR(
  time,
  event,
  X,
  SL_control = list(SL.library = c("SL.mean", "SL.glm"), V = 3),
  HAL_control = list(n_bins = c(5), grid_type = c("equal_mass"), V = 3),
  deriv_method = "m-spline",
  eval_region,
  n_eval_pts = 101,
  alpha = 0.05
)
```

Arguments

`time`	`n x 1` numeric vector of observed monitoring times. For individuals that were never monitored, this can be set to any arbitrary value, including `NA`, as long as the corresponding `event` variable is `NA`.
`event`	`n x 1` numeric vector of status indicators of whether an event was observed prior to the monitoring time. This value must be `NA` for individuals that were never monitored.
`X`	`n x p` dataframe of observed covariate values.
`SL_control`	List of `SuperLearner` control parameters. This should be a named list; see `SuperLearner` documentation for further information.
`HAL_control`	List of `haldensify` control parameters. This should be a named list; see `haldensify` documentation for further information.
`deriv_method`	Method for computing derivative. Options are `"m-spline"` (the default, fit a smoothing spline to the estimated function and differentiate the smooth approximation), `"linear"` (linearly interpolate the estimated function and use the slope of that line), and `"line"` (use the slope of the line connecting the endpoints of the estimated function).
`eval_region`	Region over which to estimate the survival function.
`n_eval_pts`	Number of points in grid on which to evaluate survival function. The points will be evenly spaced, on the quantile scale, between the endpoints of `eval_region`.
`alpha`	The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05

Value

Data frame giving results, with columns:

`t`	Time at which survival function is estimated
`S_hat_est`	Survival function estimate
`S_hat_cil`	Lower bound of confidence interval
`S_hat_ciu`	Upper bound of confidence interval

Examples

```
## Not run: # This is a small simulation example
set.seed(123)
n <- 300
x <- cbind(2*rbinom(n, size = 1, prob = 0.5)-1,
           2*rbinom(n, size = 1, prob = 0.5)-1)
t <- rweibull(n,
              shape = 0.75,
              scale = exp(0.4*x[,1] - 0.2*x[,2]))
y <- rweibull(n,
              shape = 0.75,
              scale = exp(0.4*x[,1] - 0.2*x[,2]))

# round y to nearest quantile of y, just so there aren't so many unique values
quants <- quantile(y, probs = seq(0, 1, by = 0.05), type = 1)
for (i in 1:length(y)){
  y[i] <- quants[which.min(abs(y[i] - quants))]
}
delta <- as.numeric(t <= y)

dat <- data.frame(y = y, delta = delta, x1 = x[,1], x2 = x[,2])

dat$delta[dat$y > 1.8] <- NA
dat$y[dat$y > 1.8] <- NA
eval_region <- c(0.05, 1.5)
res <- survML::currstatCIR(time = dat$y,
                           event = dat$delta,
                           X = dat[,3:4],
                           SL_control = list(SL.library = c("SL.mean", "SL.glm"),
                                             V = 3),
                           HAL_control = list(n_bins = c(5),
                                              grid_type = c("equal_mass"),
                                              V = 3),
                           eval_region = eval_region)

xvals = res$t
yvals = res$S_hat_est
fn=stepfun(xvals, c(yvals[1], yvals))
plot.function(fn, from=min(xvals), to=max(xvals))
## End(Not run)
```

`DR_pseudo_outcome_regression` *Generate oracle prediction function estimates using doubly-robust pseudo-outcome regression with SuperLearner*

Description

Generate oracle prediction function estimates using doubly-robust pseudo-outcome regression with SuperLearner

Usage

```
DR_pseudo_outcome_regression(
  time,
  event,
  X,
  newX,
  approx_times,
  S_hat,
  G_hat,
  newtimes,
  outcome,
  SL.library,
  V
)
```

Arguments

`time`	`n x 1` numeric vector of observed follow-up times. If there is censoring, these are the minimum of the event and censoring times.
`event`	`n x 1` numeric vector of status indicators of whether an event was observed.
`X`	`n x p` data.frame of observed covariate values
`newX`	`m x p` data.frame of new observed covariate values at which to obtain `m` predictions for the estimated algorithm. Must have the same names and structure as `X`.
`approx_times`	Numeric vector of length J2 giving times at which to approximate integral appearing in the pseudo-outcomes
`S_hat`	`n x J2` matrix of conditional event time survival function estimates
`G_hat`	`n x J2` matrix of conditional censoring time survival function estimates
`newtimes`	Numeric vector of times at which to generate oracle prediction function estimates
`outcome`	Outcome type, either `"survival_probability"` or `"restricted_survival_time"`
`SL.library`	Super Learner library
`V`	Number of cross-validation folds, to be passed to `SuperLearner`

Value

Matrix of predictions.

`generate_folds` *Generate cross-fitting and sample-splitting folds*

Description

Generate cross-fitting and sample-splitting folds

Usage

```
generate_folds(n, V, sample_split)
```

Arguments

`n`	Total sample size
`V`	Number of cross-fitting folds to use
`sample_split`	Logical, whether or not sample-splitting is being used

Value

Named list of cross-fitting and sample-splitting folds

`predict.stackG` *Obtain predicted conditional survival and cumulative hazard functions from a global survival stacking object*

Description

Obtain predicted conditional survival and cumulative hazard functions from a global survival stacking object

Usage

```
## S3 method for class 'stackG'
predict(
  object,
  newX,
  newtimes,
  surv_form = object$surv_form,
  time_grid_approx = object$time_grid_approx,
  ...
)
```

Arguments

- `object` — Object of class `stackG`
- `newX` — `m` x `p` data.frame of new observed covariate values at which to obtain `m` predictions for the estimated algorithm. Must have the same names and structure as `X`.
- `newtimes` — `k` x `1` numeric vector of times at which to obtain `k` predicted conditional survivals.
- `surv_form` — Mapping from hazard estimate to survival estimate. Can be either `"PI"` (product integral mapping) or `"exp"` (exponentiated cumulative hazard estimate). Defaults to the value saved in `object`.
- `time_grid_approx` — Numeric vector of times at which to approximate product integral or cumulative hazard interval. Defaults to the value saved in `object`.
- `...` — Further arguments passed to or from other methods.

Value

A named list with the following components:

- `S_T_preds` — An `m` x `k` matrix of estimated event time survival probabilities at the `m` covariate vector values and `k` times provided by the user in `newX` and `newtimes`, respectively.
- `S_C_preds` — An `m` x `k` matrix of estimated censoring time survival probabilities at the `m` covariate vector values and `k` times provided by the user in `newX` and `newtimes`, respectively.
- `Lambda_T_preds` — An `m` x `k` matrix of estimated event time cumulative hazard function values at the `m` covariate vector values and `k` times provided by the user in `newX` and `newtimes`, respectively.
- `Lambda_C_preds` — An `m` x `k` matrix of estimated censoring time cumulative hazard function values at the `m` covariate vector values and `k` times provided by the user in `newX` and `newtimes`, respectively.
- `time_grid_approx` — The approximation grid for the product integral or cumulative hazard integral, (user-specified).
- `surv_form` — Exponential or product-integral form (user-specified).

See Also

stackG

Examples

```
# This is a small simulation example
set.seed(123)
n <- 250
X <- data.frame(X1 = rnorm(n), X2 = rbinom(n, size = 1, prob = 0.5))
```

```
S0 <- function(t, x){
  pexp(t, rate = exp(-2 + x[,1] - x[,2] + .5 * x[,1] * x[,2]), lower.tail = FALSE)
}
T <- rexp(n, rate = exp(-2 + X[,1] - X[,2] + .5 *  X[,1] * X[,2]))

G0 <- function(t, x) {
  as.numeric(t < 15) *.9*pexp(t,
                              rate = exp(-2 -.5*x[,1]-.25*x[,2]+.5*x[,1]*x[,2]),
                              lower.tail=FALSE)
}
C <- rexp(n, exp(-2 -.5 * X[,1] - .25 * X[,2] + .5 * X[,1] * X[,2]))
C[C > 15] <- 15

entry <- runif(n, 0, 15)

time <- pmin(T, C)
event <- as.numeric(T <= C)

sampled <- which(time >= entry)
X <- X[sampled,]
time <- time[sampled]
event <- event[sampled]
entry <- entry[sampled]

# Note that this a very small Super Learner library, for computational purposes.
SL.library <- c("SL.mean", "SL.glm")

fit <- stackG(time = time,
              event = event,
              entry = entry,
              X = X,
              newX = X,
              newtimes = seq(0, 15, .1),
              direction = "prospective",
              bin_size = 0.1,
              time_basis = "continuous",
              time_grid_approx = sort(unique(time)),
              surv_form = "exp",
              learner = "SuperLearner",
              SL_control = list(SL.library = SL.library,
                                V = 5))

preds <- predict(object = fit,
                 newX = X,
                 newtimes = seq(0, 15, 0.1))

plot(preds$S_T_preds[1,], S0(t =  seq(0, 15, .1), X[1,]))
abline(0,1,col='red')
```

`predict.stackL` *Obtain predicted conditional survival function from a local survival stacking object*

Description

Obtain predicted conditional survival function from a local survival stacking object

Usage

```
## S3 method for class 'stackL'
predict(object, newX, newtimes, ...)
```

Arguments

`object`	Object of class `stackL`
`newX`	`m` x `p` data.frame of new observed covariate values at which to obtain `m` predictions for the estimated algorithm. Must have the same names and structure as `X`.
`newtimes`	`k` x `1` numeric vector of times at which to obtain `k` predicted conditional survivals.
`...`	Further arguments passed to or from other methods.

Value

A named list with the following components:

`S_T_preds`	An `m` x `k` matrix of estimated event time survival probabilities at the `m` covariate vector values and `k` times provided by the user in `newX` and `newtimes`, respectively.

See Also

stackL

Examples

```
# This is a small simulation example
set.seed(123)
n <- 500
X <- data.frame(X1 = rnorm(n), X2 = rbinom(n, size = 1, prob = 0.5))

S0 <- function(t, x){
  pexp(t, rate = exp(-2 + x[,1] - x[,2] + .5 * x[,1] * x[,2]), lower.tail = FALSE)
}
T <- rexp(n, rate = exp(-2 + X[,1] - X[,2] + .5 *  X[,1] * X[,2]))

G0 <- function(t, x) {
  as.numeric(t < 15) *.9*pexp(t,
```

```
                              rate = exp(-2 -.5*x[,1]-.25*x[,2]+.5*x[,1]*x[,2]),
                              lower.tail=FALSE)
}
C <- rexp(n, exp(-2 -.5 * X[,1] - .25 * X[,2] + .5 * X[,1] * X[,2]))
C[C > 15] <- 15

entry <- runif(n, 0, 15)

time <- pmin(T, C)
event <- as.numeric(T <= C)

sampled <- which(time >= entry)
X <- X[sampled,]
time <- time[sampled]
event <- event[sampled]
entry <- entry[sampled]

# Note that this a very small Super Learner library, for computational purposes.
SL.library <- c("SL.mean", "SL.glm")

fit <- stackL(time = time,
               event = event,
               entry = entry,
               X = X,
               newX = X,
               newtimes = seq(0, 15, .1),
               direction = "prospective",
               bin_size = 0.1,
               time_basis = "continuous",
               SL_control = list(SL.library = SL.library,
                                 V = 5))

preds <- predict(object = fit,
                 newX = X,
                 newtimes = seq(0, 15, 0.1))

plot(preds$S_T_preds[1,], S0(t =  seq(0, 15, .1), X[1,]))
abline(0,1,col='red')
```

`stackG` *Estimate a conditional survival function using global survival stacking*

Description

Estimate a conditional survival function using global survival stacking

Usage

```
stackG(
  time,
```

```
  event = rep(1, length(time)),
  entry = NULL,
  X,
  newX = NULL,
  newtimes = NULL,
  direction = "prospective",
  time_grid_fit = NULL,
  bin_size = NULL,
  time_basis,
  time_grid_approx = sort(unique(time)),
  surv_form = "PI",
  learner = "SuperLearner",
 SL_control = list(SL.library = c("SL.mean"), V = 10, method = "method.NNLS", stratifyCV
   = FALSE),
  tau = NULL
)
```

Arguments

time — `n x 1` numeric vector of observed follow-up times If there is censoring, these are the minimum of the event and censoring times.

event — `n x 1` numeric vector of status indicators of whether an event was observed. Defaults to a vector of 1s, i.e. no censoring.

entry — Study entry variable, if applicable. Defaults to `NULL`, indicating that there is no truncation.

X — `n x p` data.frame of observed covariate values on which to train the estimator.

newX — `m x p` data.frame of new observed covariate values at which to obtain `m` predictions for the estimated algorithm. Must have the same names and structure as `X`.

newtimes — `k x 1` numeric vector of times at which to obtain `k` predicted conditional survivals.

direction — Whether the data come from a prospective or retrospective study. This determines whether the data are treated as subject to left truncation and right censoring (`"prospective"`) or right truncation alone (`"retrospective"`).

time_grid_fit — Named list of numeric vectors of times of times on which to discretize for estimation of cumulative probability functions. This is an alternative to `bin_size` and allows for specially tailored time grids rather than simply using a quantile bin size. The list consists of vectors named `F_Y_1_grid`, `F_Y_0_grid`, `G_W_1_grid`, and `G_W_0_grid`. These denote, respectively, the grids used to estimate the conditional CDF of the `time` variable among uncensored and censored observations, and the grids used to estimate the conditional distribution of the `entry` variable among uncensored and censored observations.

bin_size — Size of time bin on which to discretize for estimation of cumulative probability functions. Can be a number between 0 and 1, indicating the size of quantile grid (e.g. `0.1` estimates the cumulative probability functions on a grid based on deciles of observed `times`). If `NULL`, creates a grid of all observed `times`.

`time_basis` How to treat time for training the binary classifier. Options are `"continuous"` and `"dummy"`, meaning an indicator variable is included for each time in the time grid.

`time_grid_approx` Numeric vector of times at which to approximate product integral or cumulative hazard interval. Defaults to `times` argument.

`surv_form` Mapping from hazard estimate to survival estimate. Can be either `"PI"` (product integral mapping) or `"exp"` (exponentiated cumulative hazard estimate).

`learner` Which binary regression algorithm to use. Currently, only `SuperLearner` is supported, but more learners will be added. See below for algorithm-specific arguments.

`SL_control` Named list of parameters controlling the Super Learner fitting process. These parameters are passed directly to the `SuperLearner` function. Parameters include `SL.library` (library of algorithms to include in the binary classification Super Learner), `V` (Number of cross validation folds on which to train the Super Learner classifier, defaults to 10), `method` (Method for estimating coefficients for the Super Learner, defaults to `"method.NNLS"`), `stratifyCV` (logical indicating whether to stratify by outcome in `SuperLearner`'s cross-validation scheme), and `obsWeights` (observation weights, passed directly to prediction algorithms by `SuperLearner`).

`tau` The maximum time of interest in a study, used for retrospective conditional survival estimation. Rather than dealing with right truncation separately than left truncation, it is simpler to estimate the survival function of `tau - time`. Defaults to `NULL`, in which case the maximum study entry time is chosen as the reference point.

Value

A named list of class `stackG`, with the following components:

`S_T_preds` An `m` x `k` matrix of estimated event time survival probabilities at the `m` covariate vector values and `k` times provided by the user in `newX` and `newtimes`, respectively.

`S_C_preds` An `m` x `k` matrix of estimated censoring time survival probabilities at the `m` covariate vector values and `k` times provided by the user in `newX` and `newtimes`, respectively.

`Lambda_T_preds` An `m` x `k` matrix of estimated event time cumulative hazard function values at the `m` covariate vector values and `k` times provided by the user in `newX` and `newtimes`, respectively.

`Lambda_C_preds` An `m` x `k` matrix of estimated censoring time cumulative hazard function values at the `m` covariate vector values and `k` times provided by the user in `newX` and `newtimes`, respectively.

`time_grid_approx` The approximation grid for the product integral or cumulative hazard integral, (user-specified).

`direction` Whether the data come from a prospective or retrospective study (user-specified).

`tau`	The maximum time of interest in a study, used for retrospective conditional survival estimation (user-specified).
`surv_form`	Exponential or product-integral form (user-specified).
`time_basis`	Whether time is included in the regression as `continuous` or `dummy` (user-specified).
`SL_control`	Named list of parameters controlling the Super Learner fitting process (user-specified).
`fits`	A named list of fitted regression objects corresponding to the constituent regressions needed for global survival stacking. Includes `P_Delta` (probability of event given covariates), `F_Y_1` (conditional cdf of follow-up times given covariates among uncensored), `F_Y_0` (conditional cdf of follow-up times given covariates among censored), `G_W_1` (conditional distribution of entry times given covariates and follow-up time among uncensored), `G_W_0` (conditional distribution of entry times given covariates and follow-up time among uncensored). Each of these objects includes estimated coefficients from the `SuperLearner` fit, as well as the time grid used to create the stacked dataset (where applicable).

References

Wolock C.J., Gilbert P.B., Simon N., and Carone, M. (2024). "A framework for leveraging machine learning tools to estimate personalized survival curves."

See Also

predict.stackG for `stackG` prediction method.

Examples

```
# This is a small simulation example
set.seed(123)
n <- 250
X <- data.frame(X1 = rnorm(n), X2 = rbinom(n, size = 1, prob = 0.5))

S0 <- function(t, x){
  pexp(t, rate = exp(-2 + x[,1] - x[,2] + .5 * x[,1] * x[,2]), lower.tail = FALSE)
}
T <- rexp(n, rate = exp(-2 + X[,1] - X[,2] + .5 *  X[,1] * X[,2]))

G0 <- function(t, x) {
  as.numeric(t < 15) *.9*pexp(t,
                              rate = exp(-2 -.5*x[,1]-.25*x[,2]+.5*x[,1]*x[,2]),
                              lower.tail=FALSE)
}
C <- rexp(n, exp(-2 -.5 * X[,1] - .25 * X[,2] + .5 * X[,1] * X[,2]))
C[C > 15] <- 15

entry <- runif(n, 0, 15)

time <- pmin(T, C)
event <- as.numeric(T <= C)
```

```
sampled <- which(time >= entry)
X <- X[sampled,]
time <- time[sampled]
event <- event[sampled]
entry <- entry[sampled]

# Note that this a very small Super Learner library, for computational purposes.
SL.library <- c("SL.mean", "SL.glm")

fit <- stackG(time = time,
              event = event,
              entry = entry,
              X = X,
              newX = X,
              newtimes = seq(0, 15, .1),
              direction = "prospective",
              bin_size = 0.1,
              time_basis = "continuous",
              time_grid_approx = sort(unique(time)),
              surv_form = "exp",
              learner = "SuperLearner",
              SL_control = list(SL.library = SL.library,
                                V = 5))

plot(fit$S_T_preds[1,], S0(t =  seq(0, 15, .1), X[1,]))
abline(0,1,col='red')
```

`stackL` *Estimate a conditional survival function via local survival stacking*

Description

Estimate a conditional survival function via local survival stacking

Usage

```
stackL(
  time,
  event = rep(1, length(time)),
  entry = NULL,
  X,
  newX,
  newtimes,
  direction = "prospective",
  bin_size = NULL,
  time_basis = "continuous",
  learner = "SuperLearner",
 SL_control = list(SL.library = c("SL.mean"), V = 10, method = "method.NNLS", stratifyCV
```

```
    = FALSE),
  tau = NULL
)
```

Arguments

- `time` — `n x 1` numeric vector of observed follow-up times If there is censoring, these are the minimum of the event and censoring times.
- `event` — `n x 1` numeric vector of status indicators of whether an event was observed. Defaults to a vector of 1s, i.e. no censoring.
- `entry` — Study entry variable, if applicable. Defaults to `NULL`, indicating that there is no truncation.
- `X` — `n x p` data.frame of observed covariate values on which to train the estimator.
- `newX` — `m x p` data.frame of new observed covariate values at which to obtain `m` predictions for the estimated algorithm. Must have the same names and structure as `X`.
- `newtimes` — `k x 1` numeric vector of times at which to obtain `k` predicted conditional survivals.
- `direction` — Whether the data come from a prospective or retrospective study. This determines whether the data are treated as subject to left truncation and right censoring (`"prospective"`) or right truncation alone (`"retrospective"`).
- `bin_size` — Size of bins for the discretization of time. A value between 0 and 1 indicating the size of observed event time quantiles on which to grid times (e.g. 0.02 creates a grid of 50 times evenly spaced on the quantile scaled). If NULL, defaults to every observed event time.
- `time_basis` — How to treat time for training the binary classifier. Options are `"continuous"` and `"dummy"`, meaning an indicator variable is included for each time in the time grid.
- `learner` — Which binary regression algorithm to use. Currently, only `SuperLearner` is supported, but more learners will be added. See below for algorithm-specific arguments.
- `SL_control` — Named list of parameters controlling the Super Learner fitting process. These parameters are passed directly to the `SuperLearner` function. Parameters include `SL.library` (library of algorithms to include in the binary classification Super Learner), `V` (Number of cross validation folds on which to train the Super Learner classifier, defaults to 10), `method` (Method for estimating coefficients for the Super Learner, defaults to `"method.NNLS"`), `stratifyCV` (logical indicating whether to stratify by outcome in `SuperLearner`'s cross-validation scheme), and `obsWeights` (observation weights, passed directly to prediction algorithms by `SuperLearner`).
- `tau` — The maximum time of interest in a study, used for retrospective conditional survival estimation. Rather than dealing with right truncation separately than left truncation, it is simpler to estimate the survival function of `tau - time`. Defaults to `NULL`, in which case the maximum study entry time is chosen as the reference point.

Value

A named list of class stackL.

S_T_preds
: An m x k matrix of estimated event time survival probabilities at the m covariate vector values and k times provided by the user in newX and newtimes, respectively.

fit
: The Super Learner fit for binary classification on the stacked dataset.

References

Polley E.C. and van der Laan M.J. (2011). "Super Learning for Right-Censored Data" in Targeted Learning.

Craig E., Zhong C., and Tibshirani R. (2021). "Survival stacking: casting survival analysis as a classification problem."

See Also

predict.stackL for stackL prediction method.

Examples

```
# This is a small simulation example
set.seed(123)
n <- 500
X <- data.frame(X1 = rnorm(n), X2 = rbinom(n, size = 1, prob = 0.5))

S0 <- function(t, x){
  pexp(t, rate = exp(-2 + x[,1] - x[,2] + .5 * x[,1] * x[,2]), lower.tail = FALSE)
}
T <- rexp(n, rate = exp(-2 + X[,1] - X[,2] + .5 *  X[,1] * X[,2]))

G0 <- function(t, x) {
  as.numeric(t < 15) *.9*pexp(t,
                              rate = exp(-2 -.5*x[,1]-.25*x[,2]+.5*x[,1]*x[,2]),
                              lower.tail=FALSE)
}
C <- rexp(n, exp(-2 -.5 * X[,1] - .25 * X[,2] + .5 * X[,1] * X[,2]))
C[C > 15] <- 15

entry <- runif(n, 0, 15)

time <- pmin(T, C)
event <- as.numeric(T <= C)

sampled <- which(time >= entry)
X <- X[sampled,]
time <- time[sampled]
event <- event[sampled]
entry <- entry[sampled]

# Note that this a very small Super Learner library, for computational purposes.
```

```
SL.library <- c("SL.mean", "SL.glm")

fit <- stackL(time = time,
              event = event,
              entry = entry,
              X = X,
              newX = X,
              newtimes = seq(0, 15, .1),
              direction = "prospective",
              bin_size = 0.1,
              time_basis = "continuous",
              SL_control = list(SL.library = SL.library,
                                V = 5))

plot(fit$S_T_preds[1,], S0(t =  seq(0, 15, .1), X[1,]))
abline(0,1,col='red')
```

---

vim *Estimate AUC VIM*

Description

Estimate AUC VIM

Usage

```
vim(
  type,
  time,
  event,
  X,
  landmark_times = stats::quantile(time[event == 1], probs = c(0.25, 0.5, 0.75)),
  restriction_time = max(time[event == 1]),
  approx_times = NULL,
  large_feature_vector,
  small_feature_vector,
  conditional_surv_preds = NULL,
  large_oracle_preds = NULL,
  small_oracle_preds = NULL,
  conditional_surv_generator = NULL,
  conditional_surv_generator_control = NULL,
  large_oracle_generator = NULL,
  large_oracle_generator_control = NULL,
  small_oracle_generator = NULL,
  small_oracle_generator_control = NULL,
  cf_folds = NULL,
  cf_fold_num = 5,
```

```
  sample_split = TRUE,
  ss_folds = NULL,
  robust = TRUE,
  scale_est = FALSE,
  alpha = 0.05,
  verbose = FALSE
)
```

Arguments

- `type` — Type of VIM to compute. Options include "accuracy", "AUC", "Brier", "R-squared" "C-index", and "survival_time_MSE".
- `time` — n x 1 numeric vector of observed follow-up times. If there is censoring, these are the minimum of the event and censoring times.
- `event` — n x 1 numeric vector of status indicators of whether an event was observed.
- `X` — n x p data.frame of observed covariate values
- `landmark_times` — Numeric vector of length J1 giving landmark times at which to estimate VIM ("accuracy", "AUC", "Brier", "R-squared").
- `restriction_time` — Maximum follow-up time for calculation of "C-index" and "survival_time_MSE".
- `approx_times` — Numeric vector of length J2 giving times at which to approximate integrals. Defaults to a grid of 100 timepoints, evenly spaced on the quantile scale of the distribution of observed event times.
- `large_feature_vector` — Numeric vector giving indices of features to include in the 'large' prediction model.
- `small_feature_vector` — Numeric vector giving indices of features to include in the 'small' prediction model. Must be a subset of `large_feature_vector`.
- `conditional_surv_preds` — User-provided estimates of the conditional survival functions of the event and censoring variables given the full covariate vector (if not using the `vim()` function to compute these nuisance estimates). Must be a named list of lists with elements `S_hat`, `S_hat_train`, `G_hat`, and `G_hat_train`. Each of these is itself a list of length `K`, where `K` is the number of cross-fitting folds. Each element of these lists is a matrix with J2 columns and number of rows equal to either the number of samples in the kth fold (for `S_hat` or `G_hat`) or the number of samples used to compute the nuisance estimator for the kth fold.
- `large_oracle_preds` — User-provided estimates of the oracle prediction function using `large_feature_vector`. Must be a named list of lists with elements `f_hat` and `f_hat_train`. Each of these is itself a list of length `K`. Each element of these lists is a matrix with J1 columns (for landmark time VIMs) or 1 column (for "C-index" and "survival_time_MSE").
- `small_oracle_preds` — User-provided estimates of the oracle prediction function using `small_feature_vector`. Must be a named list of lists with elements `f_hat` and `f_hat_train`. Each of

these is itself a list of length K. Each element of these lists is a matrix with J1 columns (for landmark time VIMs) or 1 column (for "C-index" and "survival_time_MSE").

conditional_surv_generator
: A user-written function to estimate the conditional survival functions of the event and censoring variables. Must take arguments time, event, folds (cross-fitting fold identifiers), and newtimes (times at which to generate predictions).

conditional_surv_generator_control
: A list of arguments to pass to conditional_surv_generator.

large_oracle_generator
: A user-written function to estimate the oracle prediction function using large_feature_vector.Must take arguments time, event, and folds (cross-fitting fold identifiers).

large_oracle_generator_control
: A list of arguments to pass to large_oracle_generator.

small_oracle_generator
: A user-written function to estimate the oracle prediction function using small_feature_vector.Must take arguments time, event, and folds (cross-fitting fold identifiers).

small_oracle_generator_control
: A list of arguments to pass to small_oracle_generator.

cf_folds
: Numeric vector of length n giving cross-fitting folds

cf_fold_num
: The number of cross-fitting folds, if not providing cf_folds

sample_split
: Logical indicating whether or not to sample split

ss_folds
: Numeric vector of length n giving sample-splitting folds

robust
: Logical, whether or not to use the doubly-robust debiasing approach. This option is meant for illustration purposes only — it should be left as TRUE.

scale_est
: Logical, whether or not to force the VIM estimate to be nonnegative

alpha
: The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05

verbose
: Whether to print progress messages.

Value

Named list with the following elements:

result
: Data frame giving results. See the documentation of the individual vim_* functions for details.

folds
: A named list giving the cross-fitting fold IDs (cf_folds) and sample-splitting fold IDs (ss_folds).

approx_times
: A vector of times used to approximate integrals appearing in the form of the VIM estimator.

conditional_surv_preds
: A named list containing the estimated conditional event and censoring survival functions.

large_oracle_preds
: A named list containing the estimated large oracle prediction function.

small_oracle_preds
: A named list containing the estimated small oracle prediction function.

See Also

vim_accuracy vim_AUC vim_brier vim_cindex vim_rsquared vim_survival_time_mse

Examples

```
# This is a small simulation example
set.seed(123)
n <- 100
X <- data.frame(X1 = rnorm(n), X2 = rbinom(n, size = 1, prob = 0.5))

T <- rexp(n, rate = exp(-2 + X[,1] - X[,2] + .5 *  X[,1] * X[,2]))

C <- rexp(n, exp(-2 -.5 * X[,1] - .25 * X[,2] + .5 * X[,1] * X[,2]))
C[C > 15] <- 15

time <- pmin(T, C)
event <- as.numeric(T <= C)

# landmark times for AUC
landmark_times <- c(3)

output <- vim(type = "AUC",
              time = time,
              event = event,
              X = X,
              landmark_times = landmark_times,
              large_feature_vector = 1:2,
              small_feature_vector = 2,
          conditional_surv_generator_control = list(SL.library = c("SL.mean", "SL.glm")),
             large_oracle_generator_control = list(SL.library = c("SL.mean", "SL.glm")),
             small_oracle_generator_control = list(SL.library = c("SL.mean", "SL.glm")),
              cf_fold_num = 2,
              sample_split = FALSE,
              scale_est = TRUE)

print(output$result)
```

`vim_accuracy` *Estimate classification accuracy VIM*

Description

Estimate classification accuracy VIM

Usage

```
vim_accuracy(
  time,
  event,
```

```
  approx_times,
  landmark_times,
  f_hat,
  fs_hat,
  S_hat,
  G_hat,
  cf_folds,
  sample_split,
  ss_folds,
  scale_est = FALSE,
  alpha = 0.05
)
```

Arguments

time	n x 1 numeric vector of observed follow-up times If there is censoring, these are the minimum of the event and censoring times.
event	n x 1 numeric vector of status indicators of whether an event was observed. Defaults to a vector of 1s, i.e. no censoring.
approx_times	Numeric vector of length J1 giving times at which to approximate integrals.
landmark_times	Numeric vector of length J2 giving times at which to estimate accuracy
f_hat	Full oracle predictions (n x J1 matrix)
fs_hat	Residual oracle predictions (n x J1 matrix)
S_hat	Estimates of conditional event time survival function (n x J2 matrix)
G_hat	Estimate of conditional censoring time survival function (n x J2 matrix)
cf_folds	Numeric vector of length n giving cross-fitting folds
sample_split	Logical indicating whether or not to sample split
ss_folds	Numeric vector of length n giving sample-splitting folds
scale_est	Logical, whether or not to force the VIM estimate to be nonnegative
alpha	The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05

Value

A data frame giving results, with the following columns:

landmark_time	Time at which AUC is evaluated.
est	VIM point estimate.
var_est	Estimated variance of the VIM estimate.
cil	Lower bound of the VIM confidence interval.
ciu	Upper bound of the VIM confidence interval.
cil_1sided	Lower bound of a one-sided confidence interval.
p	p-value corresponding to a hypothesis test of null importance.

`large_predictiveness`
: Estimated predictiveness of the large oracle prediction function.

`small_predictiveness`
: Estimated predictiveness of the small oracle prediction function.

`vim`
: VIM type.

`large_feature_vector`
: Group of features available for the large oracle prediction function.

`small_feature_vector`
: Group of features available for the small oracle prediction function.

`vim_AUC` *Estimate AUC VIM*

Description

Estimate AUC VIM

Usage

```
vim_AUC(
  time,
  event,
  approx_times,
  landmark_times,
  f_hat,
  fs_hat,
  S_hat,
  G_hat,
  cf_folds,
  sample_split,
  ss_folds,
  robust = TRUE,
  scale_est = FALSE,
  alpha = 0.05
)
```

Arguments

`time`	`n x 1` numeric vector of observed follow-up times If there is censoring, these are the minimum of the event and censoring times.
`event`	`n x 1` numeric vector of status indicators of whether an event was observed. Defaults to a vector of 1s, i.e. no censoring.
`approx_times`	Numeric vector of length J1 giving times at which to approximate integrals.
`landmark_times`	Numeric vector of length J2 giving times at which to estimate AUC
`f_hat`	Full oracle predictions (n x J1 matrix)

fs_hat	Residual oracle predictions (n x J1 matrix)
S_hat	Estimates of conditional event time survival function (n x J2 matrix)
G_hat	Estimate of conditional censoring time survival function (n x J2 matrix)
cf_folds	Numeric vector of length n giving cross-fitting folds
sample_split	Logical indicating whether or not to sample split
ss_folds	Numeric vector of length n giving sample-splitting folds
robust	Logical, whether or not to use the doubly-robust debiasing approach. This option is meant for illustration purposes only — it should be left as TRUE.
scale_est	Logical, whether or not to force the VIM estimate to be nonnegative
alpha	The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05

Value

A data frame giving results, with the following columns:

landmark_time	Time at which AUC is evaluated.
est	VIM point estimate.
var_est	Estimated variance of the VIM estimate.
cil	Lower bound of the VIM confidence interval.
ciu	Upper bound of the VIM confidence interval.
cil_1sided	Lower bound of a one-sided confidence interval.
p	p-value corresponding to a hypothesis test of null importance.
large_predictiveness	Estimated predictiveness of the large oracle prediction function.
small_predictiveness	Estimated predictiveness of the small oracle prediction function.
vim	VIM type.
large_feature_vector	Group of features available for the large oracle prediction function.
small_feature_vector	Group of features available for the small oracle prediction function.

See Also

vim for example usage

`vim_brier` *Estimate Brier score VIM*

Description

Estimate Brier score VIM

Usage

```
vim_brier(
  time,
  event,
  approx_times,
  landmark_times,
  f_hat,
  fs_hat,
  S_hat,
  G_hat,
  cf_folds,
  ss_folds,
  sample_split,
  scale_est = FALSE,
  alpha = 0.05
)
```

Arguments

`time`	`n x 1` numeric vector of observed follow-up times If there is censoring, these are the minimum of the event and censoring times.
`event`	`n x 1` numeric vector of status indicators of whether an event was observed. Defaults to a vector of 1s, i.e. no censoring.
`approx_times`	Numeric vector of length J1 giving times at which to approximate integrals.
`landmark_times`	Numeric vector of length J2 giving times at which to estimate Brier score
`f_hat`	Full oracle predictions (n x J1 matrix)
`fs_hat`	Residual oracle predictions (n x J1 matrix)
`S_hat`	Estimates of conditional event time survival function (n x J2 matrix)
`G_hat`	Estimate of conditional censoring time survival function (n x J2 matrix)
`cf_folds`	Numeric vector of length n giving cross-fitting folds
`ss_folds`	Numeric vector of length n giving sample-splitting folds
`sample_split`	Logical indicating whether or not to sample split
`scale_est`	Logical, whether or not to force the VIM estimate to be nonnegative
`alpha`	The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05

Value

A data frame giving results, with the following columns:

- `landmark_time` Time at which AUC is evaluated.
- `est` VIM point estimate.
- `var_est` Estimated variance of the VIM estimate.
- `cil` Lower bound of the VIM confidence interval.
- `ciu` Upper bound of the VIM confidence interval.
- `cil_1sided` Lower bound of a one-sided confidence interval.
- `p` p-value corresponding to a hypothesis test of null importance.
- `large_predictiveness` Estimated predictiveness of the large oracle prediction function.
- `small_predictiveness` Estimated predictiveness of the small oracle prediction function.
- `vim` VIM type.
- `large_feature_vector` Group of features available for the large oracle prediction function.
- `small_feature_vector` Group of features available for the small oracle prediction function.

See Also

vim for example usage

`vim_cindex` *Estimate concordance index VIM*

Description

Estimate concordance index VIM

Usage

```
vim_cindex(
  time,
  event,
  approx_times,
  restriction_time,
  f_hat,
  fs_hat,
  S_hat,
  G_hat,
  cf_folds,
  sample_split,
```

```
  ss_folds,
  scale_est = FALSE,
  alpha = 0.05
)
```

Arguments

`time` n x 1 numeric vector of observed follow-up times If there is censoring, these are the minimum of the event and censoring times.

`event` n x 1 numeric vector of status indicators of whether an event was observed. Defaults to a vector of 1s, i.e. no censoring.

`approx_times` Numeric vector of length J1 giving times at which to approximate integrals.

`restriction_time` Restriction time (upper bound for event times to be compared in computing the C-index)

`f_hat` Full oracle predictions (n x J1 matrix)

`fs_hat` Residual oracle predictions (n x J1 matrix)

`S_hat` Estimates of conditional event time survival function (n x J2 matrix)

`G_hat` Estimate of conditional censoring time survival function (n x J2 matrix)

`cf_folds` Numeric vector of length n giving cross-fitting folds

`sample_split` Logical indicating whether or not to sample split

`ss_folds` Numeric vector of length n giving sample-splitting folds

`scale_est` Logical, whether or not to force the VIM estimate to be nonnegative

`alpha` The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05

Value

A data frame giving results, with the following columns:

`restriction_time` Restriction time (upper bound for event times to be compared in computing the C-index).

`est` VIM point estimate.

`var_est` Estimated variance of the VIM estimate.

`cil` Lower bound of the VIM confidence interval.

`ciu` Upper bound of the VIM confidence interval.

`cil_1sided` Lower bound of a one-sided confidence interval.

`p` p-value corresponding to a hypothesis test of null importance.

`large_predictiveness` Estimated predictiveness of the large oracle prediction function.

`small_predictiveness` Estimated predictiveness of the small oracle prediction function.

`vim`	VIM type.
`large_feature_vector`	Group of features available for the large oracle prediction function.
`small_feature_vector`	Group of features available for the small oracle prediction function.

See Also

vim for example usage

`vim_rsquared` *Estimate R-squared (proportion of explained variance) VIM based on event occurrence by a landmark time*

Description

Estimate R-squared (proportion of explained variance) VIM based on event occurrence by a landmark time

Usage

```
vim_rsquared(
  time,
  event,
  approx_times,
  landmark_times,
  f_hat,
  fs_hat,
  S_hat,
  G_hat,
  cf_folds,
  ss_folds,
  sample_split,
  scale_est = FALSE,
  alpha = 0.05
)
```

Arguments

`time`	`n x 1` numeric vector of observed follow-up times If there is censoring, these are the minimum of the event and censoring times.
`event`	`n x 1` numeric vector of status indicators of whether an event was observed. Defaults to a vector of 1s, i.e. no censoring.
`approx_times`	Numeric vector of length J1 giving times at which to approximate integrals.
`landmark_times`	Numeric vector of length J2 giving times at which to estimate Brier score
`f_hat`	Full oracle predictions (n x J1 matrix)

fs_hat	Residual oracle predictions (n x J1 matrix)
S_hat	Estimates of conditional event time survival function (n x J2 matrix)
G_hat	Estimate of conditional censoring time survival function (n x J2 matrix)
cf_folds	Numeric vector of length n giving cross-fitting folds
ss_folds	Numeric vector of length n giving sample-splitting folds
sample_split	Logical indicating whether or not to sample split
scale_est	Logical, whether or not to force the VIM estimate to be nonnegative
alpha	The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05

Value

A data frame giving results, with the following columns:

landmark_time	Time at which AUC is evaluated.
est	VIM point estimate.
var_est	Estimated variance of the VIM estimate.
cil	Lower bound of the VIM confidence interval.
ciu	Upper bound of the VIM confidence interval.
cil_1sided	Lower bound of a one-sided confidence interval.
p	p-value corresponding to a hypothesis test of null importance.
large_predictiveness	Estimated predictiveness of the large oracle prediction function.
small_predictiveness	Estimated predictiveness of the small oracle prediction function.
vim	VIM type.
large_feature_vector	Group of features available for the large oracle prediction function.
small_feature_vector	Group of features available for the small oracle prediction function.

See Also

vim for example usage

vim_survival_time_mse *Estimate restricted predicted survival time MSE VIM*

Description

Estimate restricted predicted survival time MSE VIM

Usage

```
vim_survival_time_mse(
  time,
  event,
  approx_times,
  restriction_time,
  f_hat,
  fs_hat,
  S_hat,
  G_hat,
  cf_folds,
  sample_split,
  ss_folds,
  scale_est = FALSE,
  alpha = 0.05
)
```

Arguments

`time`	`n x 1` numeric vector of observed follow-up times If there is censoring, these are the minimum of the event and censoring times.
`event`	`n x 1` numeric vector of status indicators of whether an event was observed. Defaults to a vector of 1s, i.e. no censoring.
`approx_times`	Numeric vector of length J1 giving times at which to approximate integrals.
`restriction_time`	restriction time
`f_hat`	Full oracle predictions (n x J1 matrix)
`fs_hat`	Residual oracle predictions (n x J1 matrix)
`S_hat`	Estimates of conditional event time survival function (n x J2 matrix)
`G_hat`	Estimate of conditional censoring time survival function (n x J2 matrix)
`cf_folds`	Numeric vector of length n giving cross-fitting folds
`sample_split`	Logical indicating whether or not to sample split
`ss_folds`	Numeric vector of length n giving sample-splitting folds
`scale_est`	Logical, whether or not to force the VIM estimate to be nonnegative
`alpha`	The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05

Value

A data frame giving results, with the following columns:

`restriction_time`
: Restriction time (upper bound for event times to be compared in computing the restricted survival time).

`est`
: VIM point estimate.

`var_est`
: Estimated variance of the VIM estimate.

`cil`
: Lower bound of the VIM confidence interval.

`ciu`
: Upper bound of the VIM confidence interval.

`cil_1sided`
: Lower bound of a one-sided confidence interval.

`p`
: p-value corresponding to a hypothesis test of null importance.

`large_predictiveness`
: Estimated predictiveness of the large oracle prediction function.

`small_predictiveness`
: Estimated predictiveness of the small oracle prediction function.

`vim`
: VIM type.

`large_feature_vector`
: Group of features available for the large oracle prediction function.

`small_feature_vector`
: Group of features available for the small oracle prediction function.

See Also

vim for example usage

Index