

Package: survML (via r-universe)

October 4, 2026

Title Tools for Flexible Survival Analysis Using Machine Learning

Version 1.2.0.9000

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.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

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), copula (>= 1.1.0)

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>

Config/pak/sysreqs libgs10-dev libicu-dev

Repository <https://cwolock.r-universe.dev>

Date/Publication 2025-12-08 20:48:02 UTC

RemoteUrl <https://github.com/cwolock/survml>

RemoteRef HEAD

RemoteSha bb9a5793f732fba8e68311c664004e64967f7415

Contents

aggregate_vim	2
boost_c_index	3
crossfit_oracle_preds	5
crossfit_surv_preds	6
currstatCIR	7
DR_pseudo_outcome_regression	9
generate_folds	11
generate_nuisance_predictions_stackG	12
generate_oracle_predictions_boost	13
generate_oracle_predictions_DR	15
generate_oracle_predictions_SL	16
multiseed_vim	17
predict.stackG	21
predict.stackL	24
stackG	25
stackL	29
vim	32
vim_accuracy	37
vim_AUC	38
vim_brier	40
vim_cindex	41
vim_rsquared	43
vim_survival_time_mse	45
Index	47

aggregate_vim	<i>Aggregate multiseed VIM results</i>
---------------	--

Description

Aggregate multiseed VIM results

Usage

```
aggregate_vim(result_list, agg_method, ci_grid, n_eff, alpha = 0.05)
```

Arguments

result_list	List of result data frames return by the vim function.
agg_method	P-value aggregation method use to combine results from different seeds. Current options are "bonferroni" (Bonferroni's method), "hommel" (Hommel's method), "arithmetic" (arithmetic mean), "geometric" (geometric mean), "harmonic" (harmonic mean), "compound_bg" (compound Bonferroni and geometric mean), and "compound_ba" (compound Bonferroni and arithmetic mean). These approaches are discussed at length in Vovk and Wang (2020). Defaults to "compound_bg", which has been shown to work well in many settings.

ci_grid	Grid of VIM values over which to construct a confidence interval by inverting a hypothesis test. The aggregation works by constructing hypothesis tests (at level α) of the null corresponding to each value in ci_grid, and then inverting these tests to yield a $1 - \alpha$ confidence interval. For example, for "AUC" importance, the VIM takes values in (0,1), so a grid of values between 0 and 1 would be a reasonable choice.
n_eff	The effective sample size. Without sample-splitting, this is simply the sample size. With sample-splitting, this is the sample size divided by two (i.e., the size of each of the two halves of the data).
alpha	The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05.

Value

Named list with the following elements:

agg_result	Data frame giving results aggregated over seeds.
agg_method	P-value aggregation method used.
n_seed	Number of iterations (seeds) used to perform the VIM estimation procedure.
vim_objects	A list of vim return objects, each corresponding to a different seed.

boost_c_index	<i>Gradient boosting for C-index</i>
---------------	--------------------------------------

Description

Using doubly-robust gradient boosting to generate estimates of the prediction function that maximizes the C-index

Usage

```
boost_c_index(
  time,
  event,
  X,
  newX,
  S_hat,
  G_hat,
  V,
  approx_times,
  tuning,
  produce_fit = TRUE,
  subsample_n,
  boosting_params
)
```

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.
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
V	Number of cross-validation folds for selection of tuning parameters
approx_times	Numeric vector of length J2 giving times at which to approximate C-index integral. Note that the last time in approx_times is taken to be the restriction time (i.e., the maximum follow-up) for comparison of pairs of individuals. Essentially, this time should be chosen such that the conditional survival function is identified at this time for all covariate values X present in the data. Choosing the restriction time such that roughly 10% of individuals remain at-risk at that time has been shown to work reasonably well in simulations.
tuning	Logical, whether or not to use cross-validation to select tuning parameters
produce_fit	Logical, whether to produce a fitted prediction function using the selected optimal parameters.
subsample_n	Number of samples to use for boosting procedure. Using a subsample of the full sample can greatly reduce runtime
boosting_params	Named list of parameter values for the boosting procedure. Elements of this list include mstop (number of boosting iterations), nu (learning rate), sigma (smoothness parameter for sigmoid approximation, with smaller meaning less smoothing), and learner (base learner, can take values "glm", "gam", or "tree")

Value

Vector of predictions corresponding to newX

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))

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)
```

```

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

# Note that we do not use times beyond the 90th percentile of observed follow-up times
approx_times <- c(0, unique(quantile(time, probs = seq(0, 0.9, by = 0.01))))

# estimate conditional survival functions at approx_times
fit <- stackG(time = time,
              event = event,
              X = X,
              newX = X,
              newtimes = approx_times,
              direction = "prospective",
              bin_size = 0.1,
              time_basis = "continuous",
              surv_form = "PI",
              learner = "SuperLearner",
              time_grid_approx = approx_times,
              SL_control = list(SL.library = SL.library,
                                V = 3))

# use boosting to estimate optimal (according to C-index) prediction function
boosted_preds <- boost_c_index(time = time,
                               event = event,
                               X = X,
                               newX = X,
                               approx_times = approx_times,
                               S_hat = fit$S_T_preds,
                               G_hat = fit$S_C_preds,
                               V = 3,
                               tuning = TRUE,
                               produce_fit = TRUE,
                               subsample_n = 200,
                               boosting_params = list(mstop = c(100, 200),
                                                         nu = 0.1,
                                                         sigma = 0.1,
                                                         learner = "glm"))

boosted_preds

```

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	<i>Generate cross-fitted conditional survival predictions</i>
---------------------	---

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	<i>Estimate a survival function under current status sampling</i>
-------------	---

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",
  sample_split = FALSE,
  m = 5,
  eval_region = NULL,
  n_eval_pts = 101,
  eval_grid = NULL,
  alpha = 0.05,
  sensitivity_analysis = FALSE,
  copula_control = list(taus = c(-0.1, -0.05, 0.05, 0.1))
)
```

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).
sample_split	Logical indicating whether to perform inference using sample splitting
m	Number of sample-splitting folds, defaults to 5.
eval_region	Left and right endpoints of region over which to estimate the survival function. Alternatively, provide eval_grid to explicitly specify the grid of points on 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.
eval_grid	Grid of time points at which to estimate the survival function. This is an alternative in place of specifying eval_region.
alpha	The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05
sensitivity_analysis	Logical, whether to perform a copula-based sensitivity analysis. Defaults to FALSE
copula_control	A named list of control parameters for the copula-based sensitivity analysis. This should be a named list.

Value

List of data frames giving results. If not performing a sensitivity analysis, a single data frame is returned; if performing a sensitivity analysis, a separate data frame will be returned for each value of the copula association parameter. The results data frames have 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

References

Wolock C.J., et al. (2025). "Investigating symptom duration using current status data: a case study of post-acute COVID-19 syndrome."

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),
  sensitivity_analysis = FALSE,
  eval_region = eval_region)$primary_results

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

Doubly-robust pseudo-outcome regression

Description

Generate estimates of conditional survival probability or conditional restricted mean survival time 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. For outcome "survival_probability", this is the times at which the survival function is to be estimated. For outcome "restricted_survival_time", this is simply the restriction time.
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 corresponding to newX and newtimes.

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))

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)

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

approx_times <- c(0, sort(unique(time)))

# estimate conditional survival functions at approx_times
fit <- stackG(time = time,
              event = event,
              X = X,
              newX = X,
              newtimes = approx_times,
              direction = "prospective",
              bin_size = 0.1,
              time_basis = "continuous",
              surv_form = "PI",
              learner = "SuperLearner",
              time_grid_approx = approx_times,
              SL_control = list(SL.library = SL.library,
                                V = 3))

# use DR pseudo-outcome regression to (robustly) estimate survival at t = 5
DR_preds <- DR_pseudo_outcome_regression(time = time,
                                         event = event,
                                         X = X,
                                         newX = X,
                                         newtimes = 5,
                                         approx_times = approx_times,
                                         S_hat = fit$S_T_preds,
                                         G_hat = fit$S_C_preds,
                                         outcome = "survival_probability",
                                         SL.library = SL.library,
                                         V = 3)

DR_preds

```

generate_folds

Generate cross-fitting and sample-splitting folds

Description

Generate cross-fitting and sample-splitting folds

Usage

```
generate_folds(n, V, sample_split, cf_folds = NULL)
```

Arguments

n	Total sample size
V	Number of cross-fitting folds to use
sample_split	Logical, whether or not sample-splitting is being used
cf_folds	Cross-fitting folds, if already provided by user.

Value

Named list of cross-fitting and sample-splitting folds

generate_nuisance_predictions_stackG

Estimate conditional survival function nuisance parameters using survival stacking

Description

Estimate conditional survival function nuisance parameters using survival stacking

Usage

```
generate_nuisance_predictions_stackG(
  time,
  event,
  X,
  X_holdout,
  newtimes,
  SL.library = c("SL.mean", "SL.glm", "SL.earth", "SL.gam", "SL.ranger"),
  V = 5,
  bin_size = 0.05,
  approx_times
)
```

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
X_holdout	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.

SL.library	Super Learner library
V	Number of cross-validation folds, to be passed to SuperLearner
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. See stackG documentation.
approx_times	Numeric vector of times at which to approximate product integral or cumulative hazard interval. See stackG documentation.

Value

A list containing elements S_hat (conditional event survival function, corresponding to X_holdout and newtimes), S_hat_train (conditional event survival function, corresponding to X and newtimes), G_hat (conditional censoring survival function, corresponding to X_holdout and newtimes), and G_hat_train (conditional censoring survival function, corresponding to X and newtimes)

See Also

[stackG](#)

generate_oracle_predictions_boost

Estimate oracle prediction function using DR gradient boosting

Description

Estimate oracle prediction function using DR gradient boosting

Usage

```
generate_oracle_predictions_boost(
  time,
  event,
  X,
  X_holdout,
  nuisance_preds,
  restriction_time,
  approx_times,
  V = 5,
  indx,
  tuning = FALSE,
  subsample_n = length(time),
  boosting_params = list(mstop = c(100), nu = c(0.1), sigma = c(0.01), learner =
    c("glm"))
)
```

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
X_holdout	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.
nuisance_preds	Named list of conditional survival function predictions with elements "S_hat", "S_hat_train", "G_hat", and "G_hat_train". This should match the output of conditional_surv_generator.
restriction_time	Maximum follow-up time for calculation of the C-index. Essentially, this time should be chosen such that the conditional survival function is identified at this time for all covariate values X present in the data. Choosing the restriction time such that roughly 10% of individuals remain at-risk at that time has been shown to work reasonably well in simulations.
approx_times	Numeric vector of length J2 giving times at which to approximate C-index integral.
V	Number of cross-validation folds for selection of tuning parameters
indx	Numeric index of column(s) of X to be removed, i.e., not used in the oracle prediction function.
tuning	Logical, whether or not to use cross-validation to select tuning parameters
subsample_n	Number of samples to use for boosting procedure. Using a subsample of the full sample can greatly reduce runtime
boosting_params	Named list of parameter values for the boosting procedure. Elements of this list include mstop (number of boosting iterations), nu (learning rate), sigma (smoothness parameter for sigmoid approximation, with smaller meaning less smoothing), and learner (base learner, can take values "glm", "gam", or "tree")

Value

A list containing elements `f0_hat` and `f0_hat_train`, the estimated oracle prediction functions for `X_holdout` and `X`, respectively.

See Also

[boost_c_index](#)

generate_oracle_predictions_DR

Estimate full oracle prediction function using DR pseudo-outcome regression

Description

Estimate full oracle prediction function using DR pseudo-outcome regression

Usage

```
generate_oracle_predictions_DR(
  time,
  event,
  X,
  X_holdout,
  nuisance_preds,
  outcome,
  landmark_times,
  restriction_time,
  approx_times,
  SL.library = c("SL.mean", "SL.glm", "SL.earth", "SL.gam", "SL.ranger"),
  V = 5,
  indx
)
```

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
X_holdout	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.
nuisance_preds	Named list of conditional survival function predictions with elements "S_hat", "S_hat_train", "G_hat", and "G_hat_train". This should match the output of conditional_surv_generator.
outcome	Outcome type, either "survival_probability" or "restricted_survival_time"
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 "survival_time_MSE". Essentially, this time should be chosen such that the conditional survival function is identified at this time for all covariate values X present in the data. Choosing the

	restriction time such that roughly 10% of individuals remain at-risk at that time has been shown to work reasonably well in simulations.
approx_times	Numeric vector of length J2 giving times at which to approximate integral appearing in the pseudo-outcomes
SL.library	Super Learner library
V	Number of cross-validation folds, to be passed to SuperLearner
indx	Numeric index of column(s) of X to be removed, i.e., not used in the oracle prediction function.

Value

A list containing elements $f_{\hat{0}}$ and $f_{\hat{0}_{\text{train}}}$, the estimated oracle prediction functions for X_{holdout} and X , respectively.

See Also

[DR_pseudo_outcome_regression](#)

generate_oracle_predictions_SL

Estimate small oracle prediction function using Super Learner regression

Description

When the oracle prediction function is a conditional mean, the small oracle prediction function can be estimated by regressing the large oracle prediction function on the small feature vector. This function performs this regression using Super Learner.

Usage

```
generate_oracle_predictions_SL(
  time,
  event,
  X,
  X_holdout,
  nuisance_preds,
  outcome,
  landmark_times,
  restriction_time,
  approx_times,
  SL.library = c("SL.mean", "SL.glm", "SL.earth", "SL.gam", "SL.ranger"),
  V = 5,
  indx
)
```

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
X_holdout	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.
nuisance_preds	Named list of conditional survival function predictions with elements "S_hat", "S_hat_train", "G_hat", and "G_hat_train". This should match the output of conditional_surv_generator.
outcome	Outcome type, either "survival_probability" or "restricted_survival_time"
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 "survival_time_MSE". Essentially, this time should be chosen such that the conditional survival function is identified at this time for all covariate values X present in the data. Choosing the restriction time such that roughly 10% of individuals remain at-risk at that time has been shown to work reasonably well in simulations.
approx_times	Numeric vector of length J2 giving times at which to approximate integral appearing in the pseudo-outcomes
SL.library	Super Learner library
V	Number of cross-validation folds, to be passed to SuperLearner
indx	Numeric index of column(s) of X to be removed, i.e., not used in the oracle prediction function.

Value

A list containing elements `f0_hat` and `f0_hat_train`, the estimated small oracle prediction functions for `X_holdout` and `X`, respectively.

multiseed_vim

Estimate variable importance with multiple seeds

Description

Repeat the VIM estimation procedure multiple times and aggregate the results, mitigating the additional randomness introduced by sample-splitting and cross-fitting.

Usage

```

multiseed_vim(
  n_seed,
  agg_method = "compound_bg",
  ci_grid,
  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_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_fold_num = 5,
  sample_split = TRUE,
  scale_est = FALSE,
  alpha = 0.05,
  verbose = FALSE
)

```

Arguments

<code>n_seed</code>	Number of iterations (seeds) to perform the VIM estimation procedure. These will be aggregated into a single result.
<code>agg_method</code>	P-value aggregation method use to combine results from different seeds. Current options are "bonferroni" (Bonferroni's method), "hommel" (Hommel's method), "arithmetic" (arithmetic mean), "geometric" (geometric mean), "harmonic" (harmonic mean), "compound_bg" (compound Bonferroni and geometric mean), and "compound_ba" (compound Bonferroni and arithmetic mean). These approaches are discussed at length in Vovk and Wang (2020). Defaults to "compound_bg", which has been shown to work well in many settings.
<code>ci_grid</code>	Grid of VIM values over which to construct a confidence interval by inverting a hypothesis test. The aggregation works by constructing hypothesis tests (at level α) of the null corresponding to each value in <code>ci_grid</code> , and then inverting these tests to yield a $1 - \alpha$ confidence interval. For example, for "AUC" importance, the VIM takes values in (0,1), so a grid of values between 0 and 1 would be a reasonable choice.
<code>type</code>	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". Essentially, this time should be chosen such that the conditional survival function is identified at this time for all covariate values X present in the data. Choosing the restriction time such that roughly 10% of individuals remain at-risk at that time has been shown to work reasonably well in simulations.
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_generator	A function to estimate the conditional survival functions of the event and censoring variables. Must take arguments (time, event, X) (for training purposes) and (X_holdout and newtimes) (covariate values and times at which to generate predictions). Defaults to generate_nuisance_predictions_stackG , a pre-built generator function based on the stackG function. Alternatively, the user can provide their own function for this argument, or provide pre-computed estimates to conditional_surv_preds in lieu of this argument.
conditional_surv_generator_control	A list of arguments to pass to conditional_surv_generator.
large_oracle_generator	A function to estimate the oracle prediction function using large_feature_vector. Must take arguments time, event, X, X_holdout, and nuisance_preds. For all VIM types except for "C-index", defaults to generate_oracle_predictions_DR , a pre-built generator function using doubly-robust pseudo-outcome regression. For "C-index", defaults to generate_oracle_predictions_boost , a pre-built generator function using doubly-robust gradient boosting. Alternatively, the user can provide their own function, or provide pre-computed estimates to large_oracle_preds in lieu of this argument.
large_oracle_generator_control	A list of arguments to pass to large_oracle_generator.
small_oracle_generator	A function to estimate the oracle prediction function using small_feature_vector. Must take arguments time, event, X, X_holdout, and nuisance_preds. For all VIM types except for "C-index", defaults to generate_oracle_predictions_SL ,

a pre-built generator function based on regression the large oracle predictions on the small feature vector. For "C-index", defaults to [generate_oracle_predictions_boost](#), a pre-built generator function using doubly-robust gradient boosting. Alternatively, the user can provide their own function, or provide pre-computed estimates to `small_oracle_preds` in lieu of this argument.

<code>small_oracle_generator_control</code>	A list of arguments to pass to <code>small_oracle_generator</code> .
<code>cf_fold_num</code>	The number of cross-fitting folds, if not providing <code>cf_folds</code> . Note that with <code>samples-splitting</code> , the data will be split into $2 \times \text{cf_fold_num}$ folds (i.e., there will be <code>cf_fold_num</code> folds within each half of the data).
<code>sample_split</code>	Logical indicating whether or not to sample split. Sample-splitting is required for valid hypothesis testing of null importance and is generally recommended. Defaults to TRUE.
<code>scale_est</code>	Logical, whether or not to force the VIM estimate to be nonnegative.
<code>alpha</code>	The level at which to compute confidence intervals and hypothesis tests. Defaults to 0.05.
<code>verbose</code>	Whether to print progress messages.

Details

Using a larger value of `n_seed` will result in more stable results, at a greater computational cost.

Value

Named list with the following elements:

<code>agg_result</code>	Data frame giving results aggregated over seeds.
<code>agg_method</code>	P-value aggregation method used.
<code>n_seed</code>	Number of iterations (seeds) used to perform the VIM estimation procedure.
<code>vim_objects</code>	A list of <code>vim</code> return objects, each corresponding to a different seed.

References

- Vovk V. and Wang R. (2020). "Combining p-values via averaging."
- Wolock C.J., Gilbert P.B., Simon N., and Carone, M. (2025). "Assessing variable importance in survival analysis using machine learning."

See Also

[vim](#)

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 <- multiseed_vim(n_seed = 2,
  agg_method = "compound_bg",
  ci_grid = seq(0, 1, by = 0.01),
  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 = TRUE,
  scale_est = TRUE)

print(output$result)

```

predict.stackG	<i>Obtain predicted conditional survival and cumulative hazard functions from a global survival stacking object</i>
----------------	---

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

<code>object</code>	Object of class <code>stackG</code>
<code>newX</code>	$m \times p$ 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 <code>X</code> .
<code>newtimes</code>	$k \times 1$ numeric vector of times at which to obtain k predicted conditional survivals.
<code>surv_form</code>	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 <code>object</code> .
<code>time_grid_approx</code>	Numeric vector of times at which to approximate product integral or cumulative hazard interval. Defaults to the value saved in <code>object</code> .
<code>...</code>	Further arguments passed to or from other methods.

Value

A named list with the following components:

<code>S_T_preds</code>	An $m \times k$ matrix of estimated event time survival probabilities at the m covariate vector values and k times provided by the user in <code>newX</code> and <code>newtimes</code> , respectively.
<code>S_C_preds</code>	An $m \times k$ matrix of estimated censoring time survival probabilities at the m covariate vector values and k times provided by the user in <code>newX</code> and <code>newtimes</code> , respectively.
<code>Lambda_T_preds</code>	An $m \times k$ matrix of estimated event time cumulative hazard function values at the m covariate vector values and k times provided by the user in <code>newX</code> and <code>newtimes</code> , respectively.
<code>Lambda_C_preds</code>	An $m \times k$ matrix of estimated censoring time cumulative hazard function values at the m covariate vector values and k times provided by the user in <code>newX</code> and <code>newtimes</code> , respectively.
<code>time_grid_approx</code>	The approximation grid for the product integral or cumulative hazard integral, (user-specified).
<code>surv_form</code>	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	<i>Obtain predicted conditional survival function from a local survival stacking object</i>
----------------	---

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')

```

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.

<code>bin_size</code>	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.
<code>time_basis</code>	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.
<code>time_grid_approx</code>	Numeric vector of times at which to approximate product integral or cumulative hazard interval. Defaults to <code>times</code> argument.
<code>surv_form</code>	Mapping from hazard estimate to survival estimate. Can be either "PI" (product integral mapping) or "exp" (exponentiated cumulative hazard estimate).
<code>learner</code>	Which binary regression algorithm to use. Currently, only SuperLearner is supported, but more learners will be added. See below for algorithm-specific arguments.
<code>SL_control</code>	Named list of parameters controlling the Super Learner fitting process. These parameters are passed directly to the SuperLearner function. Parameters include <code>SL.library</code> (library of algorithms to include in the binary classification Super Learner), <code>V</code> (Number of cross validation folds on which to train the Super Learner classifier, defaults to 10), <code>method</code> (Method for estimating coefficients for the Super Learner, defaults to "method.NNLS"), <code>stratifyCV</code> (logical indicating whether to stratify by outcome in SuperLearner's cross-validation scheme), and <code>obsWeights</code> (observation weights, passed directly to prediction algorithms by SuperLearner).
<code>tau</code>	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 $\tau - \text{time}$. 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:

<code>S_T_preds</code>	An $m \times k$ matrix of estimated event time survival probabilities at the m covariate vector values and k times provided by the user in <code>newX</code> and <code>newtimes</code> , respectively.
<code>S_C_preds</code>	An $m \times k$ matrix of estimated censoring time survival probabilities at the m covariate vector values and k times provided by the user in <code>newX</code> and <code>newtimes</code> , respectively.
<code>Lambda_T_preds</code>	An $m \times k$ matrix of estimated event time cumulative hazard function values at the m covariate vector values and k times provided by the user in <code>newX</code> and <code>newtimes</code> , respectively.
<code>Lambda_C_preds</code>	An $m \times k$ matrix of estimated censoring time cumulative hazard function values at the m covariate vector values and k times provided by the user in <code>newX</code> and <code>newtimes</code> , respectively.

<code>time_grid_approx</code>	The approximation grid for the product integral or cumulative hazard integral, (user-specified).
<code>direction</code>	Whether the data come from a prospective or retrospective study (user-specified).
<code>tau</code>	The maximum time of interest in a study, used for retrospective conditional survival estimation (user-specified).
<code>surv_form</code>	Exponential or product-integral form (user-specified).
<code>time_basis</code>	Whether time is included in the regression as continuous or dummy (user-specified).
<code>SL_control</code>	Named list of parameters controlling the Super Learner fitting process (user-specified).
<code>fits</code>	A named list of fitted regression objects corresponding to the constituent regressions needed for global survival stacking. Includes <code>P_Delta</code> (probability of event given covariates), <code>F_Y_1</code> (conditional cdf of follow-up times given covariates among uncensored), <code>F_Y_0</code> (conditional cdf of follow-up times given covariates among censored), <code>G_W_1</code> (conditional distribution of entry times given covariates and follow-up time among uncensored), <code>G_W_0</code> (conditional distribution of entry times given covariates and follow-up time among censored). 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 $\tau - \text{time}$. 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 $m \times k$ 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 variable importance

Description

Compute estimates of and confidence intervals for nonparametric variable importance based on the difference predictiveness obtained with and without the feature of interest. Designed for use with time-to-event outcomes subject to right censoring that may be informed by measured covariates.

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_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,
conditional_surv_preds = NULL,
large_oracle_preds = NULL,
small_oracle_preds = 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". Essentially, this time should be chosen such that the conditional survival function is identified at this time for all covariate values X present in the data. Choosing the restriction time such that roughly 10% of individuals remain at-risk at that time has been shown to work reasonably well in simulations.
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_generator	A function to estimate the conditional survival functions of the event and censoring variables. Must take arguments (time, event, X) (for training purposes) and (X_holdout and newtimes) (covariate values and times at which to generate predictions). Defaults to generate_nuisance_predictions_stackG , a pre-built

generator function based on the [stackG](#) function. Alternatively, the user can provide their own function for this argument, or provide pre-computed estimates to `conditional_surv_preds` in lieu of this argument.

`conditional_surv_generator_control`

A list of arguments to pass to `conditional_surv_generator`.

`large_oracle_generator`

A function to estimate the oracle prediction function using `large_feature_vector`.

Must take arguments `time`, `event`, `X`, `X_holdout`, and `nuisance_preds`. For all VIM types except for "C-index", defaults to [generate_oracle_predictions_DR](#), a pre-built generator function using doubly-robust pseudo-outcome regression. For "C-index", defaults to [generate_oracle_predictions_boost](#), a pre-built generator function using doubly-robust gradient boosting. Alternatively, the user can provide their own function, or provide pre-computed estimates to `large_oracle_preds` in lieu of this argument.

`large_oracle_generator_control`

A list of arguments to pass to `large_oracle_generator`.

`small_oracle_generator`

A function to estimate the oracle prediction function using `small_feature_vector`.

Must take arguments `time`, `event`, `X`, `X_holdout`, and `nuisance_preds`. For all VIM types except for "C-index", defaults to [generate_oracle_predictions_SL](#), a pre-built generator function based on regression the large oracle predictions on the small feature vector. For "C-index", defaults to [generate_oracle_predictions_boost](#), a pre-built generator function using doubly-robust gradient boosting. Alternatively, the user can provide their own function, or provide pre-computed estimates to `small_oracle_preds` in lieu of this argument.

`small_oracle_generator_control`

A list of arguments to pass to `small_oracle_generator`.

`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 `conditional_surv_generator` functionality 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`. If using sample splitting, each of these is itself a list of length $2K$, where K is the number of cross-fitting folds (if not using sample splitting, each is a list of length K). Each element of these lists is a matrix with J_2 columns and number of rows equal to either the number of samples in the k th fold (for `S_hat` and `G_hat`) or the number of samples used to compute the nuisance estimates for the k th fold (for `S_hat_train` and `G_hat_train`).

`large_oracle_preds`

User-provided estimates of the oracle prediction function using `large_feature_vector` (if not using the `large_oracle_generator` functionality to compute these nuisance estimates). Must be a named list of lists with elements `f0_hat` and `f0_hat_train`. If using sample splitting, each of these is itself a list of length $2K$ (if not using sample splitting, each is a list of length K). Each element of these lists is a matrix with J_1 columns (for landmark time VIMs) or 1 column (for "C-index" and "survival_time_MSE") and number of rows equal to either the number of samples in the k th fold (for `f0_hat`) or the number of samples used to compute the nuisance estimates for the k th fold (for `f0_hat_train`).

<code>small_oracle_preds</code>	User-provided estimates of the oracle prediction function using <code>small_feature_vector</code> (if not using the <code>small_oracle_generator</code> functionality to compute these nuisance estimates). Must be a named list of lists with elements <code>f0_hat</code> and <code>f0_hat_train</code> . If using sample splitting, each of these is itself a list of length <code>2K</code> (if not using sample splitting, each is a list of length <code>K</code>). Each element of these lists is a matrix with <code>J1</code> columns (for landmark time VIMs) or 1 column (for "C-index" and "survival_time_MSE") and number of rows equal to either the number of samples in the <code>k</code> th fold (for <code>f0_hat</code>) or the number of samples used to compute the nuisance estimates for the <code>k</code> th fold (for <code>f0_hat_train</code>).
<code>cf_folds</code>	Numeric vector of length <code>n</code> giving cross-fitting folds, if specifying the folds explicitly. This is required if you are providing pre-computed nuisance estimations — if providing a nuisance generator function, the <code>vim()</code> will assign folds.
<code>cf_fold_num</code>	The number of cross-fitting folds, if not providing <code>cf_folds</code> . Note that with samples-splitting, the data will be split into $2 \times \text{cf_fold_num}$ folds (i.e., there will be <code>cf_fold_num</code> folds within each half of the data).
<code>sample_split</code>	Logical indicating whether or not to sample split. Sample-splitting is required for valid hypothesis testing of null importance and is generally recommended. Defaults to <code>TRUE</code> .
<code>ss_folds</code>	Numeric vector of length <code>n</code> giving sample-splitting folds, if specifying the folds explicitly. This is required if you are providing pre-computed nuisance estimations — if providing a nuisance generator function, the <code>vim()</code> will assign folds.
<code>robust</code>	Logical, whether or not to use the doubly-robust debiasing approach. This option is meant for illustration purposes only — it should be left as <code>TRUE</code> .
<code>scale_est</code>	Logical, whether or not to force the VIM estimate to be nonnegative.
<code>alpha</code>	The level at which to compute confidence intervals and hypothesis tests. Defaults to <code>0.05</code> .
<code>verbose</code>	Whether to print progress messages.

Details

For nuisance estimation, it is generally advisable to use the pre-built nuisance generator functions provided by `survML`. See the "Variable importance in survival analysis" vignette, or the [package website](#) for an illustration.

Value

Named list with the following elements:

<code>result</code>	Data frame giving results. See the documentation of the individual <code>vim_*</code> functions for details.
<code>folds</code>	A named list giving the cross-fitting fold IDs (<code>cf_folds</code>) and sample-splitting fold IDs (<code>ss_folds</code>).
<code>approx_times</code>	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.

References

Wolock C.J., Gilbert P.B., Simon N., and Carone, M. (2025). "Assessing variable importance in survival analysis using machine learning."

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	<i>Estimate AUC VIM</i>
---------	-------------------------

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	<i>Estimate concordance index VIM</i>
------------	---------------------------------------

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

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:

<code>restriction_time</code>	Restriction time (upper bound for event times to be compared in computing the restricted survival time).
<code>est</code>	VIM point estimate.
<code>var_est</code>	Estimated variance of the VIM estimate.
<code>cil</code>	Lower bound of the VIM confidence interval.
<code>ciu</code>	Upper bound of the VIM confidence interval.
<code>cil_1sided</code>	Lower bound of a one-sided confidence interval.
<code>p</code>	p-value corresponding to a hypothesis test of null importance.
<code>large_predictiveness</code>	Estimated predictiveness of the large oracle prediction function.
<code>small_predictiveness</code>	Estimated predictiveness of the small oracle prediction function.
<code>vim</code>	VIM type.
<code>large_feature_vector</code>	Group of features available for the large oracle prediction function.
<code>small_feature_vector</code>	Group of features available for the small oracle prediction function.

See Also

[vim](#) for example usage

Index

aggregate_vim, [2](#)

boost_c_index, [3](#), [14](#)

crossfit_oracle_preds, [5](#)
crossfit_surv_preds, [6](#)
currstatCIR, [7](#)

DR_pseudo_outcome_regression, [9](#), [16](#)

generate_folds, [11](#)
generate_nuisance_predictions_stackG,
[12](#), [19](#), [33](#)
generate_oracle_predictions_boost, [13](#),
[19](#), [20](#), [34](#)
generate_oracle_predictions_DR, [15](#), [19](#),
[34](#)
generate_oracle_predictions_SL, [16](#), [19](#),
[34](#)

multiseed_vim, [17](#)

predict.stackG, [21](#), [28](#)
predict.stackL, [24](#), [31](#)

stackG, [13](#), [19](#), [22](#), [25](#), [34](#)
stackL, [24](#), [29](#)

vim, [20](#), [32](#), [39](#), [41](#), [43](#), [44](#), [46](#)
vim_accuracy, [36](#), [37](#)
vim_AUC, [36](#), [38](#)
vim_brier, [36](#), [40](#)
vim_cindex, [36](#), [41](#)
vim_rsquared, [36](#), [43](#)
vim_survival_time_mse, [36](#), [45](#)