Package: targeted (via r-universe)

October 20, 2026

Type Package

Title Targeted Inference

Version 0.9.0

Author Klaus K. Holst [aut, cre], Benedikt Sommer [aut], Andreas Nordland [aut], Christian B. Pipper [ctb]

Maintainer Klaus K. Holst <klaus@holst.it>

Description Various methods for targeted and semiparametric inference including augmented inverse probability weighted (AIPW) estimators for missing data and causal inference (Bang and Robins (2005) <doi:10.1111/j.1541-0420.2005.00377.x>), one-step imputation (Nordland et al (2026)) <doi:10.48550/arXiv.2606.07174>), variable importance and conditional average treatment effects (CATE) (van der Laan (2006) <doi:10.2202/1557-4679.1008>), estimators for risk differences and relative risks (Richardson et al. (2017) <doi:10.1080/01621459.2016.1192546>), assumption lean inference for generalized linear model parameters (Vansteelandt et al. (2022) <doi:10.1111/rssb.12504>).

Depends R (>= 4.1)

Imports R6, Rcpp (>= 1.1.1), abind, cli, future.apply, lava (>= 1.9.2), methods, mets (>= 1.3.12), quadprog, progressr, rlang, survival

Suggests SuperLearner (>= 2.0-28), MASS, cmprsk, data.table, e1071, earth, glmnet, grf, hal9001, mgcv, nnls, optimx, polle (>= 1.5), pracma, quarto, randomForestSRC, ranger, riskRegression, scatterplot3d, tinytest, viridisLite, xgboost (>= 3.1.2.1),

BugReports https://github.com/kkholst/targeted/issues

URL https://kkholst.github.io/targeted/, https://github.com/kkholst/targeted

License Apache License (== 2.0)

LinkingTo Rcpp, RcppArmadillo

LazyLoad yes

NeedsCompilation yes

ByteCompile yes

RcppModules riskregmodel

Encoding UTF-8

VignetteBuilder quarto

SystemRequirements Quarto command line tools (https://github.com/quarto-dev/quarto-cli).

Config/roxygen2/version 8.1.0

Repository https://cran.r-universe.dev

Date/Publication 2026-08-21 09:50:02 UTC

RemoteUrl https://github.com/cran/targeted

RemoteRef HEAD

RemoteSha 3a58eb7d88b29975d329d41b94838f51f79d0c4c

Contents

aipw . . . 4
alean . . . 4
ate . . . 6
calibration . . . 8
calibration-class . . . 10
cate . . . 11
cross_validated-class . . . 14
cumhaz . . . 15
cv.default . . . 16
cv.learner_sl . . . 18
deprecate_arg_warn . . . 19
design . . . 19
estimate_truncatedscore . . . 20
expand.list . . . 22
fit_survival_models . . . 23
int_surv . . . 24
learner . . . 24
learner_expand_grid . . . 29
learner_gam . . . 30
learner_glm . . . 31
learner_glmnet_cv . . . 32
learner_grf . . . 34
learner_hal . . . 36
learner_isoreg . . . 37
learner_mars . . . 38
learner_naivebayes . . . 39
learner_sl . . . 40

learner_stratify . . . 42
learner_surv_cox . . . 43
learner_surv_rf . . . 44
learner_svm . . . 45
learner_xgboost . . . 46
metalearner_convexcomb . . . 48
metalearner_discrete . . . 48
metalearner_nnls . . . 49
moi . . . 50
naivebayes . . . 53
naivebayes-class . . . 54
nondom . . . 55
pava . . . 56
predict.density . . . 57
predict.naivebayes . . . 57
predict.superlearner . . . 58
predict_filter_bound . . . 58
predict_filter_bound_dynamic . . . 59
RATE . . . 60
RATE.surv . . . 61
rcai . . . 63
riskreg . . . 64
riskreg_cens . . . 66
score.superlearner . . . 68
scoring . . . 68
SL . . . 69
softmax . . . 70
solve_ode . . . 70
specify_ode . . . 71
stratify . . . 72
superlearner . . . 73
survival_treatment_level_estfun . . . 75
targeted-class . . . 76
terms.design . . . 77
test_intersection_sw . . . 77
test_zmax_onesided . . . 79
truncatedscore . . . 80
weights.superlearner . . . 81

Index **82**

aipw *AIPW estimator*

Description

AIPW for the mean (and linear projections of the EIF) with missing observations

Usage

```
aipw(response.model, propensity.model, formula = ~1, data, ...)
```

Arguments

`response.model`	(learner or formula) Model for the response given covariates
`propensity.model`	(learner or formula) Missing data mechanism model and if omitted a logistic regression model with the same covariates as `response.model` is used
`formula`	design specifying the OLS estimator with outcome given by the EIF (see `cate`)
`data`	data.frame
`...`	additional arguments (see `cate()`)

Examples

```
m <- lava::lvm(y ~ x+z, r ~ x) |>
     lava::distribution(~ r, value = lava::binomial.lvm()) |>
     transform(y0~r+y, value = \(x) { x[x[,1]==0,2] <- NA; x[,2] })
d <- lava::sim(m,5e3,seed=1)

aipw(y0 ~ x, ~ x + z, data=d)
```

alean *Assumption Lean inference for generalized linear model parameters*

Description

Assumption lean inference via cross-fitting (Double ML). See <doi:10.1111/rssb.12504

Usage

```
alean(
  response_model,
  exposure_model,
  data,
  link = "identity",
  g_model,
```

```
  nfolds = 1,
  silent = FALSE,
  mc.cores,
  ...
)
```

Arguments

response_model	formula or learner object (formula => glm)
exposure_model	model for the exposure
data	data.frame
link	Link function (g)
g_model	Model for $E[g(Y\|A,W)\|W]$
nfolds	Number of folds
silent	supress all messages and progressbars
mc.cores	mc.cores Optional number of cores. parallel::mcmapply used instead of future
...	additional arguments to future.apply::future_mapply

Details

Let Y be the response variable, A the exposure and W covariates. The target parameter is:

$$\Psi(P) = \frac{E(Cov[A, g\{E(Y|A,W)\} \mid W])}{E\{Var(A \mid W)\}}$$

The response_model is the model for $E(Y|A,W)$, and exposure_model is the model for $E(A|W)$. link specifies g.

Value

alean.targeted object

Author(s)

Klaus Kähler Holst

Examples

```
sim1 <- function(n, family=gaussian(), ...) {
   m <- lava::lvm() |>
     lava::distribution(~y, value=lava::binomial.lvm()) |>
     lava::regression('a', value=function(l) l) |>
     lava::regression('y', value=function(a,l) a + l)
     if (family$family=="binomial")
        lava::distribution(m, ~a) <- lava::binomial.lvm()
   lava::sim(m, n)
}
```

```
library(splines)
f <- binomial()
d <- sim1(1e4, family=f)
e <- alean(
 response_model=learner_glm(y ~ a + bs(l, df=3), family=binomial),
 exposure_model=learner_glm(a ~ bs(l, df=3), family=f),
 data=d,
 link = "logit", mc.cores=1, nfolds=1
)
e

e <- alean(response_model=learner_glm(y ~ a + l, family=binomial),
           exposure_model=learner_glm(a ~ l),
           data=d,
           link = "logit", mc.cores=1, nfolds=1)
e
```

---

ate *AIPW (doubly-robust) estimator for Average Treatment Effect*

Description

Augmented Inverse Probability Weighting estimator for the Average (Causal) Treatment Effect. All nuisance models are here parametric (glm). For a more general approach see the `cate` implementation. In this implementation the standard errors are correct even when the nuisance models are mis-specified (the influence curve is calculated including the term coming from the parametric nuisance models). The estimate is consistent if either the propensity model or the outcome model / Q-model is correctly specified.

Usage

```
ate(
  formula,
  data = parent.frame(),
  weights,
  offset,
  family = stats::gaussian(identity),
  nuisance = NULL,
  propensity = nuisance,
  all,
  labels = NULL,
  adjust.nuisance = TRUE,
  adjust.propensity = TRUE,
  ...
)
```

Arguments

formula	formula (see details below)
data	data.frame
weights	optional frequency weights
offset	optional offset (character or vector). can also be specified in the formula.
family	Exponential family argument for outcome model
nuisance	outcome regression formula (Q-model)
propensity	propensity model formula
all	when TRUE all standard errors are calculated (default TRUE when exposure only has two levels)
labels	optional treatment labels
adjust.nuisance	adjust for uncertainty due to estimation of outcome regression model parameters
adjust.propensity	adjust for uncertainty due to estimation of propensity regression model parameters
...	additional arguments to lower level functions

Details

The formula may either be specified as: response ~ treatment | nuisance-formula | propensity-formula

For example: `ate(y~a | x+z+a | x*z, data=...)`

Alternatively, as a list: `ate(list(y~a, ~x+z, ~x*z), data=...)`

Or using the nuisance (and propensity argument): `ate(y~a, nuisance=~x+z, ...)`

Value

An object of class 'ate.targeted' is returned. See targeted-class for more details about this class and its generic functions.

Author(s)

Klaus K. Holst

See Also

cate

Examples

```
m <- lava::lvm(y ~ a+x, a~x) |>
     lava::distribution(~y, value = lava::binomial.lvm()) |>
     lava::ordinal(K=4, ~a) |>
     transform(~a, value = factor)
d <- lava::sim(m, 1e3, seed=1)
# (a <- ate(y~a|a*x|x, data=d))
(a <- ate(y~a, nuisance=~a*x, propensity=~x, data = d))

# Comparison with randomized experiment
m0 <- lava::cancel(m, a~x)
lm(y~a-1, lava::sim(m0,2e4))

# Choosing a different contrast for the association measures
summary(a, contrast=c(2,4))
```

---

calibration *Calibration (training)*

Description

Calibration for multiclassication methods

Usage

```
calibration(
  pr,
  cl,
  weights = NULL,
  threshold = 10,
  method = "bin",
  breaks = nclass.Sturges,
  df = 3,
  ...
)
```

Arguments

pr	matrix with probabilities for each class
cl	class variable
weights	counts
threshold	do not calibrate if less then 'threshold' events
method	either 'isotonic' (pava), 'logistic', 'mspline' (monotone spline), 'bin' (local constant)
breaks	optional number of bins (only for method 'bin')
df	degrees of freedom (only for spline methods)
...	additional arguments to lower level functions

Details

...

Value

An object of class 'calibration' is returned. See calibration-class for more details about this class and its generic functions.

Author(s)

Klaus K. Holst

Examples

```
sim1 <- function(n, beta=c(-3, rep(.5,10)), rho=.5) {
 p <- length(beta)-1
 xx <- lava::rmvn0(n,sigma=diag(nrow=p)*(1-rho)+rho)
 y <- rbinom(n, 1, lava::expit(cbind(1,xx)%*%beta))
 d <- data.frame(y=y, xx)
 names(d) <- c("y",paste0("x",1:p))
 return(d)
}

set.seed(1)
beta <- c(-2,rep(1,10))
d <- sim1(1e4, beta=beta)
a1 <- naivebayes(y ~ ., data=d)
a2 <- glm(y ~ ., data=d, family=binomial)
## a3 <- randomForest(factor(y) ~ ., data=d, family=binomial)

d0 <- sim1(1e4, beta=beta)
p1 <- predict(a1, newdata=d0)
p2 <- predict(a2, newdata=d0, type="response")
## p3 <- predict(a3, newdata=d0, type="prob")

c2 <- calibration(p2, d0$y, method="isotonic")
c1 <- calibration(p1, d0$y, breaks=100)
if (interactive()) {
  plot(c1)
  plot(c2,col="red",add=TRUE)
  abline(a=0,b=1)
  with(c1$xy[[1]], points(pred,freq,type="b", col="red"))
}

set.seed(1)
beta <- c(-2,rep(1,10))
dd <- lava::csplit(sim1(1e4, beta=beta), k=3)
mod <- naivebayes(y ~ ., data=dd[[1]])
p1 <- predict(mod, newdata=dd[[2]])
cal <- calibration(p1, dd[[2]]$y)
p2 <- predict(mod, newdata=dd[[3]])
pp <- predict(c1, p2)
```

```
cc <- calibration(pp, dd[[3]]$y)
if (interactive()) {#'
  plot(cal)
  plot(cc, add=TRUE, col="blue")
}
```

`calibration-class` *calibration class object*

Description

The functions `calibration` returns an object of the class `calibration`.

An object of class '`calibration`' is a list with at least the following components:

stepfun estimated step-functions (see `stepfun`) for each class

classes the unique classes

model model/method type (string)

xy list of data.frame's with predictions (pr) and estimated probabilities of success (only for 'bin' method)

Value

objects of the S3 class '`calibration`'

S3 generics

The following S3 generic functions are available for an object of class `calibration`:

`predict` Apply calibration to new data.

`plot` Plot the calibration curves (reliability plot).

`print` Basic print method.

See Also

`calibration`, `calibrate`

Examples

```
## See example(calibration) for examples
```

cate *Conditional Average Treatment Effect estimation*

Description

Conditional Average Treatment Effect estimation with cross-fitting.

Usage

```
cate(
  response.model,
  treatment.model,
  cate.model = ~1,
  calibration.model = NULL,
  missing.model = NULL,
  data,
  contrast,
  nfolds = 1,
  rep = 1,
  id = NULL,
  silent = FALSE,
  stratify = FALSE,
  mc.cores = NULL,
  var.type = "IC",
  second.order = TRUE,
  response_model = deprecated,
  cate_model = deprecated,
  propensity_model = deprecated,
  propensity.model = deprecated,
  treatment = deprecated,
  ...
)
```

Arguments

`response.model`	formula or learner object (formula => learner_glm)
`treatment.model`	formula or learner object (formula => learner_glm)
`cate.model`	formula specifying regression design for conditional average treatment effects
`calibration.model`	linear calibration model. Specify covariates in addition to predicted potential outcomes to include in the calibration.
`missing.model`	formula or learner object (formula => learner_glm(family = binomial); default `NULL`. Model for the missingness mechanism $P(R = 1 \mid X, A)$. Required when the outcome in `response.model` contains NAs. If the formula LHS is omitted, the observation indicator is used automatically. When `stratify = TRUE` the

	missing model is fit separately per treatment arm. When supplied, the AIPW score is inverse-probability-of-observation weighted and (if `second.order = TRUE`) an additional second-order term is added to the influence function.
`data`	data.frame
`contrast`	treatment contrast (default 1 vs 0)
`nfolds`	number of folds (positive integer), or a pre-specified list of fold indices where each element is an integer vector of observation indices forming a partition of `1:nrow(data)`.
`rep`	number of replications of cross-fitting procedure by averaging estimates and influence functions from each replication
`id`	(integer or character) optional subject id vector of length `nrow(data)`. The `id` can also be specified as part of the `cate.model` argument with a formula syntax: `~ 1 + cluster(id)`.
`silent`	suppress all messages and progressbars
`stratify`	if TRUE the response.model and missing.model (if provided) will be stratified by treatment
`mc.cores`	(optional) number of cores. parallel::mcmapply used instead of future
`var.type`	when equal to "IC" the asymptotic variance is derived from the influence function. Otherwise, based on expressions in Bannick et al. (2025) valid under different covariate-adaptive randomization schemes (only available for ATE and when `calibration.model` is also specified)
`second.order`	add seconder order term to IF to handle misspecification of outcome models
`response_model`	Deprecated. Use response.model instead.
`cate_model`	Deprecated. Use cate.model instead.
`propensity_model`	Deprecated. Use treatment.model instead.
`propensity.model`	Deprecated. Use treatment.model instead.
`treatment`	Deprecated. Use cate.model instead.
`...`	additional arguments to future.apply::future_mapply

Details

We have observed data (Y, A, W) where Y is the response variable, A the binary treatment, and W covariates. We further let V be a subset of the covariates. Define the conditional potential mean outcome

$$\psi_a(P)(V) = E_P[E_P(Y \mid A = a, W)|V]$$

and let $m(V;\beta)$ denote a parametric working model, then the target parameter is the mean-squared error

$$\beta(P) = \text{argmin}_\beta \, E_P[\{\Psi_1(P)(V) - \Psi_0(P)(V)\} - m(V;\beta)]^2$$

Missing data is handled under a Missing At Random assumption (MAR). Let R denote the indicator for data not being missing, $R \perp Y|W, A$. The nuisance models are $Q(w, a) = E(Y|W = w, A =$

$a)$, $g_a(w) = P(A = a|W = w)$, and $\rho(w, a) = P(R = 1|W = w, A = a)$. For the expected potential outcome $E[Y(a)]$, the AIPW estimator then takes the form

$$\frac{1}{n}\sum_{i=1}^{n} R_i I(A_i = a)/\{g_a(W_i)\rho(W_i, a)\}(Y_i - Q(W_i, a)) + Q(W_i, a)$$

.

Value

cate.targeted object

Author(s)

Klaus Kähler Holst, Andreas Nordland

References

Mark J. van der Laan (2006) Statistical Inference for Variable Importance, The International Journal of Biostatistics.

Bannick, Shao & Liu et al. (2025) A General Form of Covariate Adjustment in Clinical Trials under Covariate-Adaptive Randomization, Biometrika.

Examples

```
sim1 <- function(n=1000, ...) {
  w1 <- rnorm(n)
  w2 <- rnorm(n)
  a <- rbinom(n, 1, plogis(-1 + w1))
  y <- cos(w1) + w2*a + 0.2*w2^2 + a + rnorm(n)
  data.frame(y, a, w1, w2)
}

d <- sim1(5000)
## ATE
cate(cate.model=~1,
     response.model=y~a*(w1+w2),
     treatment.model=a~w1+w2,
     data=d)
## CATE
cate(cate.model=~1+w2,
     response.model=y~a*(w1+w2),
     treatment.model=a~w1+w2,
     data=d)

## Not run:
# superlearner
mod1 <- list(
   glm = learner_glm(y~w1+w2),
   gam = learner_gam(y~s(w1) + s(w2))
)
s1 <- learner_sl(mod1, nfolds=5)
```

```
cate(cate.model=~1,
     response.model=s1,
     treatment.model=learner_glm(a~w1+w2, family=binomial),
     data=d,
     stratify=TRUE)

## End(Not run)

## Missing data
sim_missing_cate <- function(n = 5000, seed = 1) {
  set.seed(seed)
  w1 <- rnorm(n)
  w2 <- rnorm(n)
  a  <- rbinom(n, 1, 0.5) # randomized trial
  y_full <- 1 + a + w1 + 0.5 * w2 + rnorm(n)
  pR <- plogis(0.5 - 1 * w2 * a + 0.5 * a)
  R  <- rbinom(n, 1, pR)
  y  <- ifelse(R == 1, y_full, NA_real_)
  data.frame(y0 = y_full, y = y, a = a, w1 = w1, w2 = w2)
}
d <- sim_missing_cate()

# ignoring missing data (complete-case analysis)
cate(cate.model = ~1,
     response.model = y ~ a * w2, # wrong outcome model
     treatment.model = a ~ 1,
     data = na.omit(d), nfolds = 1L)
# MAR analysis
fit <- cate(cate.model = ~1,
            response.model = y ~ a * w2,
            treatment.model = a ~ 1,
            missing.model  = ~ a * (w1 + w2),
            data = d, nfolds = 1L)
fit
```

`cross_validated-class` *cross_validated class object*

Description

The functions `cv` returns an object of the type `cross_validated`.

An object of class '`cross_validated`' is a list with at least the following components:

cv An array with the model score(s) evaluated for each fold, repetition, and model estimates (see `estimate.default`)

names Names (character vector) of the models

rep number of repetitions of the CV

folds Number of folds of the CV

Value

objects of the S3 class 'cross_validated'

S3 generics

The following S3 generic functions are available for an object of class cross_validated:

coef Extract average model scores from the cross-validation procedure.

print Basic print method.

summary Summary of the cross-validation procedure.'

See Also

cv

Examples

```
# See example(cv) for examples
```

---

cumhaz *Predict the cumulative hazard/survival function for a survival model*

Description

Predict the cumulative hazard/survival function for a survival model

Usage

```
cumhaz(
  object,
  newdata,
  times = NULL,
  individual.time = FALSE,
  extend = FALSE,
  ...
)
```

Arguments

object	Survival model object: phreg, coxph, rfsrc, ranger, survSuperLearner
newdata	data.frame
times	numeric vector: Time points at which the survival model is evaluated. If NULL, the time points associated with the survival model is used.
individual.time	logical: If TRUE the survival object is evaluated at different time points for each row in newdata. The number of rows in newdata and the length of times must be the same.

extend	if TRUE, prints information for all specified 'times', even if there are no subjects left at the end of the specified 'times' (see survival::summary.survfit).
...	Additional arguments.

Value

List with elements:

- time: numeric vector
- chf: cumulative hazard function. If individual.time = FALSE, matrix with dimension (nrow(newdata), length(times)). If individual.time = TRUE, vector of length length(times).
- surv: survival function, exp(-chf).
- dchf: t(diff(rbind(0, t(chf))))

Author(s)

Klaus K. Holst, Andreas Nordland

`cv.default` *Cross-validation*

Description

Generic cross-validation function

Usage

```
## Default S3 method:
cv(
  object,
  data,
  response = NULL,
  nfolds = 5,
  rep = 1,
  weights = NULL,
  model.score = scoring,
  seed = NULL,
  shared = NULL,
  args.pred = NULL,
  args.future = list(),
  mc.cores,
  silent = FALSE,
  ...
)
```

Arguments

object	List of learner objects
data	data.frame or matrix
response	Response variable (vector or name of column in data).
nfolds	Number of folds (nfolds=0 simple test/train split into two folds 1:([n]/2), ([n]+1/2):n with last part used for testing)
rep	Number of repetitions (default 1)
weights	Optional frequency weights
model.score	Model scoring metric (default: MSE / Brier score). Must be a function with arguments response and prediction, and may optionally include weights, object and newdata arguments
seed	Random seed (argument parsed to future_Apply::future_lapply)
shared	Function applied to each fold with results send to each model
args.pred	Optional arguments to prediction function (see details below)
args.future	Arguments to future.apply::future_mapply
mc.cores	Optional number of cores. parallel::mcmapply used instead of future
silent	suppress all messages and progressbars
...	Additional arguments parsed to elements in object

Details

object should be list of objects of class learner. Alternatively, each element of models should be a list with a fitting function and a prediction function.

The response argument can optionally be a named list where the name is then used as the name of the response argument in models. Similarly, if data is a named list with a single data.frame/matrix then this name will be used as the name of the data/design matrix argument in models.

Value

An object of class 'cross_validated' is returned. See cross_validated-class for more details about this class and its generic functions.

Author(s)

Klaus K. Holst

See Also

cv.learner_sl

Examples

```
m <- list(learner_glm(Sepal.Length~1),
          learner_glm(Sepal.Length~Species),
          learner_glm(Sepal.Length~Species + Petal.Length))
x <- cv(m, rep=10, data=iris)
x
```

`cv.learner_sl` *Cross-validation for learner_sl*

Description

Cross-validation estimation of the generalization error of the super learner and each of the separate models in the ensemble. Both the chosen model scoring metrics as well as the model weights of the stacked ensemble.

Usage

```
## S3 method for class 'learner_sl'
cv(object, data, nfolds = 5, rep = 1, model.score = scoring, ...)
```

Arguments

`object`	(learner_sl) Instantiated learner_sl object.
`data`	data.frame or matrix
`nfolds`	Number of folds (nfolds=0 simple test/train split into two folds 1:([n]/2), ([n]+1/2):n with last part used for testing)
`rep`	Number of repetitions (default 1)
`model.score`	Model scoring metric (default: MSE / Brier score). Must be a function with arguments response and prediction, and may optionally include weights, object and newdata arguments
`...`	Additional arguments parsed to elements in `object`

Examples

```
sim1 <- function(n = 5e2) {
   x1 <- rnorm(n, sd = 2)
   x2 <- rnorm(n)
   y <- x1 + cos(x1) + rnorm(n, sd = 0.5**.5)
   data.frame(y, x1, x2)
}
sl <- learner_sl(list(
                   "mean" = learner_glm(y ~ 1),
                   "glm" = learner_glm(y ~ x1),
                   "glm2" = learner_glm(y ~ x1 + x2)
                  ))
cv(sl, data = sim1(), rep = 2)
```

deprecate_arg_warn *Cast warning for deprecated function argument names*

Description

Cast warning for deprecated function argument names

Usage

```
deprecate_arg_warn(old, new, fun, vers)
```

Arguments

old	deprecated argument name
new	argument that should be used instead
fun	function name where arguments are deprecated
vers	version when argument is deprecated

design *Extract design matrix*

Description

Extract design matrix from data.frame and formula

Usage

```
design(
  formula,
  data,
  ...,
  intercept = FALSE,
  response = TRUE,
  rm.envir = FALSE,
  specials = NULL,
  specials.call = NULL,
  levels = NULL,
  design.matrix = TRUE,
  na.action = na.omit
)
```

Arguments

formula	formula
data	data.frame
...	additional arguments (e.g, specials such weights, offsets, ...)
intercept	(logical) If FALSE an intercept is not included in the design matrix
response	(logical) if FALSE the response variable is dropped
rm.envir	(logical) Remove environment from terms attribute of returned object
specials	character vector specifying functions in the formula that should be marked as special in the terms object
specials.call	(call) specials optionally defined as a call-type
levels	a named list of character vectors giving the full set of levels to be assumed for each factor
design.matrix	(logical) if FALSE then only response and specials are returned. Otherwise, the design.matrix x is als part of the returned object.
na.action	(function) method to handle missing data (default: na.omit)

Value

An object of class 'design'

Author(s)

Klaus Kähler Holst

---

estimate_truncatedscore

Estimation of mean clinical outcome truncated by event process

---

Description

Let Y denote the clinical outcome, A the binary treatment variable, X baseline covariates, T the failure time, and $epsilon = 1, 2$ the cause of failure. The following are our two target parameters

$$E(Y|T > t, A = 1) - E(Y|T > t, A = 0)$$

$$P(T < t, \epsilon = 1|A = 1) - P(T < t, \epsilon = 1|A = 0)$$

Usage

```
estimate_truncatedscore(
  data,
  mod.y,
  mod.r,
  mod.a,
  mod.event,
  time,
  cause = NULL,
  cens.code = 0,
  naive = FALSE,
  control = list(),
  ...
)
```

Arguments

data	(data.frame)
mod.y	(formula or learner) Model for clinical outcome given T>time. Using a formula specifies a glm with an identity link (see example).
mod.r	(formula or learner) Model for missing data mechanism for clinical outcome at T=time. Using a formula specifies a glm with a log link.
mod.a	(formula or learner) Treatment model (in RCT should just be 'a ~ 1'). Using a formula specifies a glm with a log link.
mod.event	(formula) Model for time-to-event process ('Event(time,status) ~ x').
time	(numeric) Landmark time.
cause	(integer) Primary event (in the 'status' variable of the 'Event' statement).
cens.code	(integer) Censoring code.
naive	(logical) If TRUE, the unadjusted estimates ignoring baseline covariates is returned as the attribute 'naive'.
control	(list) optimization routine parameters.
...	Additional arguments passed to mets::binregATE.

Value

lava::estimate.default object

Author(s)

Klaus Kähler Holst

Examples

```
data(truncatedscore)
mod1 <- learner_glm(y ~ a * (x1 + x2))
mod2 <- learner_glm(r ~ a * (x1 + x2), family = binomial)
```

```
a <- estimate_truncatedscore(
  data = truncatedscore,
  mod.y = mod1,
  mod.r = mod2,
  mod.a = a ~ 1,
  mod.event = mets::Event(time, status) ~ x1+x2,
  time = 2
)
s <- summary(a, noninf.t = -0.1)
print(s)
parameter(s)

# the above is equivalent to
# a <- estimate_truncatedscore(
#   data = truncatedscore,
#   mod.y = y ~ a * (x1 + x2),
#   mod.r = r ~ a * (x1 + x2),
#   mod.a = a ~ 1,
#   mod.event = mets::Event(time, status) ~ x1+x2,
#   time = 2
# )
```

`expand.list` *Create a list from all combination of input variables*

Description

Similar to `expand.grid` function, this function creates all combinations of the input arguments but returns the result as a list.

Usage

```
expand.list(..., INPUT = NULL, envir = NULL)
```

Arguments

`...`	input variables
`INPUT`	optional list of variables
`envir`	environment environment to evalute formulas in

Value

list

Author(s)

Klaus Kähler Holst

Examples

```
expand.list(x = 2:4, z = c("a", "b"))
```

---

fit_survival_models *Fit survival nuisance models*

Description

Fit survival nuisance models

Usage

```
fit_survival_models(
  data,
  response,
  censoring,
  response_call = "phreg",
  response_args = list(),
  censoring_call = "phreg",
  censoring_args = list()
)
```

Arguments

data	data.frame
response	Response formula (e.g., Surv(time, event) ~ A + W)
censoring	Censoring formula (e.g., Surv(time, event == 0) ~ A + W))
response_call	Model call for the response model (e.g. "mets::phreg")
response_args	Additional arguments passed to the response model
censoring_call	Similar to response_callb
censoring_args	Similar to response_args

Value

List with elements T_model and C_model

Author(s)

Andreas Nordland, Klaus K. Holst

int_surv *Integral approximation of a time dependent function.*

Description

Computes an approximation of $\int_s tart^s topS(t)dt$, where $S(t)$ is a survival function, for a selection of start and stop time points.

Usage

```
int_surv(times, surv, start = 0, stop = max(times), extend = FALSE)
```

Arguments

times	Numeric vector, sorted time points.
surv	Numeric vector, values of a survival function evaluated at time points given by times.
start	Numeric vector, start of the integral.
stop	Numeric vector, end of the integral.
extend	should the integral be extended beyond the last observed time point

Value

Numeric vector, value of the integral.

Author(s)

Andreas Nordland

learner *R6 class for prediction models*

Description

Interface for statistical and machine learning models to be used for nuisance model estimation in targeted learning.

The following list provides an overview of constructors for many commonly used models.

Regression and classification: learner_glm, learner_gam, learner_grf, learner_hal, learner_glmnet_cv, learner_svm, learner_xgboost, learner_mars
Regression: learner_isoreg
Classification: learner_naivebayes
Ensemble (super learner): learner_sl

The following constructors for commonly used filters are available: predict_filter_bound, predict_filter_bound_dynamic

Public fields

`info` Optional information/name of the model

Active bindings

`clear` Remove fitted model from the learner object

`fit` Return estimated model object.

`formula` Return model formula. Use learner$update() to update the formula.

`predict.filter` Return instantiated prediction filter function

`predict.filter.generator` Return prediction filter generator function

Methods

Public methods:

- learner$new()
- learner$estimate()
- learner$predict()
- learner$update()
- learner$print()
- learner$summary()
- learner$response()
- learner$design()
- learner$opt()
- learner$clone()

`learner$new()`: Create a new prediction model object

Usage:

```
learner$new(
  formula = NULL,
  estimate,
  predict = stats::predict,
  predict.args = NULL,
  estimate.args = NULL,
  info = NULL,
  specials = c(),
  formula.keep.specials = FALSE,
  predict.filter = function(data) function(pred, newdata) pred,
  intercept = FALSE
)
```

Arguments:

`formula` formula specifying outcome and design matrix

`estimate` function for fitting the model. This must be a function with response, 'y', and design matrix, 'x'. Alternatively, a function with a formula and data argument. See the examples section.

`predict` prediction function (must be a function of model object, 'object', and new design matrix, 'newdata')

`predict.args` optional arguments to prediction function

`estimate.args` optional arguments to estimate function

`info` optional description of the model

`specials` optional specials terms (weights, offset, id, subset, ...) passed on to design

`formula.keep.specials` if TRUE then special terms defined by `specials` will be removed from the formula before it is being passed to the estimate print.function()

`predict.filter` function to post-process predictions. Useful to bound predictions or handle NAs. The argument is experimental and its behavior may change in the future.

`intercept` (logical) include intercept in design matrix

`learner$estimate()`: Estimation method

Usage:

```
learner$estimate(data, ..., store = TRUE)
```

Arguments:

`data` data.frame

`...` Additional arguments to estimation and prediction filter generator function

`store` Logical determining if estimated model should be stored inside the class.

`learner$predict()`: Prediction method

Usage:

```
learner$predict(newdata, ..., object = NULL)
```

Arguments:

`newdata` data.frame

`...` Additional arguments to prediction method and prediction filter function

`object` Optional model fit object

`learner$update()`: Update formula

Usage:

```
learner$update(formula)
```

Arguments:

`formula` formula or character which defines the new response

`learner$print()`: Print method

Usage:

```
learner$print()
```

`learner$summary()`: Summary method to provide more extensive information than learner$print().

Usage:

```
learner$summary()
```

Returns: summarized_learner object, which is a list with the following elements:

info description of the learner

formula formula specifying outcome and design matrix

estimate function for fitting the model

estimate.args arguments to estimate function

predict function for making predictions from fitted model

predict.args arguments to predict function

specials provided special terms

intercept include intercept in design matrix

Examples:

```
lr <- learner_glm(y ~ x, family = "nb")
lr$summary()

lr_sum <- lr$summary() # store returned summary in new object
names(lr_sum)
print(lr_sum)
```

`learner$response()`: Extract response from data

Usage:

```
learner$response(data, eval = TRUE, ...)
```

Arguments:

`data` data.frame

`eval` when FALSE return the untransformed outcome (i.e., return 'a' if formula defined as I(a==1) ~ ...)

`...` additional arguments to design

`learner$design()`: Generate design object (design matrix and response) from data

Usage:

```
learner$design(data, ...)
```

Arguments:

`data` data.frame

`...` additional arguments to design

`learner$opt()`: Get options

Usage:

```
learner$opt(arg)
```

Arguments:

`arg` name of option to get value of

`learner$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Klaus Kähler Holst, Benedikt Sommer

Examples

```
data(iris)
rf <- function(formula, ...) {
  learner$new(formula,
    info = "grf::probability_forest",
    estimate = function(x, y, ...) {
      grf::probability_forest(X = x, Y = y, ...)
    },
    predict = function(object, newdata) {
      predict(object, newdata)$predictions
    },
    estimate.args = list(...)
  )
}

args <- expand.list(
  num.trees = c(100, 200), mtry = 1:3,
  formula = c(Species ~ ., Species ~ Sepal.Length + Sepal.Width)
)
models <- lapply(args, function(par) do.call(rf, par))

x <- models[[1]]$clone()
x$estimate(iris)
predict(x, newdata = head(iris))


# Reduce Ex. timing
a <- targeted::cv(models, data = iris)
cbind(coef(a), attr(args, "table"))


# defining learner via function with arguments y (response)
# and x (design matrix)
f1 <- learner$new(
  estimate = function(y, x) lm.fit(x = x, y = y),
  predict = function(object, newdata) newdata %*% object$coefficients
)
# defining the learner via arguments formula and data
f2 <- learner$new(
  estimate = function(formula, data, ...) glm(formula, data, ...)
)
# generic learner defined from function (predict method derived per default
# from stats::predict
f3 <- learner$new(
  estimate = function(dt, ...) {
    lm(y ~ x, data = dt)
  }
)
```

```
## ------------------------------------------------
## Method `learner$summary()`
## ------------------------------------------------

lr <- learner_glm(y ~ x, family = "nb")
lr$summary()

lr_sum <- lr$summary() # store returned summary in new object
names(lr_sum)
print(lr_sum)
```

`learner_expand_grid` *Construct learners from a grid of parameters*

Description

Construct learners from a grid of parameters

Usage

```
learner_expand_grid(fun, args, names = TRUE, params = FALSE)
```

Arguments

`fun`	(function) A function that returns a learner.
`args`	(list) Parameters that generate a grid of parameters with expand.list, where the set of parameters are then passed on to `fun`.
`names`	(logical or character) If FALSE, then return a list without names. If TRUE, a named list is returned (see details).
`params`	(logical) If FALSE, then no information about the parameters defined by `args` are added to the names of the returned list.

Value

list

Examples

```
lrs <- learner_expand_grid(
  learner_xgboost,
  list(formula = Sepal.Length ~ ., eta = c(0.2, 0.5, 0.3))
)
lrs # use info of constructed learner as names

lrs <- learner_expand_grid(
  learner_xgboost,
  list(formula = Sepal.Length ~ ., eta = c(0.2, 0.5, 0.3)),
```

```
  names = "xgboost"
)
names(lrs) # use xgboost instead of info field for names

lrs <- learner_expand_grid(
  learner_xgboost,
  list(formula = Sepal.Length ~ ., eta = c(0.2, 0.5, 0.3)),
  names = "xgboost",
  params = TRUE
)
names(lrs) # also add parameters to names

lrs <- learner_expand_grid(
  learner_xgboost,
  list(formula = Sepal.Length ~ ., eta = c(0.2, 0.5, 0.3)),
  names = FALSE
)
names(lrs) # unnamed list since names = FALSE
```

`learner_gam` *Construct a learner*

Description

Constructs learner class object for fitting generalized additive models with mgcv::gam.

Usage

```
learner_gam(
  formula,
  info = "mgcv::gam",
  family = gaussian(),
  select = FALSE,
  gamma = 1,
  learner.args = NULL,
  ...
)
```

Arguments

`formula`	(formula) Formula specifying response and design matrix.
`info`	(character) Optional information to describe the instantiated learner object.
`family`	This is a family object specifying the distribution and link to use in fitting etc (see `glm` and `family`). See `family.mgcv` for a full list of what is available, which goes well beyond exponential family. Note that `quasi` families actually result in the use of extended quasi-likelihood if `method` is set to a RE/ML method (McCullagh and Nelder, 1989, 9.6).

select	If this is TRUE then gam can add an extra penalty to each term so that it can be penalized to zero. This means that the smoothing parameter estimation that is part of fitting can completely remove terms from the model. If the corresponding smoothing parameter is estimated as zero then the extra penalty has no effect. Use gamma to increase level of penalization.
gamma	Increase this beyond 1 to produce smoother models. gamma multiplies the effective degrees of freedom in the GCV or UBRE/AIC. n/gamma can be viewed as an effective sample size in the GCV score, and this also enables it to be used with REML/ML. Ignored with P-RE/ML or the efs optimizer.
learner.args	(list) Additional arguments to learner$new().
...	Additional arguments to mgcv::gam.

Value

learner object.

Examples

```
n <- 5e2
x1 <- rnorm(n, sd = 2)
x2 <- rnorm(n)
y <- x1 + cos(x1) + rnorm(n, sd = 0.5**.5)
d0 <- data.frame(y, x1, x2)

lr <- learner_gam(y ~ s(x1) + x2)
lr$estimate(d0)
if (interactive()) {
  plot(lr$fit)
}
```

---

learner_glm *Construct a learner*

Description

Constructs a learner class object for fitting generalized linear models with stats::glm and MASS::glm.nb. Negative binomial regression is supported with family = "nb" (or alternatively family = "negbin").

Usage

```
learner_glm(
  formula,
  info = "glm",
  family = gaussian(),
  learner.args = NULL,
  ...
)
```

Arguments

formula	(formula) Formula specifying response and design matrix.
info	(character) Optional information to describe the instantiated learner object.
family	a description of the error distribution and link function to be used in the model. For glm this can be a character string naming a family function, a family function or the result of a call to a family function. For glm.fit only the third option is supported. (See family for details of family functions.)
learner.args	(list) Additional arguments to learner$new().
...	Additional arguments to stats::glm or MASS::glm.nb.

Value

learner object.

Examples

```
n <- 5e2
x <- rnorm(n)
w <- 50 + rexp(n, rate = 1 / 5)
y <- rpois(n, exp(2 + 0.5 * x + log(w)) * rgamma(n, 1 / 2, 1 / 2))
d0 <- data.frame(y, x, w)

lr <- learner_glm(y ~ x) # linear Gaussian model
lr$estimate(d0)
coef(lr$fit)

# negative binomial regression model with offset (using MASS::glm.nb)
lr <- learner_glm(y ~ x + offset(log(w)), family = "nb")
lr$estimate(d0)
coef(lr$fit)
lr$predict(data.frame(x = 1, w = c(1, 5))) # response scale
lr$predict(data.frame(x = 1, w = c(1, 5)), type = "link") # link scale
```

---

learner_glmnet_cv *Construct a learner*

Description

Constructs a learner class object for fitting entire lasso or elastic-net regularization paths for various linear and non-linear regression models with glmnet::cv.glmnet. Predictions are returned for the value of lambda that gives minimum cvm. That is, glmnet::predict.cv.glmnet is called with s = "lambda.min".

Usage

```
learner_glmnet_cv(
  formula,
  info = "glmnet::cv.glmnet",
  family = gaussian(),
  lambda = NULL,
  alpha = 1,
  nfolds = 10,
  learner.args = NULL,
  ...
)
```

Arguments

`formula`	(formula) Formula specifying response and design matrix.
`info`	(character) Optional information to describe the instantiated learner object.
`family`	Either a character string representing one of the built-in families, or else a `glm()` family object. For more information, see Details section below or the documentation for response type (above).
`lambda`	Optional user-supplied lambda sequence; default is `NULL`, and `glmnet` chooses its own sequence. Note that this is done for the full model (master sequence), and separately for each fold. The fits are then alligned using the master sequence (see the `allignment` argument for additional details). Adapting `lambda` for each fold leads to better convergence. When `lambda` is supplied, the same sequence is used everywhere, but in some GLMs can lead to convergence issues.
`alpha`	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $$(1-\alpha)/2||\beta||_2^2 + \alpha||\beta||_1.$$ `alpha=1` is the lasso penalty, and `alpha=0` the ridge penalty.
`nfolds`	number of folds - default is 10. Although `nfolds` can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is `nfolds=3`
`learner.args`	(list) Additional arguments to learner$new().
`...`	Other arguments that can be passed to glmnet, for example `alpha`, `nlambda`, etc. See `glmnet` for details.

Value

learner object.

Examples

```
# continuous outcome
n <- 5e2
x1 <- rnorm(n, sd = 2)
x2 <- rnorm(n)
```

```
lp <- x1 + x2*x1 + cos(x1)
y <- rnorm(n, lp, sd = 2)
d0 <- data.frame(y, x1, x2)

lr <- learner_glmnet_cv(y ~ x1 + x2)
lr$estimate(d0, nfolds = 3)
lr$predict(data.frame(x1 = c(0, 1), x2 = 1))

# count outcome with different exposure time
w <- 50 + rexp(n, rate = 1 / 5)
y <- rpois(n, exp(0.5 * x1 - 1 * x2 + log(w)) * rgamma(n, 1 / 2, 1 / 2))
d0 <- data.frame(y, x1, x2, w)

lr <- learner_glmnet_cv(y ~ x1 + x2 + offset(log(w)), family = "poisson")
lr$estimate(d0, nfolds = 3)
lr$predict(data.frame(x1 = 1, x2 = 1, w = c(1, 5)))
```

`learner_grf` *Construct a learner*

Description

Constructs a learner class object for fitting generalized random forest models with grf::regression_forest or grf::probability_forest. As shown in the examples, the constructed learner returns predicted class probabilities of class 2 in case of binary classification. A `n times p` matrix, with `n` being the number of observations and `p` the number of classes, is returned for multi-class classification.

Usage

```
learner_grf(
  formula,
  num.trees = 2000,
  min.node.size = 5,
  alpha = 0.05,
  sample.fraction = 0.5,
  num.threads = 1,
  model = "grf::regression_forest",
  info = model,
  learner.args = NULL,
  ...
)
```

Arguments

`formula`	(formula) Formula specifying response and design matrix.
`num.trees`	Number of trees grown in the forest. Note: Getting accurate confidence intervals generally requires more trees than getting accurate predictions. Default is 2000.

`min.node.size` A target for the minimum number of observations in each tree leaf. Note that nodes with size smaller than min.node.size can occur, as in the original randomForest package. Default is 5.

`alpha` A tuning parameter that controls the maximum imbalance of a split. Default is 0.05.

`sample.fraction` Fraction of the data used to build each tree. Note: If honesty = TRUE, these subsamples will further be cut by a factor of honesty.fraction. Default is 0.5.

`num.threads` Number of threads used in training. By default, the number of threads is set to the maximum hardware concurrency.

`model` (character) grf model to estimate. Usually regression_forest (grf::regression_forest) or probability_forest (grf::probability_forest).

`info` (character) Optional information to describe the instantiated learner object.

`learner.args` (list) Additional arguments to learner$new().

`...` Additional arguments to `model`

Value

learner object.

Examples

```
n <- 5e2
x1 <- rnorm(n, sd = 2)
x2 <- rnorm(n)
lp <- x2*x1 + cos(x1)
yb <- rbinom(n, 1, lava::expit(lp))
y <-  lp + rnorm(n, sd = 0.5**.5)
d <- data.frame(y, yb, x1, x2)

# regression
lr <- learner_grf(y ~ x1 + x2)
lr$estimate(d)
lr$predict(head(d))

# binary classification
lr <- learner_grf(as.factor(yb) ~ x1 + x2, model = "probability_forest")
lr$estimate(d)
lr$predict(head(d)) # predict class probabilities of class 2

# multi-class classification
lr <- learner_grf(Species ~ ., model = "probability_forest")
lr$estimate(iris)
lr$predict(head(iris))
```

`learner_hal` *Construct a learner*

Description

Constructs a learner class object for fitting a highly adaptive lasso model with hal9001::fit_hal.

Usage

```
learner_hal(
  formula,
  info = "hal9001::fit_hal",
  smoothness_orders = 0,
  reduce_basis = NULL,
  family = "gaussian",
  learner.args = NULL,
  ...
)
```

Arguments

`formula`	(formula) Formula specifying response and design matrix.
`info`	(character) Optional information to describe the instantiated learner object.
`smoothness_orders`	An `integer`, specifying the smoothness of the basis functions. See details for `smoothness_orders` for more information.
`reduce_basis`	Am optional `numeric` value bounded in the open unit interval indicating the minimum proportion of 1's in a basis function column needed for the basis function to be included in the procedure to fit the lasso. Any basis functions with a lower proportion of 1's than the cutoff will be removed. Defaults to 1 over the square root of the number of observations. Only applicable for models fit with zero-order splines, i.e. `smoothness_orders = 0`.
`family`	A `character` or a `family` object (supported by `glmnet`) specifying the error/link family for a generalized linear model. `character` options are limited to "gaussian" for fitting a standard penalized linear model, "binomial" for penalized logistic regression, "poisson" for penalized Poisson regression, "cox" for a penalized proportional hazards model, and "mgaussian" for multivariate penalized linear model. Note that passing in family objects leads to slower performance relative to passing in a character family (if supported). For example, one should set `family = "binomial"` instead of `family = binomial()` when calling `fit_hal`.
`learner.args`	(list) Additional arguments to learner$new().
`...`	Additional arguments to hal9001::fit_hal.

Value

learner object.

Examples

```
## Not run:
n <- 5e2
x1 <- rnorm(n, sd = 2)
x2 <- rnorm(n)
y <- x1 + cos(x1) + rnorm(n, sd = 0.5**.5)
d <- data.frame(y, x1, x2)
lr <- learner_hal(y ~ x1 + x2, smoothness_orders = 0.5, reduce_basis = 1)
lr$estimate(d)
lr$predict(data.frame(x1 = 0, x2 = c(-1, 1)))

## End(Not run)
```

---

learner_isoreg *Construct a learner*

Description

Constructs a learner class object for isotonic regression with isoregw.

Usage

```
learner_isoreg(formula, info = "targeted::isoregw", learner.args = NULL, ...)
```

Arguments

formula	(formula) Formula specifying response and design matrix.
info	(character) Optional information to describe the instantiated learner object.
learner.args	(list) Additional arguments to learner$new().
...	Additional arguments to isoregw.

Value

learner object.

Examples

```
x <- runif(5e3, -5, 5)
pr <- lava::expit(-1 + x)
y <- rbinom(length(pr), 1, pr)
d <- data.frame(y, x)

lr <- learner_isoreg(y ~ x)
lr$estimate(d)
```

```
pr_iso <- lr$predict(d)

if (interactive()) {
  plot(pr ~ x, cex=0.3)
  lines(sort(x), pr_iso[order(x)], col="red", type="s")
}
```

`learner_mars` *Construct a learner*

Description

Constructs a learner class object for fitting multivariate adaptive regression splines with earth::earth.

Usage

```
learner_mars(
  formula,
  info = "earth::earth",
  degree = 1,
  nprune = NULL,
  glm = NULL,
  learner.args = NULL,
  ...
)
```

Arguments

`formula`	(formula) Formula specifying response and design matrix.
`info`	(character) Optional information to describe the instantiated learner object.
`degree`	Maximum degree of interaction (Friedman's mi). Default is 1, meaning build an additive model (i.e., no interaction terms).
`nprune`	Maximum number of terms (including intercept) in the pruned model. Default is NULL, meaning all terms created by the forward pass (but typically not all terms will remain after pruning). Use this to enforce an upper bound on the model size (that is less than `nk`), or to reduce exhaustive search time with `pmethod="exhaustive"`.
	The following arguments are for cross validation.
`glm`	NULL (default) or a list of arguments to pass on to `glm`. See the documentation of `glm` for a description of these arguments See "*Generalized linear models*" in the vignette. Example: `earth(survived~., data=etitanic, degree=2, glm=list(family=binomial))`
	The following arguments are for the forward pass.
`learner.args`	(list) Additional arguments to learner$new().
`...`	Additional arguments to earth::earth.

Value

learner object.

Examples

```
# poisson regression
n <- 5e2
x <- rnorm(n)
w <- 50 + rexp(n, rate = 1 / 5)
y <- rpois(n, exp(2 + 0.5 * x + log(w)) * rgamma(n, 1 / 2, 1 / 2))
d0 <- data.frame(y, x, w)

lr <- learner_mars(y ~ x + offset(log(w)), degree = 2,
  glm = list(family = poisson())
)
lr$estimate(d0)
lr$predict(data.frame(x = 0, w = c(1, 2)))
```

---

learner_naivebayes *Construct a learner*

Description

Constructs a learner class object for fitting a naive bayes classifier with naivebayes. As shown in the examples, the constructed learner returns predicted class probabilities of class 2 in case of binary classification. A `n times p` matrix, with `n` being the number of observations and `p` the number of classes, is returned for multi-class classification.

Usage

```
learner_naivebayes(
  formula,
  info = "Naive Bayes",
  laplace.smooth = 0,
  kernel = FALSE,
  learner.args = NULL,
  ...
)
```

Arguments

`formula`	(formula) Formula specifying response and design matrix.
`info`	(character) Optional information to describe the instantiated learner object.
`laplace.smooth`	Laplace smoothing
`kernel`	If TRUE a kernel estimator is used for numeric predictors (otherwise a gaussian model is used)
`learner.args`	(list) Additional arguments to learner$new().
`...`	Additional arguments to naivebayes.

Value

learner object.

Examples

```
n <- 5e2
x1 <- rnorm(n, sd = 2)
x2 <- rnorm(n)
y <- rbinom(n, 1, lava::expit(x2*x1 + cos(x1)))
d <- data.frame(y, x1, x2)

# binary classification
lr <- learner_naivebayes(y ~ x1 + x2)
lr$estimate(d)
lr$predict(head(d))

# multi-class classification
lr <- learner_naivebayes(Species ~ .)
lr$estimate(iris)
lr$predict(head(iris))
```

---

learner_sl *Construct a learner*

Description

Constructs a learner class object for fitting a superlearner.

Usage

```
learner_sl(
  learners,
  info = NULL,
  nfolds = 5L,
  meta.learner = metalearner_nnls,
  model.score = mse,
  learner.args = NULL,
  ...
)
```

Arguments

learners	(list) List of learner objects (i.e. learner_glm)
info	(character) Optional information to describe the instantiated learner object.
nfolds	(integer) Number of folds to use in cross-validation to estimate the ensemble weights.

meta.learner	(function) Algorithm to learn the ensemble weights (default non-negative least squares). Must be a function of the response (nx1 vector), y, and the base learner predictions (nxp matrix), pred, with p being the number of learners. The function can optionally accept a model.score argument for scoring the base learners. See metalearner_nnls, metalearner_convexcomb and metalearner_discrete for the available meta learners.
model.score	(function) Method for scoring the predictions of each base learner. Expects two arguments; vector of response variable and prediction from a base learner (see targeted:::mse for additional details).
learner.args	(list) Additional arguments to learner$new().
...	Additional arguments to superlearner

Value

learner object.

See Also

cv.learner_sl

Examples

```
sim1 <- function(n = 5e2) {
   x1 <- rnorm(n, sd = 2)
   x2 <- rnorm(n)
   y <- x1 + cos(x1) + rnorm(n, sd = 0.5**.5)
   data.frame(y, x1, x2)
}
d <- sim1()

m <- list(
  "mean" = learner_glm(y ~ 1),
  "glm" = learner_glm(y ~ x1 + x2),
  "iso" = learner_isoreg(y ~ x1)
)

s <- learner_sl(m, nfolds = 10)
s$estimate(d)
pr <- s$predict(d)
if (interactive()) {
    plot(y ~ x1, data = d)
    points(d$x1, pr, col = 2, cex = 0.5)
    lines(cos(x1) + x1 ~ x1, data = d[order(d$x1), ],
          lwd = 4, col = lava::Col("darkblue", 0.3))
}
print(s)
# weights(s$fit)
# score(s$fit)

cvres <- cv(s, data = d, nfolds = 3, rep = 2)
cvres
```

```
# coef(cvres)
# score(cvres)
```

`learner_stratify` *Construct stratified learner*

Description

This function creates a stratified learner from an existing learner wrapper function such as learner_glm or learner_xgboost. The stratification variable can be specified either using the `stratify` argument (which can be given as a string "a" or a formula , for example ~ I(a==0)), or it can be defined as a special term directly in the formula, y ~ ... + stratify(a). The formula will subsequently be passed to the `learner_` wrapper without the stratify special term.

Usage

```
learner_stratify(
  formula,
  learner,
  stratify = NULL,
  info = NULL,
  learner.args = list(),
  ...
)
```

Arguments

`formula`	formula specifying outcome and design matrix
`learner`	(learner) learner object
`stratify`	(character,formula) variables to stratify by
`info`	optional description of the model
`learner.args`	(list) optional arguments to the learner constructor
`...`	additional arguments passed to the learner constructor

Value

learner object

Examples

```
simdata <- function(n=1000) {
  a <- rbinom(n, 1, 0.5)
  x <- rnorm(n)
  y <- rbinom(n, 1, plogis(-1 + a + a * x))
  data.frame(y, a, x)
}
d <- simdata()
```

```
lr <- learner_stratify(
  y ~ x + stratify(a),
  learner_glm,
  family=binomial()
)
lr$estimate(d)
lr$predict(head(d))
```

`learner_surv_cox` *Construct a learner*

Description

Constructs a learner class object for fitting Cox proportional hazards models.

Usage

```
learner_surv_cox(formula, info = "mets::phreg", learner.args = NULL, ...)
```

Arguments

`formula`	(formula) Formula specifying response and design matrix.
`info`	(character) Optional information to describe the instantiated learner object.
`learner.args`	(list) Additional arguments to learner$new().
`...`	Additional arguments passed to lower-level functions (e.g., optimization controls).

Value

learner object.

Author(s)

Klaus Kähler Holst

Examples

```
data(sTRACE, package="mets")
mod <- learner_surv_cox(Surv(time, status>0) ~ sex + strata(age))
mod$estimate(sTRACE)
mod$predict(head(sTRACE), times=5) # P(T>t|X)
```

`learner_surv_rf` *Construct a learner*

Description

Constructs a learner class object for random survival forests

Usage

```
learner_surv_rf(
  formula,
  info = "survival forest (ranger)",
  num.threads = 1L,
  learner.args = NULL,
  ...
)
```

Arguments

`formula`	(formula) Formula specifying response and design matrix.
`info`	(character) Optional information to describe the instantiated learner object.
`num.threads`	Number of threads. Use 0 for all available cores. Default is 2 if not set by options/environment variables (see below).
`learner.args`	(list) Additional arguments to learner$new().
`...`	Further arguments passed to or from other methods (currently ignored).

Value

learner object.

Author(s)

Klaus Kähler Holst

Examples

```
data(sTRACE, package="mets")
mod <- learner_surv_rf(Surv(time, status>0) ~ sex + age)
mod$estimate(sTRACE)
mod$predict(head(sTRACE), times=5) # P(T>t|X)
```

learner_svm *Construct a learner*

Description

Constructs a learner class object for fitting support vector machines with e1071::svm. As shown in the examples, the constructed learner returns predicted class probabilities of class 2 in case of binary classification. A `n times p` matrix, with `n` being the number of observations and `p` the number of classes, is returned for multi-class classification.

Usage

```
learner_svm(
  formula,
  info = "e1071::svm",
  cost = 1,
  epsilon = 0.1,
  kernel = "radial",
  learner.args = NULL,
  ...
)
```

Arguments

`formula`	(formula) Formula specifying response and design matrix.
`info`	(character) Optional information to describe the instantiated learner object.
`cost`	cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation.
`epsilon`	epsilon in the insensitive-loss function (default: 0.1)
`kernel`	the kernel used in training and predicting. You might consider changing some of the following parameters, depending on the kernel type. **linear:** $u'v$ **polynomial:** $(\gamma u'v + coef0)^{degree}$ **radial basis:** $e^(- \gamma \lvert u - v \rvert^2)$ **sigmoid:** $tanh(\gamma u'v + coef0)$
`learner.args`	(list) Additional arguments to learner$new().
`...`	Additional arguments to e1071::svm.

Value

learner object.

Examples

```
n <- 5e2
x1 <- rnorm(n, sd = 2)
x2 <- rnorm(n)
lp <- x2*x1 + cos(x1)
yb <- rbinom(n, 1, lava::expit(lp))
y <-  lp + rnorm(n, sd = 0.5**.5)
d <- data.frame(y, yb, x1, x2)

# regression
lr <- learner_svm(y ~ x1 + x2)
lr$estimate(d)
lr$predict(head(d))

# binary classification
lr <- learner_svm(as.factor(yb) ~ x1 + x2)
# alternative to transforming response variable to factor
# lr <- learner_svm(yb ~ x1 + x2, type = "C-classification")
lr$estimate(d)
lr$predict(head(d)) # predict class probabilities of class 2
lr$predict(head(d), probability = FALSE) # predict labels

# multi-class classification
lr <- learner_svm(Species ~ .)
lr$estimate(iris)
lr$predict(head(iris))
```

---

learner_xgboost *Construct a learner*

Description

Constructs a learner class object for xgboost::xgboost.

Usage

```
learner_xgboost(
  formula,
  max_depth = 2L,
  learning_rate = 1,
  nrounds = 2L,
  subsample = 1,
  reg_lambda = 1,
  objective = "reg:squarederror",
  info = paste("xgboost", objective),
  learner.args = NULL,
  ...
)
```

Arguments

formula	(formula) Formula specifying response and design matrix.
max_depth	(integer) Maximum depth of a tree.
learning_rate	(numeric) Learning rate.
nrounds	Number of boosting iterations / rounds. Note that the number of default boosting rounds here is not automatically tuned, and different problems will have vastly different optimal numbers of boosting rounds.
subsample	(numeric) Subsample ratio of the training instance.
reg_lambda	(numeric) L2 regularization term on weights.
objective	(character) Specify the learning task and the corresponding learning objective. See xgboost::xgboost for all available options.
info	(character) Optional information to describe the instantiated learner object.
learner.args	(list) Additional arguments to learner$new().
...	Additional arguments to xgboost::xgboost.

Value

learner object.

Examples

```
n  <- 1e3
x1 <- rnorm(n, sd = 2)
x2 <- rnorm(n)
lp <- x2*x1 + cos(x1)
yb <- rbinom(n, 1, lava::expit(lp))
y <-  lp + rnorm(n, sd = 0.5**.5)
d0 <- data.frame(y, yb, x1, x2)

# regression
lr <- learner_xgboost(y ~ x1 + x2, nrounds = 5)
lr$estimate(d0)
lr$predict(head(d0))

# binary classification
lr <- learner_xgboost(yb ~ x1 + x2, nrounds = 5,
 objective = "binary:logistic"
)
lr$estimate(d0)
lr$predict(head(d0))

# multi-class classification
d0 <- iris
d0$y <- as.numeric(d0$Species)- 1

lr <- learner_xgboost(y ~ ., objective = "multi:softprob", num_class = 3)
lr$estimate(d0)
lr$predict(head(d0))
```

metalearner_convexcomb *Convex combination meta learner*

Description

Estimates the ensemble weights of a superlearner by minimizing the cross-validated MSE as a convex combination of the candidate predictions, i.e. by least squares regression of the response on the candidate predictions subject to the constraint that the weights are non-negative and sum to one.

Usage

```
metalearner_convexcomb(y, pred, ...)
```

Arguments

y	(numeric) Response vector.
pred	(matrix) Matrix of cross-validated predictions with one column per candidate learner.
...	Additional arguments (currently ignored).

Value

(numeric) Vector of ensemble weights, one element per column of pred.

See Also

superlearner learner_sl

metalearner_discrete *Discrete meta learner*

Description

Implements the discrete super learner: the candidate learner with the lowest risk (computed via the model.score argument of superlearner) is given weight one and all other learners weight zero.

Usage

```
metalearner_discrete(y, pred, model.score, ...)
```

Arguments

y	(numeric) Response vector.
pred	(matrix) Matrix of cross-validated predictions with one column per candidate learner.
model.score	(function) Method for scoring the predictions of each base learner.
...	Additional arguments (currently ignored).

Value

(numeric) Vector of ensemble weights, one element per column of pred.

See Also

superlearner learner_sl

---

metalearner_nnls *Non-negative least squares meta learner*

Description

Estimates the ensemble weights of a superlearner by minimizing the cross-validated MSE via non-negative least squares regression. The estimated weights are non-negative and normalized to sum to one.

Usage

```
metalearner_nnls(y, pred, method = "quadprog", ...)
```

Arguments

y	(numeric) Response vector.
pred	(matrix) Matrix of cross-validated predictions with one column per candidate learner.
method	(character) Quadratic-programming solver used to compute the non-negative least squares weights. Either "quadprog" (default, using quadprog::solve.QP) or "nnls" (using nnls::nnls).
...	Additional arguments (currently ignored).

Details

targeted:::metalearner_nnls2 is an internal wrapper for using the "nnls" package instead of "quadprog".

Value

(numeric) Vector of ensemble weights, one element per column of pred.

See Also

superlearner learner_sl

`moi` *Average Treatment Effect Estimation with Missing Outcome Imputation*

Description

Estimates the Average Treatment Effect (ATE) in settings where the outcome may be missing (not observed for all individuals). The treatment effect implied by a parametric imputation model is targeted directly through an efficient one-step estimator constructed from its influence function (Nordland et al., 2026).

Usage

```
moi(
  data,
  response.model,
  treatment.model,
  missing.model,
  imputation.model,
  imputation.subset = NULL,
  imputation.augmentation = FALSE,
  imputation.augmentation.model = NULL,
  return.all = FALSE,
  nfolds = 1,
  silent = FALSE,
  stratify = FALSE,
  mc.cores = NULL,
  second.order = TRUE
)
```

Arguments

`data`	A `data.frame` containing all variables required by the models. `data.table` and `tbl_df` objects are automatically coerced to `data.frame`.
`response.model`	A `formula` or `learner` object specifying the response/outcome and the associated baseline adjusted model. If a `formula` is provided, it is automatically wrapped in `learner_glm`. Used to estimate $E[\Delta Y \mid A = a]$.
`treatment.model`	A base R `stats` formula specifying the binary treatment variable. Only an intercept is allowed on the right-hand side, e.g., `A ~ 1`.
`missing.model`	A `formula` or `learner` object specifying the model for the probability of the outcome being observed/non-missing (i.e., $P(\Delta = 1 \mid A = a)$). If a `formula` is provided, it is wrapped in `learner_glm(..., family = binomial())`. Used to estimate $P(\Delta = 0 \mid A = a)$.

`imputation.model`
: A `formula` or `learner_glm` object specifying the missing outcome imputation model. If a `formula` is provided, it is wrapped in `learner_glm`. Used to estimate $E[U(X, A, Z; \theta)|A = a, \Delta = 0]$.

`imputation.subset`
: Optional character string giving an R expression that evaluates to a logical vector indicating which rows of `data` to use when fitting the imputation model. The expression is parsed and evaluated in the context of `data`; for example, `imputation.subset = "!is.na(y)"` restricts the fit to the observed outcomes. If `NULL` (default), all rows are used.

`imputation.augmentation`
: Logical. If `TRUE`, an augmentation term is added to the imputation estimator for improved efficiency. Default is `FALSE`.

`imputation.augmentation.model`
: A `formula`, `learner`, or `NULL` specifying a working model for the conditional imputation mean $E[U(X, A, Z; \theta) \mid W, A]$, used to augment the imputation estimator. Only used when `imputation.augmentation = TRUE`; if `NULL`, the imputation model U itself is used. Default is `NULL`.

`return.all`
: Logical. If `TRUE`, the returned object includes all intermediate estimates in addition to the final ATE estimate. Default is `FALSE`.

`nfolds`
: number of folds (positive integer), or a pre-specified list of fold indices where each element is an integer vector of observation indices forming a partition of `1:nrow(data)`.

`silent`
: suppress all messages and progressbars

`stratify`
: if TRUE the response.model and missing.model (if provided) will be stratified by treatment

`mc.cores`
: (optional) number of cores. parallel::mcmapply used instead of future

`second.order`
: add seconder order term to IF to handle misspecification of outcome models

Details

The `moi` function implements an estimator for the Average Treatment Effect where missing outcomes are imputed using a parametric (glm) model.

The function estimate the target parameter

$$E[\tilde{Y}|A = 1] - E[\tilde{Y}|A = 0],$$

where

$$E[\tilde{Y}|A = a] = E[\Delta Y|A = a] + P(\Delta = 0|A = a) \cdot E[U(X, A, Z; \theta)|A = a, \Delta = 0],$$

and Δ denotes the non-missing indicator, and U denotes the imputation model possibly depending on baseline covariates X, the treatment A, and a post randomization variable Z.

Inference in based on the estimated influence functions (IFs) of the associated (covariate adjusted) one-step estimators.

When `imputation.augmentation = TRUE`, an augmentation term built from the efficient influence function is added, giving an efficient one-step estimator of the treatment effect implied by the imputation model (Nordland et al., 2026). The augmentation uses a working model for the conditional imputation mean $E[U(X, A, Z; \theta) \mid W, A]$ (see `imputation.augmentation.model`) together with the missingness model.

If no observations are missing in an arm a, the imputation contribution for that arm vanishes ($P(\Delta = 0|A = a) = 0$) and $E[\tilde{Y}|A = a] = E[Y|A = a]$. If no observations are missing in any arm, `moi` reduces to a standard `cate` call with `cate.model = ~ 1`.

Value

An object of class `moi.targeted` (inheriting from `targeted`), a list with components:

call The matched call.

estimate A `lava::estimate` object containing the per-arm expected potential outcomes $E[\tilde{Y}|A = a]$ and the ATE contrast $E[\tilde{Y}|A = 1] - E[\tilde{Y}|A = 0]$, with influence-function-based standard errors. Row labels follow the `cate` convention: per-arm rows are labeled $E[\tilde{y}(1)]$ and $E[\tilde{y}(0)]$ (or `E[tildeY(1)]` / `E[tildeY(0)]` in non-UTF-8 locales), and the contrast row is labeled $E[\tilde{y}(1)] - E[\tilde{y}(0)]$.

levels Treatment levels (character).

intermediate (only if `return.all = TRUE`) Intermediate estimates: $E[\Delta Y|A = a]$, $P(\Delta = 0|A = a)$, and $E[U|A = a, \Delta = 0]$.

Standard methods (`print`, `summary`, `coef`, `vcov`, `IC`) are provided.

Author(s)

Andreas Nordland

References

Nordland, A., Holst, K. K., Redek, D., Pipper, C. B. & Iversen, A. T. (2026) One-step Outcome Imputation: An Alternative to Multiple Imputation. arXiv: https://arxiv.org/abs/2606.07174.

See Also

`cate` for Conditional Average Treatment Effect estimation, `learner` for creating learner objects, `lava::estimate` for combining and transforming estimators

Examples

```
sim_moi <- function(n = 1000, ...) {
  w <- rnorm(n)
  a <- rbinom(n, 1, 0.5)
  y <- 1 + a + w + rnorm(n)
  ## outcome observed (delta = 1) with probability depending on w
  delta <- rbinom(n, 1, lava::expit(1 + w))
  y[delta == 0] <- NA
  data.frame(y, a, w)
}
```

```
d <- sim_moi(1000)
## ATE with missing outcomes imputed by a working glm model
moi(data = d,
    response.model = y ~ a + w,
    treatment.model = a ~ 1,
    missing.model = ~ a + w,
    imputation.model = y ~ a + w,
    imputation.subset = "!is.na(y)")
```

`naivebayes` *Naive Bayes classifier*

Description

Naive Bayes Classifier

Usage

```
naivebayes(
  formula,
  data,
  weights = NULL,
  kernel = FALSE,
  laplace.smooth = 0,
  prior = NULL,
  ...
)
```

Arguments

`formula`	Formula with syntax: response ~ predictors | weights
`data`	data.frame
`weights`	optional frequency weights
`kernel`	If TRUE a kernel estimator is used for numeric predictors (otherwise a gaussian model is used)
`laplace.smooth`	Laplace smoothing
`prior`	optional prior probabilities (default estimated from data)
`...`	additional arguments to lower level functions

Value

An object of class 'naivebayes' is returned. See naivebayes-class for more details about this class and its generic functions.

Author(s)

Klaus K. Holst

Examples

```
library(data.table)
data(iris)
m <- naivebayes(Species ~ Sepal.Width + Petal.Length, data = iris)
pr <- predict(m, newdata = iris)

# using weights to reduce the size of the dataset
n <- 5e2
x <- rnorm(n, sd = 2) > 0
y <- rbinom(n, 1, lava::expit(x))
# full data set
d1 <- data.frame(y, x = as.factor(x > 0))
m1 <- naivebayes(y ~ x, data = d1)
# reduced data set
d2 <- data.table(d1)[, .(.N), by = .(y, x)]
m2 <- naivebayes(y ~ x, data = d2, weights = d2$N)
all(predict(m1, d1) == predict(m2, d1))
```

---

naivebayes-class *naivebayes class object*

Description

The functions naivebayes returns an object of the type `naivebayes`.

An object of class '`naivebayes`' is a list with at least the following components:

prior Matrix with prior probabilities, i.e. marginal class probabilities Pr(class)

pcond list of matrices with conditional probabilities of the features given the classes (one list element per class), Pr(x|class)

classes Names (character vector) of the classes

xvar Names of predictors

xmodel Conditional model for each predictor

design Model design object

call The function call which instantiated the object

Value

objects of the S3 class '`naivebayes`'

S3 generics

The following S3 generic functions are available for an object of class `naivebayes`:

`predict` Predict class probabilities for new features data.

`print` Basic print method.

See Also

`naivebayes()`

Examples

```
## See example(naivebayes) for examples
```

---

nondom *Find non-dominated points of a set*

Description

Find the non-dominated point of a set (minima of a point set).

Usage

```
nondom(x, ...)
```

Arguments

x	matrix
...	additional arguments to lower level functions

Details

A point x dominates y if it is never worse and at least in one case strictly better. Formally, let f_i denote the ith coordinate of the condition (objective) function, then for all i: f_i(x)<=f_i(y) and there exists j: f_j(x)<f_j(y).

Based on the algorithm of Kung et al. 1975.

Value

matrix

Author(s)

Klaus Kähler Holst

Examples

```
rbind(
  c(1.0, 0.5),
  c(0.0, 1.0),
  c(1.0, 0.0),
  c(0.5, 1.0),
  c(1.0, 1.0),
  c(0.8, 0.8)) |> nondom()
```

---

pava *Pooled Adjacent Violators Algorithm*

Description

Pooled Adjacent Violators Algorithm

Usage

```
pava(y, x = numeric(0), weights = numeric(0))
```

Arguments

y	response variable
x	(optional) predictor vector (otherwise y is assumed to be a priori sorted according to relevant predictor)
weights	weights (optional) weights

Value

List with index (idx) of jump points and values (value) at each jump point.

Author(s)

Klaus K. Holst

Examples

```
x <- runif(5e3, -5, 5)
pr <- lava::expit(-1 + x)
y <- rbinom(length(pr), 1, pr)
pv <- pava(y, x)
plot(pr ~ x, cex=0.3)
with(pv, lines(sort(x)[index], value, col="red", type="s"))
```

predict.density *Prediction for kernel density estimates*

Description

Kernel density estimator predictions

Usage

```
## S3 method for class 'density'
predict(object, xnew, ...)
```

Arguments

object	density object
xnew	New data on which to make predictions for
...	additional arguments to lower level functions

Author(s)

Klaus K. Holst

predict.naivebayes *Predictions for Naive Bayes Classifier*

Description

Naive Bayes Classifier predictions

Usage

```
## S3 method for class 'naivebayes'
predict(object, newdata, expectation = NULL, threshold = c(0.001, 0.001), ...)
```

Arguments

object	density object
newdata	new data on which to make predictions
expectation	Variable to calculate conditional expectation wrt probabilities from naivebayes classifier
threshold	Threshold parameters. First element defines the threshold on the probabilities and the second element the value to set those truncated probabilities to.
...	Additional arguments to lower level functions

Author(s)

Klaus K. Holst

predict.superlearner *Predict Method for superlearner Fits*

Description

Obtains predictions for ensemble model or individual learners.

Usage

```
## S3 method for class 'superlearner'
predict(object, newdata, all.learners = FALSE, ...)
```

Arguments

object	(superlearner) Fitted superlearner object.
newdata	(data.frame) Data in which to look for variables with which to predict.
all.learners	(logical) If FALSE (default), then return the predictions from the ensemble model. Otherwise, return predictions of from all individual learners.
...	Not used.

Value

numeric (all.learners = FALSE) or matrix (all.learners = TRUE)

predict_filter_bound *Prediction filter bounding predictions to fixed range*

Description

Generates a prediction filter for the predict.filter argument of learner that bounds predictions to a fixed range (lower, upper).

Usage

```
predict_filter_bound(lower = NULL, upper = NULL)
```

Arguments

lower	(numeric) Lower bound, or NULL for no lower bound.
upper	(numeric) Upper bound, or NULL for no upper bound.

Value

A filter generator function (see learner).

Author(s)

Benedikt Sommer

Examples

```
data(cars)
lr <- learner_glm(
  speed ~ dist,
  learner.args = list(predict.filter = predict_filter_bound(upper = 10))
)
lr$estimate(cars)
lr$predict(data.frame(dist = c(10, 50)))
```

---

predict_filter_bound_dynamic

Prediction filter bounding predictions to the observed response range

---

Description

Generates a prediction filter for the `predict.filter` argument of learner that bounds predictions to the range of the response observed in the estimation data (min(response), max(response)).

Usage

```
predict_filter_bound_dynamic(lower = FALSE, upper = FALSE, response = "y")
```

Arguments

`lower`	(logical) If `TRUE`, clamp to the minimum observed response.
`upper`	(logical) If `TRUE`, clamp to the maximum observed response.
`response`	(character) Name of the response column in the data.

Value

A filter generator function (see learner).

Author(s)

Benedikt Sommer

Examples

```
data(cars)
lr <- learner_glm(
  speed ~ dist,
  learner.args = list(
    predict.filter = predict_filter_bound_dynamic(
      upper = TRUE, response = "speed"
    )
  )
)
lr$estimate(cars)
lr$predict(data.frame(dist = c(200, 2000)))
```

RATE *Responder Average Treatment Effect*

Description

Estimation of the Average Treatment Effect among Responders

Usage

```
RATE(
  response,
  post.treatment,
  treatment,
  data,
  M = 5,
  pr.treatment,
  treatment.level,
  preprocess = NULL,
  efficient = TRUE,
  ...
)
```

Arguments

response	(formula or learner) Response model. A formula (e.g., Y ~ D*A) is wrapped in learner_glm with a Gaussian family.
post.treatment	(formula or learner) Post treatment marker model. A formula (e.g., D ~ W) is wrapped in learner_glm with a binomial family.
treatment	Treatment formula (e.g, A ~ 1)
data	data.frame
M	Number of folds in cross-fitting (M=1 is no cross-fitting)
pr.treatment	(optional) Randomization probability of treatment.

`treatment.level`
: Treatment level in binary treatment (default 1)

`preprocess`
: (optional) Data preprocessing function

`efficient`
: If TRUE, the estimate will be efficient. If FALSE, the estimate will be a simple plug-in estimate.

`...`
: Additional arguments to lower level functions

Value

estimate object

Author(s)

Andreas Nordland, Klaus K. Holst

RATE.surv *Responder Average Treatment Effect*

Description

Estimation of the Average Treatment Effect among Responders for Survival Outcomes

Usage

```
RATE.surv(
  response,
  post.treatment,
  treatment,
  censoring,
  tau,
  data,
  M = 5,
  pr.treatment,
  call.response,
  args.response = list(),
  call.censoring,
  args.censoring = list(),
  preprocess = NULL,
  ...
)
```

Arguments

`response`
: Response formula (e.g., Surv(time, event) ~ D + W).

`post.treatment`
: (formula or learner) Post treatment marker model. A formula (e.g., `D ~ W`) is wrapped in learner_glm with a binomial family.

`treatment`	Treatment formula (e.g, A ~ 1)
`censoring`	Censoring formula (e.g., Surv(time, event == 0) ~ D + A + W)).
`tau`	Time-point of interest, see Details.
`data`	data.frame
`M`	Number of folds in cross-fitting (M=1 is no cross-fitting)
`pr.treatment`	(optional) Randomization probability of treatment.
`call.response`	Model call for the response model (e.g. "mets::phreg").
`args.response`	Additional arguments to the response model.
`call.censoring`	Similar to call.response.
`args.censoring`	Similar to args.response.
`preprocess`	(optional) Data preprocessing function
`...`	Additional arguments to lower level functions

Details

Estimation of

$$\frac{P(T \leq \tau|A=1) - P(T \leq \tau|A=0)}{E[D|A=1]}$$

under right censoring based on plug-in estimates of $P(T \leq \tau|A=a)$ and $E[D|A=1]$.

An efficient one-step estimator of $P(T \leq \tau|A=a)$ is constructed using the efficient influence function

$$\frac{I\{A=a\}}{P(A=a)}\Big(\frac{\Delta}{S_0^c(\tilde{T}|X)}I\{\tilde{T} \leq \tau\} + \int_0^\tau \frac{S_0(u|X) - S_0(\tau|X)}{S_0(u|X)S_0^c(u|X)}dM_0^c(u|X)\Big)$$

$$+\Big(1 - \frac{I\{A=a\}}{P(A=a)}\Big)F_0(\tau|A=a,W) - P(T \leq \tau|A=a).$$

An efficient one-step estimator of $E[D|A=1]$ is constructed using the efficient influence function

$$\frac{A}{P(A=1)}\left(D - E[D|A=1,W]\right) + E[D|A=1,W] - E[D|A=1].$$

Value

estimate object

Author(s)

Andreas Nordland, Klaus K. Holst

rcai *Calculate the right censoring augmentation integral*

Description

For a user defined function $H(u|X)$, computes the integral $\int_0^\tau \frac{H(u)|X}{S^c}$ dM^c(u|X), where S^c is the censoring time survival function and M^c is the censoring is the right censoring martingale with the Doob-Meyer decomposition $M^c = N^c - L^c$, where N^c is the counting process $N^c(s) = I\{\tilde{T} \leq s\Delta = 0\}$ and L^c is the compensator $L^c(s) = \int_0^s I\{\tilde{T} \geq u\} d\Lambda^c(u|X)$.

Usage

```
rcai(
  T_model,
  C_model,
  data,
  time,
  event,
  tau,
  H_constructor,
  sample = 0,
  blocksize = 0,
  return_all = FALSE,
  ...
)
```

Arguments

T_model	model for event time
C_model	model for censoring
data	data.frame
time	time variable
event	event variable
tau	stopping time
H_constructor	function H(u|X)
sample	approximate integral by subsampling jump-times
blocksize	evaluate cumhaz in chunks of size blocksize
return_all	if TRUE then bot counting process N and compensator term L are returned
...	additional arguments passed to lower level functions

Value

vector with integral from 0 to all jump-times

Author(s)

Andreas Nordland

`riskreg` *Risk regression*

Description

Risk regression with binary exposure and nuisance model for the odds-product.

Let A be the binary exposure, V the set of covariates, and Y the binary response variable, and define $p_a(v) = P(Y = 1 \mid A = a, V = v), a \in \{0, 1\}$.

The **target parameter** is either the *relative risk*

$$\mathrm{RR}(v) = \frac{p_1(v)}{p_0(v)}$$

or the *risk difference*

$$\mathrm{RD}(v) = p_1(v) - p_0(v)$$

We assume a target parameter model given by either

$$\log\{RR(v)\} = \alpha^t v$$

or

$$\operatorname{arctanh}\{RD(v)\} = \alpha^t v$$

and similarly a working linear **nuisance model** for the *odds-product*

$$\phi(v) = \log\left(\frac{p_0(v)p_1(v)}{(1-p_0(v))(1-p_1(v))}\right) = \beta^t v$$

.

A **propensity model** for $E(A = 1|V)$ is also fitted using a logistic regression working model

$$\operatorname{logit}\{E(A = 1 \mid V = v)\} = \gamma^t v.$$

If both the odds-product model and the propensity model are correct the estimator is efficient. Further, the estimator is consistent in the union model, i.e., the estimator is double-robust in the sense that only one of the two models needs to be correctly specified to get a consistent estimate.

Usage

```
riskreg(
  formula,
  nuisance = ~1,
  propensity = ~1,
  target = ~1,
  data,
```

```
  weights,
  type = "rr",
  optimal = TRUE,
  std.err = TRUE,
  start = NULL,
  mle = FALSE,
  ...
)
```

Arguments

formula	formula (see details below)
nuisance	nuisance model (formula)
propensity	propensity model (formula)
target	(optional) target model (formula)
data	data.frame
weights	optional weights
type	type of association measure (rd og rr)
optimal	If TRUE optimal weights are calculated
std.err	If TRUE standard errors are calculated
start	optional starting values
mle	Semi-parametric (double-robust) estimate or MLE (TRUE gives MLE)
...	additional arguments to unconstrained optimization routine (nlminb)

Details

The 'formula' argument should be given as `response ~ exposure | target-formula | nuisance-formula` or `response ~ exposure | target | nuisance | propensity`

E.g., `riskreg(y ~ a | 1 | x+z | x+z, data=...)`

Alternatively, the model can specifed using the target, nuisance and propensity arguments: `riskreg(y ~ a, target=~1, nuisance=~x+z, ...)`

The `riskreg_fit` function can be used with matrix inputs rather than formulas.

Value

An object of class '`riskreg.targeted`' is returned. See `targeted-class` for more details about this class and its generic functions.

Author(s)

Klaus K. Holst

References

Richardson, T. S., Robins, J. M., & Wang, L. (2017). On modeling and estimation for the relative risk and risk difference. Journal of the American Statistical Association, 112(519), 1121–1130. http://dx.doi.org/10.1080/01621459.2016.1192546

Examples

```
m <- lava::lvm(a[-2] ~ x,
         z ~ 1,
         lp.target[1] ~ 1,
         lp.nuisance[-1] ~ 2*x) |>
     lava::distribution(~a, value=lava::binomial.lvm("logit")) |>
     lava::binomial.rr("y","a","lp.target","lp.nuisance")
d <- sim(m,5e2,seed=1)

I <- model.matrix(~1, d)
X <- model.matrix(~1+x, d)
with(d, riskreg_mle(y, a, I, X, type="rr"))

with(d, riskreg_fit(y, a, nuisance=X, propensity=I, type="rr"))
riskreg(y ~ a | 1, nuisance=~x ,  data=d, type="rr")

## Model with same design matrix for nuisance and propensity model:
with(d, riskreg_fit(y, a, nuisance=X, type="rr"))

## a <- riskreg(y ~ a, target=~z, nuisance=~x,
## propensity=~x, data=d, type="rr")
a <- riskreg(y ~ a | z, nuisance=~x,  propensity=~x, data=d, type="rr")
a
predict(a, d[1:5,])

riskreg(y ~ a, nuisance=~x,  data=d, type="rr", mle=TRUE)
```

`riskreg_cens` *Binary regression models with right censored outcomes*

Description

Binary regression models with right censored outcomes

Usage

```
riskreg_cens(
  response,
  censoring,
  treatment = NULL,
  prediction = NULL,
  data,
  newdata,
  tau,
  type = "risk",
  M = 1,
  call.response = "phreg",
  args.response = list(),
```

```
  call.censoring = "phreg",
  args.censoring = list(),
  preprocess = NULL,
  efficient = TRUE,
  control = list(),
  ...
)
```

Arguments

response	Response formula (e.g., Surv(time, event) ~ D + W).
censoring	Censoring formula (e.g., Surv(time, event == 0) ~ D + A + W)).
treatment	Optional treatment model (learner)
prediction	Optional prediction model (learner)
data	data.frame.
newdata	Optional data.frame. In this case the uncentered influence function evalued in 'newdata' is returned with nuisance parameters obtained from 'data'.
tau	Time-point of interest, see Details.
type	"risk", "treatment", "brier"
M	Number of folds in cross-fitting (M=1 is no cross-fitting).
call.response	Model call for the response model (e.g. "mets::phreg").
args.response	Additional arguments to the response model.
call.censoring	Similar to call.response.
args.censoring	Similar to args.response.
preprocess	(optional) Data pre-processing function.
efficient	If FALSE an IPCW estimator is returned
control	See details
...	Additional arguments to lower level data pre-processing functions.

Details

The one-step estimator depends on the calculation of an integral wrt. the martingale process corresponding to the counting process N(t) = I(C>min(T,tau)). This can be decomposed into an integral wrt the counting process, $dN_c(t)$ and the compensator $d\Lambda_c(t)$ where the latter term can be computational intensive to calculate. Rather than calculating this integral in all observed time points, we can make a coarser evaluation which can be controlled by setting `control=(sample=N)`. With `N=0` the (computational intensive) standard evaluation is used.

Value

estimate object

Author(s)

Klaus K. Holst, Andreas Nordland

score.superlearner *Extract average cross-validated score of individual learners*

Description

Extract average cross-validated score of individual learners

Usage

```
## S3 method for class 'superlearner'
score(x, ...)
```

Arguments

x	(superlearner) Fitted model.
...	Not used.

scoring *Predictive model scoring*

Description

Predictive model scoring

Usage

```
scoring(
  response,
  ...,
  type = "quantitative",
  levels = NULL,
  metrics = NULL,
  weights = NULL,
  names = NULL,
  object = NULL,
  newdata = NULL,
  messages = 1
)
```

Arguments

response	Observed response
...	model predictions (continuous predictions or class probabilities (matrices))
type	continuous or categorical response (the latter is automatically chosen if response is a factor, otherwise a continuous response is assumed)
levels	(optional) unique levels in response variable
metrics	which metrics to report
weights	optional frequency weights
names	(optional) character vector of the model names in the output. If omitted these will be taken from the names of the ellipsis argument (...)
object	optional model object
newdata	(optional) data.frame on which to evaluate the model performance
messages	controls amount of messages/warnings (0: none)

Value

Numeric matrix of dimension m x p, where m is the number of different models and p is the number of model metrics

Examples

```
data(iris)
set.seed(1)
dat <- lava::csplit(iris,2)
g1 <- naivebayes(Species ~ Sepal.Width + Petal.Length, data=dat[[1]])
g2 <- naivebayes(Species ~ Sepal.Width, data=dat[[1]])
pr1 <- predict(g1, newdata=dat[[2]], wide=TRUE)
pr2 <- predict(g2, newdata=dat[[2]], wide=TRUE)
table(colnames(pr1)[apply(pr1,1,which.max)], dat[[2]]$Species)
table(colnames(pr2)[apply(pr2,1,which.max)], dat[[2]]$Species)
scoring(dat[[2]]$Species, pr1=pr1, pr2=pr2)
## quantitative response:
scoring(response=1:10, prediction=rnorm(1:10))
```

SL *SuperLearner wrapper for learner (defunct)*

Description

SL() has been removed. Use learner_sl instead.

Usage

```
SL(...)
```

Arguments

...	Ignored.

Author(s)

Klaus Kähler Holst

---

softmax *Softmax transformation*

Description

Softmax transformation

Usage

```
softmax(x, log = FALSE, ref = TRUE, ...)
```

Arguments

x	Input matrix (e.g., linear predictors of multinomial logistic model)
log	Return on log-scale (default FALSE)
ref	Add reference level (add 0 column to x)
...	Additional arguments to lower level functions

Value

Numeric matrix of dimension n x p, where n= `nrow(x)` and p = `ncol(x) + (ref==TRUE)`

---

solve_ode *Solve ODE*

Description

Solve ODE with Runge-Kutta method (RK4)

Usage

```
solve_ode(ode_ptr, input, init, par = 0)
```

Arguments

ode_ptr	pointer (externalptr) to C++ function or an R function
input	Input matrix. 1st column specifies the time points
init	Initial conditions
par	Parameters defining the ODE (parsed to ode_ptr)

Details

The external point should be created with the function `targeted::specify_ode`.

Value

Matrix with solution

Author(s)

Klaus Kähler Holst

See Also

specify_ode

Examples

```
example(specify_ode)
```

`specify_ode` *Specify Ordinary Differential Equation (ODE)*

Description

Define compiled code for ordinary differential equation.

Usage

```
specify_ode(code, fname = NULL, pname = c("dy", "x", "y", "p"))
```

Arguments

`code`	string with the body of the function definition (see details)
`fname`	Optional name of the exported C++ function
`pname`	Vector of variable names (results, inputs, states, parameters)

Details

The model (`code`) should be specified as the body of of C++ function. The following variables are defined bye default (see the argument `pname`)

dy Vector with derivatives, i.e. the rhs of the ODE (the result).

x Vector with the first element being the time, and the following elements additional exogenous input variables,

y Vector with the dependent variable

p Parameter vector

$y'(t) = f_p(x(t), y(t))$ All variables are treated as Armadillo (http://arma.sourceforge.net/) vectors/matrices.

As an example consider the *Lorenz Equations* $\frac{dx_t}{dt} = \sigma(y_t - x_t)$ $\frac{dy_t}{dt} = x_t(\rho - z_t) - y_t$ $\frac{dz_t}{dt} = x_t y_t - \beta z_t$

We can specify this model as `ode <- 'dy(0) = p(0)*(y(1)-y(0)); dy(1) = y(0)*(p(1)-y(2)); dy(2) = y(0)*y(1)-p(2)*y(2);' dy <- specify_ode(ode)`

As an example of model with exogenous inputs consider the following ODE: $y'(t) = \beta_0 + \beta_1 y(t) + \beta_2 y(t)x(t) + \beta_3 x(t) \cdot t$ This could be specified as `mod <- 'double t = x(0); dy = p(0) + p(1)*y + p(2)*x(1)*y + p(3)*x(1)*t;' dy <- specify_ode(mod)`

Value

pointer (externalptr) to C++ function

Author(s)

Klaus Kähler Holst

See Also

solve_ode

Examples

```
ode <- paste0(
  "dy(0) = p(0)*(y(1)-y(0));",
  "dy(1) = y(0)*(p(1)-y(2));",
  "dy(2) = y(0)*y(1)-p(2)*y(2);", collapse="\n"
)
 # Reduce test time
dy <- specify_ode(ode)
tt <- seq(0, 100, length.out=2e4)
yy <- solve_ode(dy, input=tt, init=c(1, 1, 1), par=c(10, 28, 8/3))
```

---

stratify *Identify Stratification Variables*

Description

This is a special function that identifies stratification variables when they appear on the right hand side of a formula.

Usage

```
stratify(..., na.group = FALSE, shortlabel, sep = ", ")
```

Arguments

...	any number of variables. All must be the same length.
na.group	a logical variable, if TRUE, then missing values are treated as a distinct level of each variable.
shortlabel	if TRUE omit variable names from resulting factor labels. The default action is to omit the names if all of the arguments are factors, and none of them was named.
sep	the character used to separate groups, in the created label

Details

When used outside of a coxph formula the result of the function is essentially identical to the interaction function, though the labels from strata are often more verbose.

Value

a new factor, whose levels are all possible combinations of the factors supplied as arguments.

See Also

survival::strata, learner_stratify, interaction

Examples

```
a <- factor(rep(1:3, 4), labels=c("low", "medium", "high"))
b <- factor(rep(1:4, 3))
levels(stratify(b))
levels(stratify(a, b, shortlabel=TRUE))
```

---

superlearner *Superlearner (stacked/ensemble learner)*

Description

This function creates a predictor object (class learner) from a list of existing learner objects. When estimating this model a stacked prediction will be created by weighting together the predictions of each of the initial learners The weights are learned using cross-validation.

Usage

```
superlearner(
  learners,
  data,
  nfolds = 10,
  meta.learner = metalearner_nnls,
  model.score = mse,
  mc.cores = NULL,
  future.seed = TRUE,
```

```
  silent = TRUE,
  name.prefix = NULL,
  ...
)
```

Arguments

learners	(list) List of learner objects (i.e. learner_glm)
data	(data.frame) Data containing the response variable and covariates.
nfolds	(integer) Number of folds to use in cross-validation to estimate the ensemble weights.
meta.learner	(function) Algorithm to learn the ensemble weights (default non-negative least squares). Must be a function of the response (nx1 vector), y, and the base learner predictions (nxp matrix), pred, with p being the number of learners. The function can optionally accept a model.score argument for scoring the base learners. See metalearner_nnls, metalearner_convexcomb and metalearner_discrete for the available meta learners.
model.score	(function) Method for scoring the predictions of each base learner. Expects two arguments; vector of response variable and prediction from a base learner (see targeted:::mse for additional details).
mc.cores	(integer) If not NULL, then parallel::mcmapply is used with mc.cores number of cores for parallelization instead of the future.apply::future_lapply package. Parallelization is disabled with mc.cores = 1.
future.seed	(logical or integer) Argument passed on to future.apply::future_lapply. If TRUE, then .Random.seed is used if it holds a L'Ecuyer-CMRG RNG seed, otherwise one is created randomly.
silent	(logical) Suppress all messages and progressbars
name.prefix	(character) Prefix used to name learner objects in learners without names. If NULL, then obtain the name from the info field of a learner.
...	Additional arguments to parallel::mclapply or future.apply::future_lapply.

References

Luedtke & van der Laan (2016) Super-Learning of an Optimal Dynamic Treatment Rule, The International Journal of Biostatistics.

See Also

predict.superlearner weights.superlearner score.superlearner

Examples

```
sim1 <- function(n = 5e2) {
   x1 <- rnorm(n, sd = 2)
   x2 <- rnorm(n)
   y <- x1 + cos(x1) + rnorm(n, sd = 0.5**.5)
   data.frame(y, x1, x2)
```

```
}
m <- list(
  "mean" = learner_glm(y ~ 1),
  "glm" = learner_glm(y ~ x1 + x2)
)
sl <- superlearner(m, data = sim1(), nfolds = 2)
predict(sl, newdata = sim1(n = 5))
predict(sl, newdata = sim1(n = 5), all.learners = TRUE)
```

`survival_treatment_level_estfun` *Treatment level estimating functions for survival outcomes under right censoring*

Description

Treatment level estimating functions for survival outcomes under right censoring

Usage

```
survival_treatment_level_estfun(
  type = "risk",
  data,
  tau,
  survival_models,
  treatment_model,
  control
)
```

Arguments

`type`	Character string, outcome of interest: "risk": P(T <= tau|A=a), "surv": P(T > tau|A=a)
`data`	data.frame
`tau`	Numeric, time-point of interest
`survival_models`	List of survival models, see fit_survival_models()
`treatment_model`	Treatment model, see fit_treatment_model()
`control`	List of control parameters, list(sample, blocksize)

Value

List with matrix elements estfun, or, and ipw.

Author(s)

Andreas Nordland

`targeted-class` *targeted class object*

Description

The functions `riskreg` and `ate` returns an object of the type `targeted`.

An object of class '`targeted`' is a list with at least the following components:

estimate An `estimate` object with the target parameter estimates (see `estimate.default`)

opt Object returned from the applied optimization routine

npar number of parameters of the model (target and nuisance)

type String describing the model

Value

objects of the S3 class '`targeted`'

S3 generics

The following S3 generic functions are available for an object of class `targeted`:

`coef` Extract target coefficients of the estimated model.

`vcov` Extract the variance-covariance matrix of the target parameters.

`IC` Extract the estimated influence function.

`print` Print estimates of the target parameters.

`summary` Extract information on both target parameeters and estimated nuisance model.'

See Also

`riskreg`, `ate`

Examples

```
## See example(riskreg) for examples
```

terms.design *Extract model component from design object*

Description

Extract model component from design object

Usage

```
## S3 method for class 'design'
terms(x, specials, ...)
```

Arguments

x	design object
specials	extract variables marked as special (e.g., "offset", "weights", ...)
...	Additional arguments to lower level functions

test_intersection_sw *Signed Wald intersection test*

Description

Calculating test statistics and p-values for the signed Wald intersection test given by

$$SW = \inf_{\theta \in \cap_{i=1}^{n} H_i} \{(\widehat{\theta} - \theta)^\top W \widehat{\Sigma} W (\widehat{\theta} - \theta)\}$$

with individual hypotheses for each coordinate of θ given by $H_i : \theta_j < \delta_j$ for some non-inferiority margin $\delta_j, j = 1, \ldots, n$. #

Usage

```
test_intersection_sw(
  par,
  vcov,
  noninf = 0,
  weights = 1,
  nsim.null = 10000,
  index = NULL,
  control = list(),
  par.name = "theta"
)
```

Arguments

par	(numeric) parameter estimates or estimate object
vcov	(matrix) asymptotic variance estimate
noninf	(numeric) non-inferiority margins
weights	(numeric) optional weights
nsim.null	(integer) number of sample used in Monte-Carlo simulation
index	(integer) subset of parameters to test
control	(list) arguments to alternating projection algorithm. See details section.
par.name	(character) parameter names in output

Details

The constrained least squares problem is solved using Dykstra's algorithm. The following parameters for the optimization can be controlled via the control list argument: dykstra_niter sets the maximum number of iterations (default 500), dykstra_tol convergence tolerance of the alternating projection algorithm (default 1e-7), pinv_tol tolerance for calculating the pseudo-inverse matrix (default length(par).*Machine$double.eps*max(eigenvalue)).

Value

htest object

Author(s)

Klaus Kähler Holst, Christian Bressen Pipper

References

Christian Bressen Pipper, Andreas Nordland & Klaus Kähler Holst (2025) A general approach to construct powerful tests for intersections of one-sided null-hypotheses based on influence functions. arXiv: https://arxiv.org/abs/2511.07096.

See Also

test_zmax_onesided lava::test_wald lava::closed_testing

Examples

```
S <- matrix(c(1, 0.5, 0.5, 2), 2, 2)
thetahat <- c(0.5, -0.2)
test_intersection_sw(thetahat, S, nsim.null = 1e5)
test_intersection_sw(thetahat, S, weights = NULL)

## Not run:
# only on 'lava' >= 1.8.2
e <- estimate(coef = thetahat, vcov = S, labels = c("p1", "p2"))
lava::closed_testing(e, test_intersection_sw, noninf = c(-0.1, -0.1)) |>
  summary()
```

```
## End(Not run)
```

`test_zmax_onesided` *One-sided Zmax test*

Description

Calculating test statistics and p-values for the onesided Zmax / minP test.z

Given parameter estimates $(\widehat{\theta}_1, \ldots, \widehat{\theta}_p)^\top$ with approximate assymptotic covariance matrix $\widehat{S}$, let $Z_i = \frac{\widehat{\theta}_i - \delta_i}{\mathrm{SE}(\widehat{\theta}_i)}$, where $\mathrm{SE}(\widehat{\theta}_i) = \widehat{S}_{ii}$. The Zmax test statistic is then $Z_{max} = \max\{Z_1, \ldots, Z_p\}$, and the null-hypothesis is $H_0 : \theta_i \leq \delta_i, i = 1, \ldots, p$ with non-inferiority margin $\delta_i, i = 1, \ldots, p$, for which the p-value is calculated as $1 - \Phi_R(Z_{max})$ where ϕ_R is the CDF of the multivariate normal distribution with mean zero and correlation matrix $R = \mathrm{diag}(S_{11}^{-0.5}, \ldots, S_{pp}^{-0.5}) S \, \mathrm{diag}(S_{11}^{-0.5}, \ldots, S_{pp}^{-0.5})$.

Usage

```
test_zmax_onesided(par, vcov, noninf = 0, index = NULL, par.name = "theta")
```

Arguments

`par`	(numeric) parameter estimates or `estimate` object
`vcov`	(matrix) asymptotic variance estimate
`noninf`	(numeric) non-inferiority margins
`index`	(integer) subset of parameters to test
`par.name`	(character) parameter names in output

Value

`htest` object

Author(s)

Christian Bressen Pipper, Klaus Kähler Holst

See Also

`test_intersection_sw()` `lava::test_wald()` `lava::closed_testing()`

`truncatedscore` *Scores truncated by death*

Description

Simulated data inspired by the FLOW study (Perkovic 2024)...elt() The following variables are considered in this simulated data set

- `time`: time of first event in years (first major irreversible kidney event or non-related death)
- `status`: event type at first major irreversible kidney event (=1), non-related death (=2), or right censoring (=0)
- `y`: clinical outcome measurement (eGFR) at landmark time (:=2)
- `r`: missing indicator for `y` (1 if observed, 0 if either t<2 or if the outcome was not measured for other reasons)
- `a`: binary treatment (1 := active, 0 := placebo)
- `x1`: covariate, clinical outcome at baseline (eGFR)
- `x2`: covariate, binary treatment usage indicator (1: SGLT2 treatment, 0: none).

The actual failure times and censoring times are also included (`failure.time`, `cens.time`), and the full-data outcome (`y0`) given t>2.

Source

Simulated data

References

Perkovic, V., Tuttle, K. R., Rossing, P., Mahaffey, K. W., Mann, J. F., Bakris, G., Baeres, F. M., Idorn, T., Bosch-Traberg, H., Lausvig, N. L., and Pratley, R. (2024). Effects of semaglutide on chronic kidney disease in patients with type 2 diabetes. New England Journal of Medicine, 391(2):109–121.

Examples

```
data(truncatedscore)
```

`weights.superlearner` *Extract ensemble weights*

Description

Extract ensemble weights

Usage

```
## S3 method for class 'superlearner'
weights(object, ...)
```

Arguments

`object`	(superlearner) Fitted model.
`...`	Not used.

Index

∗ **data**
truncatedscore, 80
.Random.seed, *74*

aipw, 4
alean, 4
ate, 6, *76*
ate.targeted *(*targeted-class*)*, 76

calibrate, *10*
calibrate *(*calibration*)*, 8
calibration, 8, *10*
calibration-class, 10
cate, 11, *52*
cate(), *4*
cross_validated *(*cross_validated-class*)*, 14
cross_validated-class, 14
cumhaz, 15
cv, *14, 15*
cv *(*cv.default*)*, 16
cv.default, 16
cv.learner_sl, *17*, 18, *41*

deprecate_arg_warn, 19
design, 19, *26, 27*, *77*

e1071::svm, *45*
earth::earth, *38*
estimate.default, *14*, *76*
estimate_truncatedscore, 20
expand.list, 22, *29*

family, *30*, *32*, *36*
family.mgcv, *30*
fit_survival_models, 23
future.apply::future_lapply, *74*

glm, *30*, *38*
glmnet, *36*
glmnet::cv.glmnet, *32*
glmnet::predict.cv.glmnet, *32*
grf::probability_forest, *34, 35*
grf::regression_forest, *34, 35*

hal9001::fit_hal, *36*

int_surv, 24
interaction, *73*
isoreg *(*pava*)*, 56
isoregw, *37*
isoregw *(*pava*)*, 56

lava::closed_testing, *78*
lava::closed_testing(), *79*
lava::estimate, *52*
lava::estimate.default, *21*
lava::test_wald, *78*
lava::test_wald(), *79*
learner, *5*, *17*, 24, *29–47*, *52*, *58, 59*, *67*, *73, 74*
learner$new(), *31–33*, *35–39*, *41*, *43–45*, *47*
learner$print(), *26*
learner$update(), *25*
learner_expand_grid, 29
learner_gam, *24*, 30
learner_glm, *24*, 31, *40*, *42*, *50, 51*, *60, 61*, *74*
learner_glmnet_cv, *24*, 32
learner_grf, *24*, 34
learner_hal, *24*, 36
learner_isoreg, *24*, 37
learner_mars, *24*, 38
learner_naivebayes, *24*, 39
learner_sl, *18*, *24*, 40, *48–50*, *69*
learner_stratify, 42, *73*
learner_surv_cox, 43
learner_surv_rf, 44
learner_svm, *24*, 45
learner_xgboost, *24*, *42*, 46

MASS::glm.nb, *31, 32*

metalearner_convexcomb, *41*, 48, *74*
metalearner_discrete, *41*, 48, *74*
metalearner_nnls, *41*, 49, *74*
mets::binregATE, *21*
mgcv::gam, *30, 31*
moi, 50

naivebayes, *39*, 53, *54*
naivebayes(), *55*
naivebayes-class, 54
nnls::nnls, *49*
nondom, 55

parallel::mclapply, *74*
parallel::mcmapply, *17*, *74*
pava, 56
predict.density, 57
predict.naivebayes, 57
predict.superlearner, 58, *74*
predict_filter_bound, *24*, 58
predict_filter_bound_dynamic, *24*, 59

quadprog::solve.QP, *49*

RATE, 60
RATE.surv, 61
rcai, 63
riskreg, 64, *76*
riskreg.targeted *(*targeted-class*)*, 76
riskreg_cens, 66
riskreg_fit *(*riskreg*)*, 64
riskreg_mle *(*riskreg*)*, 64

score.superlearner, 68, *74*
scoring, 68
SL, 69
softmax, 70
solve_ode, 70
specify_ode, 71
stats::glm, *31, 32*
stratify, 72
superlearner, *40, 41*, *48–50*, *58*, 73
survival::strata, *73*
survival::summary.survfit, *16*
survival_treatment_level_estfun, 75

targeted-class, 76
terms, *20*
terms.design, 77
test_intersection_sw, 77
test_intersection_sw(), *79*
test_zmax_onesided, *78*, 79
truncatedscore, 80

weights.superlearner, *74*, 81

xgboost::xgboost, *46, 47*