Package ‘targeted’

February 22, 2024

Type Package

Title Targeted Inference

Version 0.5

Date 2024-02-22

Author Klaus K. Holst [aut, cre],
Andreas Nordland [aut]

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>), 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 (>= 4.0), lava (>= 1.7.0)

Imports data.table, digest, futile.logger, future.apply, optimx, progressr, methods, mets, R6, Rcpp (>= 1.0.0), survival

Suggests grf, mgcv, testthat (>= 0.11), rmarkdown, scatterplot3d, SuperLearner (>= 2.0-28), knitr, xgboost, viridisLite

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

License Apache License (== 2.0)

LinkingTo Rcpp, RcppArmadillo

LazyLoad yes

NeedsCompilation yes

ByteCompile yes

RcppModules riskregmodel

Encoding UTF-8

RoxygenNote 7.3.1

VignetteBuilder knitr

Repository CRAN

Date/Publication 2024-02-22 10:00:21 UTC

R topics documented:

`aipw` *AIPW estimator*

Description

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

Usage

```
aipw(response_model, data, formula = ~1, missing_model, ...)
```

Arguments

`response_model`	Model for the response given covariates (ml_model or formula)
`data`	data.frame
`formula`	design specifying the OLS estimator with outcome given by the EIF
`missing_model`	Optional missing_model (ml_model or formula). By default will use the same design as the response_model.
`...`	arguments to cate

Examples

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

aipw(y0 ~ x, 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 ml_model 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 <- lvm() |>
     distribution(~ y, binomial.lvm()) |>
     regression('a', value=function(l) l) |>
     regression('y', value=function(a,l) a + l)
     if (family$family=="binomial")
        distribution(m, ~a) <- binomial.lvm()
   sim(m, n)
}

library(splines)
f <- binomial()
d <- sim1(1e4, family=f)
e <- alean(response_model=ML(y ~ a + bs(l, df=3), family=binomial),
           exposure_model=ML(a ~ bs(l, df=3), family=f),
           data=d,
```

```
            link = "logit", mc.cores=1, nfolds=1)
e

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

`ate` *AIPW (doubly-robust) estimator for Average Treatement 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 misspecified (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,
  ...
)
```

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`	If TRUE all standard errors are calculated (default TRUE when exposure only has two levels)
`labels`	Optional treatment labels
`...`	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 <- lvm(y ~ a+x, a~x)
distribution(m, ~y) <- binomial.lvm()
m <- ordinal(m, K=4, ~a)
transform(m, ~a) <- factor
d <- sim(m, 1e3, seed=1)
(a <- ate(y~a|a*x|x, data=d))
## ate(y~a, nuisance=~a*x, propensity=~x, ...)

# Comparison with randomized experiment
m0 <- cancel(m, a~x)
lm(y~a-1, 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 <- NB(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 <- NB(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 `targeted`:

`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 via Double Machine Learning

Usage

```
cate(
  treatment,
  response_model,
  propensity_model,
  contrast = c(1, 0),
  data,
  nfolds = 5,
```

```
  type = "dml2",
  silent = FALSE,
  stratify = FALSE,
  mc.cores,
  ...
)
```

Arguments

`treatment`	formula specifying treatment and variables to condition on
`response_model`	formula or ml_model object (formula => glm)
`propensity_model`	formula or ml_model object (formula => glm)
`contrast`	treatment contrast (default 1 vs 0)
`data`	data.frame
`nfolds`	Number of folds
`type`	'dml1' or 'dml2'
`silent`	supress all messages and progressbars
`stratify`	If TRUE the response_model will be stratified by treatment
`mc.cores`	mc.cores Optional number of cores. parallel::mcmapply used instead of future
`...`	additional arguments to future.apply::future_mapply

Value

cate.targeted object

Author(s)

Klaus Kähler Holst

Examples

```
sim1 <- function(n=1e4,
                 seed=NULL,
                 return_model=FALSE, ...) {
suppressPackageStartupMessages(require("lava"))
if (!is.null(seed)) set.seed(seed)
m <- lava::lvm()
regression(m, ~a) <- function(z1,z2,z3,z4,z5)
         cos(z1)+sin(z1*z2)+z3+z4+z5^2
regression(m, ~u) <- function(a,z1,z2,z3,z4,z5)
        (z1+z2+z3)*a + z1+z2+z3 + a
distribution(m, ~a) <- binomial.lvm()
if (return_model) return(m)
lava::sim(m, n, p=par)
}

d <- sim1(200)
```

```
e <- cate(a ~ z1+z2+z3, response=u~., data=d)
e
```

---

cate_link *Conditional Relative Risk estimation*

Description

Conditional average treatment effect estimation via Double Machine Learning

Usage

```
cate_link(
  treatment,
  link = "identity",
  response_model,
  propensity_model,
  importance_model,
  contrast = c(1, 0),
  data,
  nfolds = 5,
  type = "dml1",
  ...
)
```

Arguments

`treatment`	formula specifying treatment and variables to condition on
`link`	Link function
`response_model`	SL object
`propensity_model`	SL object
`importance_model`	SL object
`contrast`	treatment contrast (default 1 vs 0)
`data`	data.frame
`nfolds`	Number of folds
`type`	'dml1' or 'dml2'
`...`	additional arguments to SuperLearner

Value

cate.targeted object

Author(s)

Klaus Kähler Holst & Andreas Nordland

Examples

```
# Example 1:
sim1 <- function(n=1e4,
                 seed=NULL,
                 return_model=FALSE, ...){
suppressPackageStartupMessages(require("lava"))
if (!is.null(seed)) set.seed(seed)
m <- lava::lvm()
distribution(m, ~x) <- gaussian.lvm()
distribution(m, ~v) <- gaussian.lvm(mean = 10)
distribution(m, ~a) <- binomial.lvm("logit")
regression(m, "a") <- function(v, x){.1*v + x}
distribution(m, "y") <- gaussian.lvm()
regression(m, "y") <- function(a, v, x){v+x+a*x+a*v*v}
if (return_model) return(m)
lava::sim(m, n = n)
}

if (require("SuperLearner",quietly=TRUE)) {
  d <- sim1(n = 1e3, seed = 1)
  e <- cate_link(data=d,
           type = "dml2",
           treatment = a ~ v,
           response_model = y~ a*(x + v + I(v^2)),
           importance_model = SL(D_ ~ v + I(v^2)),
           nfolds = 10)
  summary(e) # the true parameters are c(1,1)
}
```

`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
```

`crr` *Conditional Relative Risk estimation*

Description

Conditional Relative Risk estimation via Double Machine Learning

Usage

```
crr(
  treatment,
  response_model,
  propensity_model,
  importance_model,
  contrast = c(1, 0),
  data,
  nfolds = 5,
  type = "dml1",
  ...
)
```

Arguments

`treatment`	formula specifying treatment and variables to condition on
`response_model`	SL object
`propensity_model`	SL object

`importance_model`	SL object
`contrast`	treatment contrast (default 1 vs 0)
`data`	data.frame
`nfolds`	Number of folds
`type`	'dml1' or 'dml2'
`...`	additional arguments to SuperLearner

Value

cate.targeted object

Author(s)

Klaus Kähler Holst & Andreas Nordland

Examples

```
sim1 <- function(n=1e4,
                 seed=NULL,
                 return_model=FALSE, ...){
suppressPackageStartupMessages(require("lava"))
if (!is.null(seed)) set.seed(seed)
m <- lava::lvm()
distribution(m, ~x) <- gaussian.lvm()
distribution(m, ~v) <- gaussian.lvm(mean = 10)
distribution(m, ~a) <- binomial.lvm("logit")
regression(m, "a") <- function(v, x){.1*v + x}
distribution(m, "y") <- gaussian.lvm()
regression(m, "y") <- function(a, v, x){v+x+a*x+a*v*v}
if (return_model) return(m)
lava::sim(m, n = n)
}

d <- sim1(n = 2e3, seed = 1)
if (require("SuperLearner",quietly=TRUE)) {
  e <- crr(data=d,
           type = "dml2",
           treatment = a ~ v,
           response_model = ML(y~ a*(x + v + I(v^2))),
           importance_model = ML(D_ ~ v + I(v^2)),
           propensity_model = ML(a ~ x + v + I(v^2), family=binomial),
           nfolds = 2)
  summary(e) # the true parameters are c(1,1)
}
```

`cv` *Cross-validation*

Description

Generic cross-validation function

Usage

```
cv(
  models,
  data,
  response = NULL,
  nfolds = 5,
  rep = 1,
  weights = NULL,
  modelscore,
  seed = NULL,
  shared = NULL,
  args.pred = NULL,
  args.future = list(),
  mc.cores,
  ...
)
```

Arguments

`models`	List of fitting functions
`data`	data.frame or matrix
`response`	Response variable (vector or name of column in `data`).
`nfolds`	Number of folds (default 5. K=0 splits in 1:n/2, n/2:n with last part used for testing)
`rep`	Number of repetitions (default 1)
`weights`	Optional frequency weights
`modelscore`	Model scoring metric (default: MSE / Brier score). Must be a function with arguments: response, prediction, weights, ...
`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
`...`	Additional arguments parsed to models in models

Details

models should be list of objects of class ml_model. 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

Examples

```
f0 <- function(data,...) lm(...,data=data)
f1 <- function(data,...) lm(Sepal.Length~Species,data=data)
f2 <- function(data,...) lm(Sepal.Length~Species+Petal.Length,data=data)
x <- cv(list(m0=f0,m1=f1,m2=f2),rep=10, data=iris, formula=Sepal.Length~.)
x
```

`design` *Extract design matrix*

Description

Extract design matrix from data.frame and formula

Usage

```
design(formula, data, intercept = FALSE, rm_envir = FALSE, ...)
```

Arguments

`formula`	formula
`data`	data.frame
`intercept`	If FALSE (default) an intercept is not included
`rm_envir`	Remove environment
`...`	additional arguments (e.g, specials such weights, offsets, subset)

Value

An object of class 'design'

Author(s)

Klaus Kähler Holst

`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(...)
```

Arguments

`...`	input variables

Value

list

Author(s)

Klaus Kähler Holst

Examples

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

`ML` *ML model*

Description

Wrapper for ml_model

Usage

```
ML(formula, model = "glm", ...)
```

Arguments

`formula`	formula
`model`	model (sl, rf, pf, glm, ...)
`...`	additional arguments to model object

Details

model 'sl' (SuperLearner::SuperLearner) args: SL.library, cvControl, f<aamily, method example:

model 'grf' (grf::regression_forest) args: num.trees, mtry, sample.weights, sample.fraction, min.node.size, ... example:

model 'grf.binary' (grf::probability_forest) args: num.trees, mtry, sample.weights, ... example:

model 'glm' args: family, weights, offset, ...

`ml_model` *R6 class for prediction models*

Description

Provides standardized estimation and prediction methods

Public fields

`info` Optional information/name of the model

`formals` List with formal arguments of estimation and prediction functions

`formula` Formula specifying response and design matrix

`args` additional arguments specified during initialization

Active bindings

`fit` Active binding returning estimated model object

Methods

Public methods:

- `ml_model$new()`
- `ml_model$estimate()`
- `ml_model$predict()`
- `ml_model$update()`
- `ml_model$print()`
- `ml_model$response()`
- `ml_model$design()`
- `ml_model$opt()`
- `ml_model$clone()`

Method `new()`: Create a new prediction model object

Usage:

```
ml_model$new(
  formula = NULL,
  estimate,
  predict = stats::predict,
  predict.args = NULL,
  info = NULL,
  specials,
  response.arg = "y",
  x.arg = "x",
  ...
)
```

Arguments:

`formula` formula specifying outcome and design matrix

`estimate` function for fitting the model (must be a function response, 'y', and design matrix, 'x'. Alternatively, a function with a single 'formula' argument)

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

`predict.args` optional arguments to prediction function

`info` optional description of the model

`specials` optional additional terms (weights, offset, id, subset, ...) passed to 'estimate'

`response.arg` name of response argument

`x.arg` name of design matrix argument

`...` optional arguments to fitting function

Method `estimate()`**:** Estimation method

Usage:

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

Arguments:

`data` data.frame

`...` Additional arguments to estimation method

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

Method `predict()`**:** Prediction method

Usage:

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

Arguments:

`newdata` data.frame

`...` Additional arguments to prediction method

`object` Optional model fit object

Method `update()`**:** Update formula

Usage:

```
ml_model$update(formula, ...)
```

Arguments:

`formula` formula or character which defines the new response

`...` Additional arguments to lower level functions

Method `print()`**:** Print method

Usage:

```
ml_model$print(...)
```

Arguments:

`...` Additional arguments to lower level functions

Method `response()`**:** Extract response from data

Usage:

```
ml_model$response(data, ...)
```

Arguments:

`data` data.frame

`...` additional arguments to 'design'

Method `design()`**:** Extract design matrix (features) from data

Usage:

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

Arguments:

`data` data.frame

`...` additional arguments to 'design'

Method `opt()`**:** Get options

Usage:

```
ml_model$opt(arg, ...)
```

Arguments:

`arg` name of option to get value of

`...` additional arguments to lower level functions

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Klaus Kähler Holst

Examples

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

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"))


ff <- ml_model$new(estimate=function(y,x) lm.fit(x=x, y=y),
        predict=function(object, newdata) newdata%*%object$coefficients)
## tmp <- ff$estimate(y, x=x)
## ff$predict(x)
```

`NB` *Naive Bayes*

Description

Naive Bayes Classifier

Usage

```
NB(
  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 'NB' is returned. See `NB-class` for more details about this class and its generic functions.

Author(s)

Klaus K. Holst

Examples

```
data(iris)
m2 <- NB(Species ~ Sepal.Width + Petal.Length, data=iris)
pr2 <- predict(m2, newdata=iris)
```

`NB-class` *NB class object*

Description

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

An object of class '`NB`' 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 number of repetitions of the CV

xmodel Number of folds of the CV

model Number of folds of the CV

Value

objects of the S3 class '`NB`'

S3 generics

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

`predict` Predict class probabilities for new features data.

`print` Basic print method.

See Also

`NB`, `NB2`

Examples

```
## See example(NB) 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.NB` *Predictions for Naive Bayes Classifier*

Description

Naive Bayes Classifier predictions

Usage

```
## S3 method for class 'NB'
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 NB 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

RATE *Responder Average Treatment Effect*

Description

Estimation of the Average Treatment Effect among Responders

Usage

```
RATE(
  response,
  post.treatment,
  treatment,
  data,
  family = gaussian(),
  M = 5,
  pr.treatment,
  treatment.level,
 SL.args.response = list(family = gaussian(), SL.library = c("SL.mean", "SL.glm")),
 SL.args.post.treatment = list(family = binomial(), SL.library = c("SL.mean", "SL.glm")),
  preprocess = NULL,
  efficient = TRUE,
  ...
)
```

Arguments

`response`	Response formula (e.g, Y~D*A)
`post.treatment`	Post treatment marker formula (e.g., D~W)
`treatment`	Treatment formula (e.g, A~1)
`data`	data.frame
`family`	Exponential family for response (default gaussian)
`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)
`SL.args.response`	Arguments to SuperLearner for the response model
`SL.args.post.treatment`	Arguments to SuperLearner for the post treatment indicator
`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(),
 SL.args.post.treatment = list(family = binomial(), SL.library = c("SL.mean", "SL.glm")),
  call.censoring,
  args.censoring = list(),
  preprocess = NULL,
  ...
)
```

Arguments

`response`	Response formula (e.g., Surv(time, event) ~ D + W).
`post.treatment`	Post treatment marker formula (e.g., D ~ W).
`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.
`SL.args.post.treatment`	Additional arguments to SuperLearner for the post treatment indicator model.
`call.censoring`	Similar to call.response.
`args.censoring`	Similar to args.response.
`preprocess`	(optional) Data pre-processing function.
`...`	Additional arguments to lower level data pre-processing functions.

Details

Estimation of

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

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

$$\frac{I\{A=a\}}{P(A=a)}\Big(\frac{\Delta}{S_0^c(\tilde{T}|X)}I\{\tilde{T} \leq \tau\} + \int_0^\tau \frac{S_0(u|X) - S_0(\tau|X)}{S_0(u|X)S_0^c(u|X)}dM_0^c(u|X))\Big) + \Big(1 - \frac{I\{A=a\}}{P(A=a)}\Big)F_0(\tau|A=a,W) - P(T \leq \tau|$$

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

`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

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

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

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

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 <- lvm(a[-2] ~ x,
         z ~ 1,
         lp.target[1] ~ 1,
         lp.nuisance[-1] ~ 2*x)
distribution(m,~a) <- binomial.lvm("logit")
m <- binomial.rr(m, "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 (ml_model)
`prediction`	Optional prediction model (ml_model)
`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", "rmst", "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

`scoring` *Predictive model scoring*

Description

Predictive model scoring

Usage

```
scoring(
  response,
  ...,
  type = "quantitative",
  levels = NULL,
  metrics = NULL,
  weights = NULL,
  names = 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 names of models coments (given as ..., alternatively these can be named arguments)
`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 <- csplit(iris,2)
g1 <- NB(Species ~ Sepal.Width + Petal.Length, data=dat[[1]])
g2 <- NB(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 ml_model*

Description

SuperLearner wrapper for ml_model

Usage

```
SL(
  formula = ~.,
  ...,
  SL.library = c("SL.mean", "SL.glm"),
  binomial = FALSE,
  data = NULL
)
```

Arguments

`formula`	Model design
`...`	Additional arguments for SuperLearner::SuperLearner
`SL.library`	character vector of prediction algorithms
`binomial`	boolean specifying binomial or gaussian family (default FALSE)
`data`	Optional data.frame

Value

ml_model object

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

`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 parameters and estimated nuisance model.'

See Also

`riskreg`, `ate`

Examples

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

Index