Package ‘targeted’

October 26, 2021

Type Package
Title Targeted Inference
Version 0.2.0
Date 2021-10-25
Author Klaus K. Holst [aut, cre]
Maintainer Klaus K. Holst <klaus@holst.it>
Depends R (>= 4.0), lava (>= 1.6.10)
Imports data.table, digest, futile.logger, future.apply, progressr, methods, Rcpp (>= 1.0.0), optimx
Suggests grf, mgcv, testthat (>= 0.11), rmarkdown, knitr
URL https://www.targetlib.org/r/
BugReports https://github.com/kkholst/targeted/issues
License Apache License (== 2.0)
LinkingTo Rcpp, RcppArmadillo
LazyLoad yes
NeedsCompilation yes
SystemRequirements C++11
ByteCompile yes
RcppModules riskregmodel
Encoding UTF-8
RoxygenNote 7.1.1
VignetteBuilder knitr
Repository CRAN
Date/Publication 2021-10-26 14:40:02 UTC
Description

Methods for targeted and semiparametric inference including augmented inverse probability weighted estimators for missing data and causal inference.

Author(s)

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

Examples

```r
example(riskreg)
example(ate)
example(calibration)
```
**Description**

Augmented Inverse Probability Weighting estimator for the Average (Causal) Treatment Effect.

**Usage**

```r
ate(
  formula,
  data = parent.frame(),
  weights,
  binary = TRUE,
  nuisance = NULL,
  propensity = nuisance,
  all,
  missing = FALSE,
  labels = NULL,
  ...
)
```

**Arguments**

- **formula**: Formula (see details below)
- **data**: data.frame
- **weights**: optional frequency weights
- **binary**: Binary response (default TRUE)
- **nuisance**: outcome regression formula
- **propensity**: propensity model formula
- **all**: If TRUE all standard errors are calculated (default TRUE when exposure only has two levels)
- **missing**: If TRUE a missing data (AIPW) estimator is returned
- **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

Examples

```r
m <- lvm(y ~ a+x, a~x)
distribution(m,~ a+y) <- binomial.lvm()
d <- sim(m,1e3,seed=1)

a <- ate(y ~ a, nuisance=~x, data=d)
summary(a)

# Multiple treatments
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,1e4)
(a <- ate(y~a|x, data=d))

# Comparison with randomized experiment
m0 <- cancel(m, a~x)
d0 <- sim(m0,2e5)
lm(y~a-1,d0)

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

---

calibration

Calibration (training)

Description

Calibration for multiclassification methods

Usage

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

```r
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(1e5, 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(6e4, beta=beta), k=3)
mod <- NB(y ~ ., data=dd[[1]])
p1 <- predict(mod, newdata=dd[[2]])
cal <- calibration(p1, dd[[2]]$y)
p2 <- predict(mod, newdata=dd[[3]])
pp <- predict(cal, p2)
c <- calibration(pp, dd[[3]]$y)
if (interactive()) {##'
  plot(cal)
  plot(c, 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

```r
## See example(calibration) for examples
```
Cross-validation

**Description**

Generic cross-validation function

**Usage**

```r
cv(
  models,  
data,
  response = NULL,
  K = 5,
  rep = 1,
  weights = NULL,
  modelscore,
  seed = TRUE,
  shared = NULL,
  args.pred = NULL,
  ...
)
```

**Arguments**

- `models` List of fitting functions
- `data` data.frame
- `response` Response variable (vector or name of column in ‘data’)
- `K` Number of folds (default 5, 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: RMSE / Brier score). Must be a function with arguments: response, prediction, weights, ...
- `seed` Random seed (argument parsed to future::future_lapply)
- `shared` Function applied to each fold with results send to each model
- `args.pred` Optional arguments to prediction function (see details below)
- `...` Additional arguments parsed to models in ‘models’

**Details**

...

**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**

```r
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~.)
```

---

**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**

```r
expand.list(...)
```

**Arguments**

...   input variables

**Value**

list

**Author(s)**

Klaus Kähler Holst

**Examples**

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

Naive Bayes Classifier

Usage

\[
\text{NB}(\text{formula, data, weights = NULL, kernel = FALSE, laplace.smooth = 0, prior = NULL, ...})
\]

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

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

The functions \texttt{NB} returns an object of the type \texttt{NB}.

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

- **\texttt{prior}**: Matrix with prior probabilities, i.e. marginal class probabilities \(\text{Pr}(\text{class})\).
- **\texttt{pcond}**: List of matrices with conditional probabilities of the features given the classes (one list element per class), \(\text{Pr}(x|\text{class})\).
- **\texttt{classes}**: Names (character vector) of the classes.
- **\texttt{xvar}**: Number of repetitions of the CV.
- **\texttt{xmodel}**: Number of folds of the CV.
- **\texttt{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 \texttt{NB}:

- **\texttt{predict}**: Predict class probabilities for new features data.
- **\texttt{print}**: Basic print method.

See Also

\texttt{NB2}, \texttt{NB2}

Examples

## See example(NB) for examples
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)
pi <- pava(y, x)
plot(pr ~ x, cex=0.3)
with(pi, 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, ...)

predict(density)
**predict.NB**

**Arguments**

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

**Value**

Numeric vector with predictions.

**Author(s)**

Klaus K. Holst

---

**Description**

Naive Bayes Classifier predictions

**Usage**

```r
## 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
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

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

or the risk difference

$$RD(v) = p_1(v) - p_0(v)$$

We assume a target parameter model given by either

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

or

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

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 $E(A = 1 \mid V)$ is also fitted using a logistic regression working model

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

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

```r
riskreg(
  formula,
  target = NULL,
  nuisance = NULL,
  propensity = nuisance,
  data,
  weights,
  type = "rr",
  optimal = TRUE,
  std.err = TRUE,
  start = NULL,
  semi = TRUE,
  ...
)
```
Arguments

- **formula**: formula (see details below)
- **target**: (optional) target model (formula)
- **nuisance**: nuisance model (formula)
- **propensity**: propensity 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
- **semi**: Semi-parametric (double-robust) estimate (FALSE 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 be specified 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


Examples

```r
m <- lvm(a[-2] ~ x,
        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 | 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, nuisance=x, data=d, type="rr")
a

---

**scoring**

*Predictive model scoring*

**Description**

Predictive model scoring

**Usage**

```r
scoring(
    response, 
    ..., 
    type = "quantitative", 
    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)
- `metrics`: which metrics to report
- `weights`: optional frequency weights
- `names`: optional names of models comments (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**

```r
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))
```

---

**softmax**

*Softmax transformation*

**Description**

Softmax transformation

**Usage**

```r
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
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 Ordinary Differential Equation (ODE)

Description
Define compiled code for ordinary differential equation.

Usage

```r
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 by 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*

\[
\begin{align*}
\frac{dx}{dt} &= \sigma(y - x) \\
\frac{dy}{dt} &= x(\rho - z) - y \\
\frac{dz}{dt} &= xy - \beta z
\end{align*}
\]

We can specify this model as

```r
doxygen<-'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

```r
doxygen<-'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.
- iid  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

* package
  targeted-package, 2

aipw (ate), 3
ate, 3, 20
ate.targeted (targeted-class), 20

calibrate, 7
calibrate (calibration), 4
calibration, 4, 6, 7
calibration-class, 6
cross_validated
  (cross_validated-class), 7
cross_validated-class, 7
cv, 7, 8

estimate.default, 7, 20
expand.list, 9

isoreg (pava), 12
isoregw (pava), 12

NB, 10, 11
NB-class, 11
NB2, 11
NB2 (NB), 10

pava, 12
predict.density, 12
predict.NB, 13

riskreg, 14, 20
riskreg.targeted (targeted-class), 20
riskreg_fit (riskreg), 14
riskreg_mle (riskreg), 14

scoring, 16
softmax, 17
solve_ode, 18
specify_ode, 19

targeted (targeted-package), 2