

# Package ‘targeted’

December 10, 2025

**Type** Package

**Title** Targeted Inference

**Version** 0.7

**Date** 2025-12-09

**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](https://doi.org/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](https://doi.org/10.2202/1557-4679.1008)>), estimators for risk differences and relative risks (Richardson et al. (2017) <[doi:10.1080/01621459.2016.1192546](https://doi.org/10.1080/01621459.2016.1192546)>), assumption lean inference for generalized linear model parameters (Vansteelandt et al. (2022) <[doi:10.1111/rssb.12504](https://doi.org/10.1111/rssb.12504)>).

**Depends** R (>= 4.1)

**Imports** R6, Rcpp (>= 1.0.0), abind, cli, future.apply, lava (>= 1.8.2), methods, mets (>= 1.3.8), 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/>

**License** Apache License (== 2.0)

**LinkingTo** Rcpp, RcppArmadillo

**LazyLoad** yes

**NeedsCompilation** yes

**ByteCompile** yes

**RcppModules** riskregmodel

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**VignetteBuilder** quarto

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

**Repository** CRAN

**Date/Publication** 2025-12-10 06:20:08 UTC

## Contents

|                                     |    |
|-------------------------------------|----|
| aipw . . . . .                      | 3  |
| alean . . . . .                     | 4  |
| ate . . . . .                       | 6  |
| calibration . . . . .               | 8  |
| calibration-class . . . . .         | 9  |
| cate . . . . .                      | 10 |
| cate_link . . . . .                 | 13 |
| constructor_shared . . . . .        | 14 |
| cross_validated-class . . . . .     | 15 |
| crr . . . . .                       | 16 |
| cumhaz . . . . .                    | 17 |
| cv.default . . . . .                | 18 |
| cv.learner_sl . . . . .             | 20 |
| deprecated_argument_names . . . . . | 21 |
| deprecate_arg_warn . . . . .        | 21 |
| design . . . . .                    | 22 |
| estimate_truncatedscore . . . . .   | 23 |
| expand.list . . . . .               | 24 |
| int_surv . . . . .                  | 25 |
| learner . . . . .                   | 26 |
| learner_expand_grid . . . . .       | 30 |
| learner_gam . . . . .               | 31 |
| learner_glm . . . . .               | 33 |
| learner_glmnet_cv . . . . .         | 34 |
| learner_grf . . . . .               | 35 |
| learner_hal . . . . .               | 37 |
| learner_isoreg . . . . .            | 38 |
| learner_mars . . . . .              | 39 |
| learner_naivebayes . . . . .        | 40 |
| learner_sl . . . . .                | 42 |
| learner_stratify . . . . .          | 43 |
| learner_svm . . . . .               | 44 |
| learner_xgboost . . . . .           | 46 |

|                                |           |
|--------------------------------|-----------|
| ML . . . . .                   | 47        |
| ml_model . . . . .             | 48        |
| naivebayes . . . . .           | 49        |
| naivebayes-class . . . . .     | 50        |
| nondom . . . . .               | 51        |
| pava . . . . .                 | 52        |
| predict.density . . . . .      | 52        |
| predict.naivebayes . . . . .   | 53        |
| predict.superlearner . . . . . | 54        |
| RATE . . . . .                 | 54        |
| RATE.surv . . . . .            | 55        |
| riskreg . . . . .              | 57        |
| riskreg_cens . . . . .         | 59        |
| score.superlearner . . . . .   | 61        |
| scoring . . . . .              | 61        |
| SL . . . . .                   | 63        |
| softmax . . . . .              | 63        |
| solve_ode . . . . .            | 64        |
| specify_ode . . . . .          | 65        |
| stratify . . . . .             | 66        |
| superlearner . . . . .         | 67        |
| targeted-class . . . . .       | 68        |
| terms.design . . . . .         | 69        |
| test_intersection_sw . . . . . | 70        |
| test_zmax_onesided . . . . .   | 71        |
| truncatedscore . . . . .       | 72        |
| weights.superlearner . . . . . | 73        |
| <b>Index</b>                   | <b>74</b> |

---

|      |                       |
|------|-----------------------|
| aipw | <i>AIPW estimator</i> |
|------|-----------------------|

---

## 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   | Model for the response given covariates (learner or formula)                  |
| propensity_model | Optional missing data mechanism model (propensity model) (learner or formula) |
| formula          | design specifying the OLS estimator with outcome given by the EIF             |

```
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,1e3,seed=1)

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

---

|       |                                                                          |
|-------|--------------------------------------------------------------------------|
| alean | <i>Assumption Lean inference for generalized linear model parameters</i> |
|-------|--------------------------------------------------------------------------|

---

### 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          suppress all messages and progressbars
mc.cores        mc.cores Optional number of cores. parallel::mcmapply used instead of future
...            additional arguments to future.apply::future_mapapply
```

## Details

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

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

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(1, df=3), family=binomial),
  exposure_model=learner_glm(a ~ bs(1, df=3), family=f),
  data=d,
  link = "logit", mc.cores=1, nfolds=1
)
e

e <- alean(response_model=learner_glm(y ~ a + 1, family=binomial),
  exposure_model=learner_glm(a ~ 1),
  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**

|                              |                                                                                               |
|------------------------------|-----------------------------------------------------------------------------------------------|
| <code>formula</code>         | formula (see details below)                                                                   |
| <code>data</code>            | <code>data.frame</code>                                                                       |
| <code>weights</code>         | optional frequency weights                                                                    |
| <code>offset</code>          | optional offset (character or vector). can also be specified in the formula.                  |
| <code>family</code>          | Exponential family argument for outcome model                                                 |
| <code>nuisance</code>        | outcome regression formula (Q-model)                                                          |
| <code>propensity</code>      | propensity model formula                                                                      |
| <code>all</code>             | when TRUE all standard errors are calculated (default TRUE when exposure only has two levels) |
| <code>labels</code>          | optional treatment labels                                                                     |
| <code>adjust.nuisance</code> | 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 | <i>Calibration (training)</i> |
|-------------|-------------------------------|

---

## Description

Calibration for multiclassification 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")
}

```

**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 with cross-fitting.

**Usage**

```
cate(
  response.model,
  propensity.model,
  cate.model = ~1,
  calibration.model = NULL,
  data,
  contrast,
```

```

    nfolds = 1,
    rep = 1,
    silent = FALSE,
    stratify = FALSE,
    mc.cores = NULL,
    rep.type = c("nuisance", "average"),
    var.type = "IC",
    second.order = TRUE,
    response_model = deprecated,
    cate_model = deprecated,
    propensity_model = deprecated,
    treatment = deprecated,
    ...
  )

```

### Arguments

|                                |                                                                                                                                                                                                                                                                                  |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>response.model</code>    | formula or learner object (formula => learner_glm)                                                                                                                                                                                                                               |
| <code>propensity.model</code>  | formula or learner object (formula => learner_glm)                                                                                                                                                                                                                               |
| <code>cate.model</code>        | formula specifying regression design for conditional average treatment effects                                                                                                                                                                                                   |
| <code>calibration.model</code> | linear calibration model. Specify covariates in addition to predicted potential outcomes to include in the calibration.                                                                                                                                                          |
| <code>data</code>              | data.frame                                                                                                                                                                                                                                                                       |
| <code>contrast</code>          | treatment contrast (default 1 vs 0)                                                                                                                                                                                                                                              |
| <code>nfolds</code>            | number of folds                                                                                                                                                                                                                                                                  |
| <code>rep</code>               | number of replications of cross-fitting procedure                                                                                                                                                                                                                                |
| <code>silent</code>            | suppress all messages and progressbars                                                                                                                                                                                                                                           |
| <code>stratify</code>          | if TRUE the response.model will be stratified by treatment                                                                                                                                                                                                                       |
| <code>mc.cores</code>          | (optional) number of cores. parallel::mcmapply used instead of future                                                                                                                                                                                                            |
| <code>rep.type</code>          | repeated cross-fitting applied by averaging nuisance models (rep.type="nuisance") or by average estimates from each replication (rep.type="average").                                                                                                                            |
| <code>var.type</code>          | 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) |
| <code>second.order</code>      | add second order term to IF to handle misspecification of outcome models                                                                                                                                                                                                         |
| <code>response_model</code>    | Deprecated. Use response.model instead.                                                                                                                                                                                                                                          |
| <code>cate_model</code>        | Deprecated. Use cate.model instead.                                                                                                                                                                                                                                              |
| <code>propensity_model</code>  | Deprecated. Use propensity.model instead.                                                                                                                                                                                                                                        |
| <code>treatment</code>         | Deprecated. Use cate.model instead.                                                                                                                                                                                                                                              |
| <code>...</code>               | 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

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

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

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

## 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.

## 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),
     propensity.model=a~w1+w2,
     data=d)
## CATE
cate(cate.model=~1+w2,
     response.model=y~a*(w1+w2),
     propensity.model=a~w1+w2,
     data=d)

## Not run: ## superlearner example
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,
    propensity.model=learner_glm(a~w1+w2, family=binomial),
    data=d,
    stratify=TRUE)

## End(Not run)

```

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)
}
```

---

constructor\_shared      *Construct a learner*

---

**Description**

Construct a learner

**Arguments**

info                    (character) Optional information to describe the instantiated [learner](#) object.

formula                (formula) Formula specifying response and design matrix.

learner.args           (list) Additional arguments to [learner\\$new\(\)](#).

**Value**

[learner](#) object.

---

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 &amp; 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 = learner_glm(y~ a*(x + v + I(v^2))),
           importance_model = learner_glm(D_ ~ v + I(v^2)),
           propensity_model = learner_glm(a ~ x + v + I(v^2), family=binomial),
           nfolds = 2)
  summary(e) # the true parameters are c(1,1)
}

```

---

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                                                                                                                              |
| 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 <a href="#">survival::summary.survfit</a> ). |
| ...             | 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,  $\exp(-\text{chf})$ .
- dchf:  $t(\text{diff}(\text{rbind}(0, t(\text{chf}))))$

**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 <a href="#">learner</a> 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. <a href="#">parallel::mcmapply</a> 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 <a href="#">learner_sl</a> 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)
```

---

|                                  |
|----------------------------------|
| deprecated_argument_names        |
| <i>Deprecated argument names</i> |

---

**Description**

Deprecated argument names

**Arguments**

- response\_model   Deprecated. Use response.model instead.
- propensity\_model   Deprecated. Use propensity.model instead.
- cate\_model        Deprecated. Use cate.model instead.
- treatment         Deprecated. Use cate.model instead.

---

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| deprecate_arg_warn | <i>Cast warning for deprecated function argument names</i> |
|--------------------|------------------------------------------------------------|

---

**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 | <i>Extract design matrix</i> |
|--------|------------------------------|

---

**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
)
```

**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      | Remove environment                                                                                                                  |
| specials      | character vector specifying functions in the formula that should be marked as special in the <a href="#">terms</a> 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. |

**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

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

**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 evaluate formulas in

**Value**

list

**Author(s)**

Klaus Kähler Holst

**Examples**

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

---

|          |                                                                                                                                                                                                                                  |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int_surv | <i>Integral approximation of a time dependent function. Computes an approximation of <math>\int_{start}^{stop} S(t)dt</math>, where <math>S(t)</math> is a survival function, for a selection of start and stop time points.</i> |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

**Description**

Integral approximation of a time dependent function. Computes an approximation of  $\int_s^{t^s} S(t)dt$ , 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      (logical) If TRUE, integral is 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](#)

## 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.

## Methods

### Public methods:

- [learner\\$new\(\)](#)
- [learner\\$estimate\(\)](#)
- [learner\\$predict\(\)](#)
- [learner\\$update\(\)](#)
- [learner\\$print\(\)](#)
- [learner\\$summary\(\)](#)
- [learner\\$response\(\)](#)
- [learner\\$design\(\)](#)
- [learner\\$opt\(\)](#)
- [learner\\$clone\(\)](#)

**Method** `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,
  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()

`intercept` (logical) include intercept in design matrix

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

*Usage:*

```
learner$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:*

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

*Arguments:*

`newdata` data.frame

`...` Additional arguments to prediction method

`object` Optional model fit object

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

*Usage:*

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

*Arguments:*

formula formula or character which defines the new response

**Method** print(): Print method

*Usage:*

```
learner$print()
```

**Method** 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)
```

**Method** 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](#)

**Method** design(): Generate [design](#) object (design matrix and response) from data

*Usage:*

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

*Arguments:*

data data.frame

... additional arguments to [design](#)

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

*Usage:*

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

*Arguments:*

`arg` name of option to get value of

**Method** `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 <a href="#">learner</a> .                                                                                 |
| args   | (list) Parameters that generate a grid of parameters with <a href="#">expand.list</a> , 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 <a href="#">learner</a> object.                                                                                                                                                                                                                                                                                                                     |
| family       | This is a family object specifying the distribution and link to use in fitting etc (see <a href="#">glm</a> and <a href="#">family</a> ). See <a href="#">family.mgcv</a> 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 ef's optimizer.                                                                                                                    |
| learner.args | (list) Additional arguments to <a href="#">learner\$new()</a> .                                                                                                                                                                                                                                                                                                                                                   |
| ...          | Additional arguments to <a href="#">mgcv::gam</a> .                                                                                                                                                                                                                                                                                                                                                               |

### 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**

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

**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 | <i>Construct a learner</i> |
|-------------------|----------------------------|

---

## 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 <a href="#">learner</a> object.                                                                                                                                                                                                                                                                                                                                                                                                          |
| family  | Either a character string representing one of the built-in families, or else a <code>glm()</code> 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 <code>glmnet</code> chooses its own sequence. Note that this is done for the full model (master sequence), and separately for each fold. The fits are then aligned using the master sequence (see the <code>alignment</code> 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 <a href="#">learner\$new()</a> .                                                                                                                        |
| ...          | 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**

|                              |                                                                                                                                                                                                         |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>formula</code>         | (formula) Formula specifying response and design matrix.                                                                                                                                                |
| <code>num.trees</code>       | Number of trees grown in the forest. Note: Getting accurate confidence intervals generally requires more trees than getting accurate predictions. Default is 2000.                                      |
| <code>min.node.size</code>   | A target for the minimum number of observations in each tree leaf. Note that nodes with size smaller than <code>min.node.size</code> can occur, as in the original random-Forest package. Default is 5. |
| <code>alpha</code>           | A tuning parameter that controls the maximum imbalance of a split. Default is 0.05.                                                                                                                     |
| <code>sample.fraction</code> | Fraction of the data used to build each tree. Note: If <code>honesty = TRUE</code> , these subsamples will further be cut by a factor of <code>honesty.fraction</code> . Default is 0.5.                |
| <code>num.threads</code>     | Number of threads used in training. By default, the number of threads is set to the maximum hardware concurrency.                                                                                       |
| <code>model</code>           | (character) grf model to estimate. Usually <code>regression_forest</code> ( <a href="#">grf::regression_forest</a> ) or <code>probability_forest</code> ( <a href="#">grf::probability_forest</a> ).    |
| <code>info</code>            | (character) Optional information to describe the instantiated <a href="#">learner</a> object.                                                                                                           |
| <code>learner.args</code>    | (list) Additional arguments to <a href="#">learner\$new()</a> .                                                                                                                                         |
| <code>...</code>             | Additional arguments to <code>model</code>                                                                                                                                                              |

**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 <a href="#">learner</a> object.                                                                                                                                                                                                                                                                                                                                                             |
| smoothness_orders | An integer, specifying the smoothness of the basis functions. See details for smoothness_orders for more information.                                                                                                                                                                                                                                                                                                                                     |
| reduce_basis      | An 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 <a href="#">family</a> object (supported by <a href="#">glmnet</a> ) 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 <a href="#">learner\$new()</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| ...          | Additional arguments to <a href="#">hal9001::fit_hal</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

**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 <a href="#">learner</a> object. |
| learner.args | (list) Additional arguments to <a href="#">learner\$new()</a> .                               |
| ...          | Additional arguments to <a href="#">isoregw</a> .                                             |

**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 <a href="#">learner</a> object.                                      |
| degree  | Maximum degree of interaction (Friedman's <i>mi</i> ). 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". |
| <b>The following arguments are for cross validation.</b> |                                                                                                                                                                                                                                                                                                                                        |
| glm                                                      | NULL (default) or a list of arguments to pass on to <a href="#">glm</a> . See the documentation of <a href="#">glm</a> for a description of these arguments See “Generalized linear models” in the vignette. Example:<br>earth(survived~., data=etitanic, degree=2, glm=list(family=binomial))                                         |
| <b>The following arguments are for the forward pass.</b> |                                                                                                                                                                                                                                                                                                                                        |
| learner.args                                             | (list) Additional arguments to <a href="#">learner\$new()</a> .                                                                                                                                                                                                                                                                        |
| ...                                                      | Additional arguments to <a href="#">earth::earth</a> .                                                                                                                                                                                                                                                                                 |

**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 | <i>Construct a learner</i> |
|--------------------|----------------------------|

---

**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 <a href="#">learner</a> 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 <a href="#">learner\$new()</a> .                                |
| ...            | Additional arguments to <a href="#">naivebayes</a> .                                           |

**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 <a href="#">learner</a> objects (i.e. <a href="#">learner_glm</a> )                                                                                                                                                                                                                                                                                                                                                                    |
| info         | (character) Optional information to describe the instantiated <a href="#">learner</a> 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 predictions (nxp matrix), pred, with p being the number of learners. Alternatively, this can be set to the character value "discrete", in which case the Discrete Super-Learner is applied where the model with the lowest risk (model-score) is given weight 1 and all other learners weight 0. |
| model.score  | (function) Model scoring method (see <a href="#">learner</a> )                                                                                                                                                                                                                                                                                                                                                                                        |
| learner.args | (list) Additional arguments to <a href="#">learner\$new()</a> .                                                                                                                                                                                                                                                                                                                                                                                       |
| ...          | Additional arguments to <a href="#">superlearner</a>                                                                                                                                                                                                                                                                                                                                                                                                  |

## 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 | <i>Construct stratified learner</i> |
|------------------|-------------------------------------|

---

## 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) <a href="#">learner</a> 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\_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 <a href="#">learner</a> 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.                                  |
|              | <b>linear:</b> $u'v$<br><b>polynomial:</b> $(\gamma u'v + coef0)^{degree}$<br><b>radial basis:</b> $e^{(-\gamma u-v ^2)}$<br><b>sigmoid:</b> $\tanh(\gamma u'v + coef0)$ |
| learner.args | (list) Additional arguments to <a href="#">learner\$new()</a> .                                                                                                          |
| ...          | Additional arguments to <a href="#">e1071::svm</a> .                                                                                                                     |

**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 | <i>Construct a learner</i> |
|-----------------|----------------------------|

---

## 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.<br>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 <code>xgboost::xgboost</code> for all available options. |
| info         | (character) Optional information to describe the instantiated <code>learner</code> object.                                                   |
| learner.args | (list) Additional arguments to <code>learner\$new()</code> .                                                                                 |
| ...          | Additional arguments to <code>xgboost::xgboost</code> .                                                                                      |

**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))
```

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 |

---

|          |                                       |
|----------|---------------------------------------|
| ml_model | <i>R6 class for prediction models</i> |
|----------|---------------------------------------|

---

**Description**

Replaced by [learner](#)

**Super class**

[targeted::learner](#) -> ml\_model

**Methods****Public methods:**

- [ml\\_model\\$new\(\)](#)
- [ml\\_model\\$clone\(\)](#)

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

*Usage:*

```
ml_model$new(...)
```

*Arguments:*

... deprecated

**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.

---

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  $\Pr(\text{class})$   
**pcond** list of matrices with conditional probabilities of the features given the classes (one list element per class),  $\Pr(x|\text{class})$   
**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  $i$ th coordinate of the condition (objective) function, then for all  $i$ :  $f_i(x) \leq f_i(y)$  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 | <i>Pooled Adjacent Violators Algorithm</i> |
|------|--------------------------------------------|

---

**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 | <i>Prediction for kernel density estimates</i> |
|-----------------|------------------------------------------------|

---

**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 | <i>Predictions for Naive Bayes Classifier</i> |
|--------------------|-----------------------------------------------|

---

**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 | <i>Predict Method for superlearner Fits</i> |
|----------------------|---------------------------------------------|

---

**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)

---

|      |                                           |
|------|-------------------------------------------|
| RATE | <i>Responder Average Treatment Effect</i> |
|------|-------------------------------------------|

---

**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, $Y \sim D*A$ )                                                              |
| post.treatment         | Post treatment marker formula (e.g., $D \sim W$ )                                                  |
| treatment              | Treatment formula (e.g, $A \sim 1$ )                                                               |
| 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., $\text{Surv}(\text{time}, \text{event}) \sim D + W$ ).           |
| post.treatment         | Post treatment marker formula (e.g., $D \sim W$ )                                        |
| treatment              | Treatment formula (e.g., $A \sim 1$ )                                                    |
| censoring              | Censoring formula (e.g., $\text{Surv}(\text{time}, \text{event} == 0) \sim D + A + W$ ). |
| 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 | Arguments to SuperLearner for the post treatment indicator                               |
| 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

$$\begin{aligned} & \frac{I\{A = a\}}{P(A = a)} \left( \frac{\Delta}{S_0^c(\tilde{T}|X)} I\{\tilde{T} \leq \tau\} + \int_0^\tau \frac{S_0(u|X) - S_0(\tau|X)}{S_0(u|X)S_0^c(u|X)} dM_0^c(u|X) \right) \\ & + \left( 1 - \frac{I\{A = a\}}{P(A = a)} \right) F_0(\tau | A = a, W) - P(T \leq \tau | A = a). \end{aligned}$$

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

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

### 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  $p_a(v) = P(Y = 1 | A = a, V = v)$ ,  $a \in \{0, 1\}$ .

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

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

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 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**

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)
```

**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., $\text{Surv}(\text{time}, \text{event}) \sim D + W$ ).                                                                        |
| censoring      | Censoring formula (e.g., $\text{Surv}(\text{time}, \text{event} == 0) \sim D + A + W$ )).                                                             |
| treatment      | Optional treatment model ( <a href="#">learner</a> )                                                                                                  |
| prediction     | Optional prediction model ( <a href="#">learner</a> )                                                                                                 |
| data           | data.frame.                                                                                                                                           |
| newdata        | Optional data.frame. In this case the uncentered influence function evaluated 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 | <i>Extract average cross-validated score of individual learners</i> |
|--------------------|---------------------------------------------------------------------|

---

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 | <i>Predictive model scoring</i> |
|---------|---------------------------------|

---

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  $m \times p$ , 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 | <i>SuperLearner wrapper for learner</i> |
|----|-----------------------------------------|

---

**Description**

SuperLearner wrapper for learner

**Usage**

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

**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                                            |
| info       | model information (optional)                                   |

**Value**

learner object

**Author(s)**

Klaus Kähler Holst

---

|         |                               |
|---------|-------------------------------|
| softmax | <i>Softmax transformation</i> |
|---------|-------------------------------|

---

**Description**

Softmax transformation

**Usage**

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

**Arguments**

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

**Value**

Numeric matrix of dimension  $n \times p$ , where  $n = \text{nrow}(x)$  and  $p = \text{ncol}(x) + (\text{ref} == \text{TRUE})$

---

solve\_ode

*Solve ODE*

---

**Description**

Solve ODE with Runge-Kutta method (RK4)

**Usage**

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

**Arguments**

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

**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 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*  $\frac{dx_t}{dt} = \sigma(y_t - x_t)$   $\frac{dy_t}{dt} = x_t(\rho - z_t) - y_t$   $\frac{dz_t}{dt} = x_t y_t - \beta z_t$

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_nnl,
  model.score = mse,
  mc.cores = NULL,
  future.seed = TRUE,
  silent = TRUE,
  name.prefix = NULL,
  ...
)
```

**Arguments**

|                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>learners</code>     | (list) List of <a href="#">learner</a> objects (i.e. <a href="#">learner_glm</a> )                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>data</code>         | (data.frame) Data containing the response variable and covariates.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>nfolds</code>       | (integer) Number of folds to use in cross-validation to estimate the ensemble weights.                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>meta.learner</code> | (function) Algorithm to learn the ensemble weights (default non-negative least squares). Must be a function of the response (nx1 vector), <code>y</code> , and the predictions (nxp matrix), <code>pred</code> , with <code>p</code> being the number of learners. Alternatively, this can be set to the character value "discrete", in which case the Discrete Super-Learner is applied where the model with the lowest risk (model-score) is given weight 1 and all other learners weight 0. |

|                          |                                                                                                                                                                                                                                                                        |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>model.score</code> | (function) Model scoring method (see <a href="#">learner</a> )                                                                                                                                                                                                         |
| <code>mc.cores</code>    | (integer) If not NULL, then <a href="#">parallel::mcmapply</a> is used with <code>mc.cores</code> number of cores for parallelization instead of the <a href="#">future.apply::future_lapply</a> package. Parallelization is disabled with <code>mc.cores = 1</code> . |
| <code>future.seed</code> | (logical or integer) Argument passed on to <a href="#">future.apply::future_lapply</a> . If TRUE, then <a href="#">.Random.seed</a> is used if it holds a L'Ecuyer-CMRG RNG seed, otherwise one is created randomly.                                                   |
| <code>silent</code>      | (logical) Suppress all messages and progressbars                                                                                                                                                                                                                       |
| <code>name.prefix</code> | (character) Prefix used to name learner objects in learners without names. If NULL, then obtain the name from the <code>info</code> field of a learner.                                                                                                                |
| <code>...</code>         | Additional arguments to <a href="#">parallel::mclapply</a> or <a href="#">future.apply::future_lapply</a> .                                                                                                                                                            |

## 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)
```

---

|                |                              |
|----------------|------------------------------|
| targeted-class | <i>targeted class object</i> |
|----------------|------------------------------|

---

## 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
```

---

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        | <a href="#">design</a> 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

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

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

### 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 | <i>One-sided Zmax test</i> |
|--------------------|----------------------------|

---

**Description**

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

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

**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 | <i>Scores truncated by death</i> |
|----------------|----------------------------------|

---

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 | <i>Extract ensemble weights</i> |
|----------------------|---------------------------------|

---

Description

Extract ensemble weights

Usage

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

Arguments

|        |                              |
|--------|------------------------------|
| object | (superlearner) Fitted model. |
| ...    | Not used.                    |

# Index

\* **data**  
    truncatedscore, 72  
    .Random.seed, 68  
  
aipw, 3  
alean, 4  
ate, 6, 68, 69  
ate.targeted (targeted-class), 68  
  
calibrate, 10  
calibrate (calibration), 8  
calibration, 8, 10  
calibration-class, 9  
cate, 10  
cate(), 4  
cate\_link, 13  
constructor\_shared, 14  
cross\_validated  
    (cross\_validated-class), 15  
cross\_validated-class, 15  
crr, 16  
cumhaz, 17  
cv, 15  
cv (cv.default), 18  
cv.default, 18  
cv.learner\_sl, 20, 20, 42  
  
deprecate\_arg\_warn, 21  
deprecated\_argument\_names, 21  
design, 22, 27, 28, 69  
  
e1071::svm, 44, 45  
earth::earth, 39, 40  
estimate.default, 15, 68  
estimate\_truncatedscore, 23  
expand.list, 24, 30  
  
family, 32, 33, 38  
family.mgcv, 32  
future.apply::future\_lapply, 68  
  
glm, 32, 40  
glmnet, 38  
glmnet::cv.glmnet, 34  
glmnet::predict.cv.glmnet, 34  
grf::probability\_forest, 35, 36  
grf::regression\_forest, 35, 36  
  
hal9001::fit\_hal, 37, 38  
  
int\_surv, 25  
interaction, 67  
isoreg (pava), 52  
isoregw, 38  
isoregw (pava), 52  
  
lava::closed\_testing, 71  
lava::closed\_testing(), 72  
lava::estimate.default, 24  
lava::test\_wald, 71  
lava::test\_wald(), 72  
learner, 4, 14, 15, 19, 26, 30–48, 60, 67, 68  
learner\$new(), 14, 32, 33, 35, 36, 38, 40–42, 45, 47  
learner\$print(), 28  
learner\$update(), 26  
learner\_expand\_grid, 30  
learner\_gam, 26, 31  
learner\_glm, 26, 33, 42, 43, 67  
learner\_glmnet\_cv, 26, 34  
learner\_grf, 26, 35  
learner\_hal, 26, 37  
learner\_isoreg, 26, 38  
learner\_mars, 26, 39  
learner\_naivebayes, 26, 40  
learner\_sl, 20, 26, 42  
learner\_stratify, 43, 67  
learner\_svm, 26, 44  
learner\_xgboost, 26, 43, 46  
  
MASS::glm.nb, 33

`mets::binregATE`, 23  
`mgcv::gam`, 31, 32  
ML, 47  
`ml_model`, 48  
  
`naivebayes`, 40, 41, 49, 50  
`naivebayes()`, 50  
`naivebayes-class`, 50  
`nondom`, 51  
  
`parallel::mclapply`, 68  
`parallel::mcmapply`, 19, 68  
`pava`, 52  
`predict.density`, 52  
`predict.naivebayes`, 53  
`predict.superlearner`, 54, 68  
  
RATE, 54  
`RATE.surv`, 55  
`riskreg`, 57, 68, 69  
`riskreg.targeted (targeted-class)`, 68  
`riskreg_cens`, 59  
`riskreg_fit (riskreg)`, 57  
`riskreg_mle (riskreg)`, 57  
  
`score.superlearner`, 61, 68  
`scoring`, 61  
SL, 63  
`softmax`, 63  
`solve_ode`, 64  
`specify_ode`, 65  
`stats::glm`, 33  
`stratify`, 66  
`superlearner`, 42, 54, 67  
`survival::strata`, 67  
`survival::summary.survfit`, 18  
  
`targeted-class`, 68  
`targeted::learner`, 48  
`terms`, 22  
`terms.design`, 69  
`test_intersection_sw`, 70  
`test_intersection_sw()`, 72  
`test_zmax_onesided`, 71, 71  
`truncatedscore`, 72  
  
`weights.superlearner`, 68, 73  
  
`xgboost::xgboost`, 46, 47