Package ‘model4you’

February 19, 2020

Title  Stratified and Personalised Models Based on Model-Based Trees and Forests
Date  2020-02-12
Version  0.9-5

Description  Model-based trees for subgroup analyses in clinical trials and model-based forests for the estimation and prediction of personalised treatment effects (personalised models). Currently partitioning of linear models, lm(), generalised linear models, glm(), and Weibull models, survreg(), is supported. Advanced plotting functionality is supported for the trees and a test for parameter heterogeneity is provided for the personalised models. For details on model-based trees for subgroup analyses see Seibold, Zeileis and Hothorn (2016) <doi:10.1515/ijb-2015-0032>; for details on model-based forests for estimation of individual treatment effects see Seibold, Zeileis and Hothorn (2017) <doi:10.1177/0962280217693034>.

Depends  R (>= 3.1.0), partykit (>= 1.2-6), grid
Imports  sandwich, stats, methods, ggplot2, Formula, gridExtra, survival
Suggests  mvtnorm, TH.data, psychotools, strucchange, plyr, knitr, ggbeeswarm, MASS
License  GPL-2 | GPL-3
Encoding  UTF-8
RoxygenNote  6.1.1
NeedsCompilation  no
Author  Heidi Seibold [aut, cre], Achim Zeileis [aut], Torsten Hothorn [aut]
Maintainer  Heidi Seibold <heidi@seibold.co>
Repository  CRAN
Date/Publication  2020-02-19 09:20:02 UTC
**R topics documented:**

- `.add_modelinfo` ........................................... 2
- `.modelfit` .................................................. 3
- `.prepare_args` ............................................. 4
- `binomial_glm_plot` ........................................ 4
- `coeftable.survreg` ........................................ 5
- `coxph_plot` .................................................. 6
- `lm_plot` ...................................................... 7
- `logLik.pmtree` ............................................... 8
- `node_pmterminal` .......................................... 9
- `objfun` ...................................................... 10
- `objfun.pmodel_identity` ................................... 11
- `objfun.pmtree` ............................................. 12
- `one_factor` .................................................. 13
- `pmforest` .................................................... 13
- `pmodel` ....................................................... 16
- `pmtest` ....................................................... 18
- `pmtree` ....................................................... 19
- `predict.pmtree` ............................................ 21
- `print.pmtree` .............................................. 23
- `rss` .......................................................... 24
- `survreg_plot` ............................................... 25
- `varimp.pmforest` .................................... 25

**Index** 27

---

### `.add_modelinfo`  
*Add model information to a personalised-model-ctree*

#### Description

For internal use.

#### Usage

`.add_modelinfo(x, nodeids, data, model, coeffun)`

#### Arguments

- `x` constparty object.
- `nodeids` node ids, usually the terminal ids.
- `data` data.
- `model` model.
- `coeffun` function that takes the model object and returns the coefficients. Useful when `coef()` does not return all coefficients (e.g. survreg).
.modelfit

Value

tree with added info. Class still to be added.

Description

Use update function to refit model and extract info such as coef, logLik and estfun.

Usage

.modelfit(model, data, coeffun = coef, weights, control, parm = NULL)

Arguments

model     model object.
data      data.
coeffun   function that takes the model object and returns the coefficients. Useful when
coeff() does not return all coefficients (e.g. survreg).
weights   weights.
control   control options from ctree_control.
parm      which parameters should be used for instability test?

Value

A function returning a list of

coefficients coef.
objfun    logLik.
object    the model object.
converged Did the model converge?
estfun    estfun.
.prepare_args

Prepare input for ctree/cforest from input of pmtree/pmforest

Description

Prepare input for ctree/cforest from input of pmtree/pmforest

Usage

.prepare_args(model, data, zformula, control, ...)

Arguments

model

data

an optional data frame.

zformula

ormula describing which variable should be used for partitioning.

control

ontrol parameters, see ctree_control.

... other arguments.

Value

args to be passed to ctree/cforest.

binomial_glm_plot

Plot for a given logistic regression model (glm with binomial family) with one binary covariate.

Description

Can be used on its own but is also useable as plotfun in node_pmterminal.

Usage

binomial_glm_plot(mod, data = NULL, plot_data = FALSE,

theme = theme_classic(), ...)

Arguments

mod A model of class glm with binomial family.

data optional data frame. If NULL the data stored in mod is used.

plot_data should the data in form of a mosaic type plot be plotted?

theme A ggplot2 theme.

... ignored at the moment.
Examples

```r
set.seed(2017)

# number of observations
n <- 1000

# balanced binary treatment
trt <- factor(rep(c("C", "A"), each = n/2),
               levels = c("C", "A"))

# unbalanced binary treatment
trt <- factor(rep("C", n/4), rep("A", 3*n/4),
              levels = c("C", "A"))

# some continuous variables
x1 <- rnorm(n)
x2 <- rnorm(n)

# linear predictor
lp <- -0.5 + 0.5*I(trt == "A") + 1*I(trt == "A")*I(x1 > 0)

# compute probability with inverse logit function
invlogit <- function(x) 1/(1 + exp(-x))
pr <- invlogit(lp)

# bernoulli response variable
y <- rbinom(n, 1, pr)
dat <- data.frame(y, trt, x1, x2)

# logistic regression model
mod <- glm(y ~ trt, data = dat, family = "binomial")
binomial_glm_plot(mod, plot_data = TRUE)

# logistic regression model tree
ltr <- pmtree(mod)
plot(ltr, terminal_panel = node_pmterminal(ltr,
                                          plotfun = binomial_glm_plot,
                                          confint = TRUE,
                                          plot_data = TRUE))
```

---

**coeftable.survreg**  
*Table of coefficients for survreg model*

**Description**

This function is mostly useful for plotting a pmtree. The generic plotting does not show the estimate and confidence interval of the scale parameter. This one does.
Usage

coeftable.survreg(model, confint = TRUE, digits = 2, intree = FALSE)

Arguments

model  model of class survreg
confint should a confidence interval be computed? Default: TRUE
digits integer, used for formatting numbers. Default: 2
intree is the table plotted within a tree? Default: FALSE

Value

None.

Examples

if(require("survival") & require("TH.data")) {
  ## Load data
data(GBSG2, package = "TH.data")

  ## Weibull model
  bmod <- survreg(Surv(time, cens) ~ horTh, data = GBSG2, model = TRUE)

  ## Coefficient table
  grid.newpage()
  coeftable.survreg(bmod)

  ## partitioned model
  tr <- pmtree(bmod)

  ## plot
  plot(tr, terminal_panel = node_pmterminal(tr, plotfun = survreg_plot,
                             confint = TRUE, coeftable = coeftable.survreg))
}

Description

Can be used on its own but is also useable as plotfun in node_pmterminal.

Usage

coxph_plot(mod, data = NULL, theme = theme_classic(), yrange = NULL)
**lm_plot**

*Density plot for a given lm model with one binary covariate.*

**Description**

Can be used on its own but is also useable as `plotfun` in `node_pmterminal`.

**Usage**

```r
lm_plot(mod, data = NULL, densest = FALSE, theme = theme_classic(),
         yrange = NULL)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| mod      | A model of class `lm`.
| data     | optional data frame. If NULL the data stored in mod is used.
| densest  | should additional to the model density kernel density estimates (see `geom_density`) be computed? |
| theme    | A ggplot2 theme. |
| yrange   | Range of the y variable to be used for plotting. If NULL the range in the data will be used. |

**Details**

In case of an offset, the value of the offset variable will be set to the median of the values in the data.
## Examples

```r
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)
data <- data.frame(weight, group)

lm.D9 <- lm(weight ~ group, data = data)

## example taken from ?lm

## example taken from ?glm (modified version)
data(anorexia, package = "MASS")
anorexia$treatment <- factor(anorexia$Treat != "Cont")
anorex.1 <- glm(Postwt ~ treatment + offset(Prewt),
                family = gaussian, data = anorexia)

lm_plot(anorex.1)
```

---

### logLik.pmtree

**Extract log-Likelihood**

#### Description

Extract sum of log-Likelihood contributions of all terminal nodes. By default the degrees of freedom from the models are used but optionally degrees of freedom for splits can be incorporated.

#### Usage

```r
## S3 method for class 'pmtree'
logLik(object, dfsplit = 0, newdata = NULL,
       weights = NULL, perm = NULL, ...)
```

#### Arguments

- `object` pmtree object.
- `dfsplit` degrees of freedom per selected split.
- `newdata` an optional new data frame for which to compute the sum of objective functions.
- `weights` weights.
- `perm` the number of permutations performed (see `varimp`).
- `...` ignored.

#### Value

Returns an object of class `logLik`.
See Also

`objfun.pmtree` for the sum of contributions to the objective function (not the same when partitioning linear models `lm`)

---

**node_pmterminal**  
*Panel-Generator for Visualization of pmtrees*

**Description**

The plot method for party and constparty objects are rather flexible and can be extended by panel functions. The pre-defined panel-generating function of class grapcon_generator for pmtrees is documented here.

**Usage**

```r
node_pmterminal(obj, coeftable = TRUE, digits = 2, confint = TRUE, plotfun, nid = function(node) paste0(nam[id_node(node)], ", n = ", node$info$nobs), ...)```

**Arguments**

- `obj` an object of class party.
- `coeftable` logical or function. If logical: should a table with coefficients be added to the plot (TRUE/FALSE)? If function: A function comparable to `coeftable.survreg`.
- `digits` integer, used for formatting numbers.
- `confint` Should a confidence interval be computed.
- `plotfun` Plotting function to be used. Needs to be of format `function(mod, data)` where `mod` is the model object. See examples for more details.
- `nid` function to retrieve info on what is plotted as node ids.
- `...` arguments passed on to `plotfun`.

**Examples**

```r
if(require("survival")) {
  ## compute survreg model
  mod_surv <- survreg(Surv(futime, fustat) ~ factor(rx), ovarian,
    dist = 'weibull')
  survreg_plot(mod_surv)

  ## partition model and plot
  tr_surv <- pmtree(mod_surv)
  plot(tr_surv, terminal_panel = node_pmterminal(tr_surv, plotfun = survreg_plot,
    confint = TRUE))
}
```

```r
if(require("survival") & require("TH.data")) {
```
## Load data

data(GBSG2, package = "TH.data")

## Weibull model

bmod <- survreg(Surv(time, cens) ~ horTh, data = GBSG2, model = TRUE)

## Coefficient table

grid.newpage()

coefftable.survreg(bmod)

## partitioned model

tr <- pmtree(bmod)

## plot with specific coeftable

plot(tr, terminal_panel = node_pmterminal(tr, plotfun = survreg_plot,
    confint = TRUE, coeftable = coeftable.survreg))

---

**objfun**  

**Objective function**

**Description**

Get the contributions of an objective function. For `glm` these are the (weighted) log-likelihood contributions, for `lm` the negative (weighted) squared error.

**Usage**

```r
objfun(x, ...)
```

```r
## S3 method for class 'survreg'
objfun(x, newdata = NULL, weights = NULL, ...)
```

```r
## S3 method for class 'lm'
objfun(x, newdata = NULL, weights = NULL, ...)
```

```r
## S3 method for class 'glm'
objfun(x, newdata = NULL, weights = NULL, log = TRUE,
    ...)
```

**Arguments**

- `x`  
  model object.
- `...`  
  further arguments passed on to `objfun` methods.
- `newdata`  
  optional. New data frame. Can be useful for model evaluation / benchmarking.
- `weights`  
  optional. Prior weights. See `glm` or `lm`.
- `log`  
  should the log-Likelihood contributions or the Likelihood contributions be returned?
Value

vector of objective function contributions.

Examples

```r
## Example taken from ?stats::glm
## Dobson (1990) Page 93: Randomized Controlled Trial:
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts))
glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
logLik_contributions <- objfun(glm.D93)
sum(logLik_contributions)
logLik(glm.D93)

if(require("survival")) {
  x <- survreg(Surv(futime, fustat) ~ rx, ovarian, dist = "weibull")
  newdata <- ovarian[3:5,]

  sum(objfun(x))
x$loglik

  objfun(x, newdata = newdata)
}
```

---

**objfun.pmodel_identity**

*Objective function of personalised models*

Description

Get the contributions of an objective function (e.g. likelihood contributions) and the sum thereof (e.g. log-Likelihood).

Usage

```r
## S3 method for class 'pmodel_identity'
objfun(x, ...)

## S3 method for class 'pmodel_identity'
logLik(object, add_df = 0, ...)
```

Arguments

- `x, object` object of class `pmodel_identity` (obtained by `pmodel(..., fun = identity)`).
- `...` additional parameters passed on to `objfun`. 
add_df

It is not very clear what the degrees of freedom are in personalised models. With this argument you can add/subtract degrees of freedom at your convenience. Default is 0 which means adding up the degrees of freedom of all individual models. For examples see `pmodel`.

---

### objfun.pmtree

**Objective function of a given pmtree**

**Description**

Returns the contributions to the objective function or the sum thereof (if `sum = TRUE`).

**Usage**

```r
## S3 method for class 'pmtree'
objfun(x, newdata = NULL, weights = NULL,
       perm = NULL, sum = FALSE, ...)
```

**Arguments**

- `x` pmtree object.
- `newdata` an optional new data frame for which to compute the sum of objective functions.
- `weights` weights.
- `perm` the number of permutations performed (see `varimp`).
- `sum` should the sum of objective functions be computed.
- `...` passed on to `predict.party`.

Note that `objfun.pmtree(x, sum = TRUE)` is much faster than `sum(objfun.pmtree(x))`.

**Value**

objective function or the sum thereof

**Examples**

```r
## generate data
set.seed(2)
n <- 1000
trt <- factor(rep(1:2, each = n/2))
age <- sample(40:60, size = n, replace = TRUE)
Eff <- -1 + I(trt == 2) + I(trt == 2) * I(age > 50)
expit <- function(x) 1/(1 + exp(-x))
success <- rbinom(n = n, size = 1, prob = expit(Eff))
dat <- data.frame(success, trt, age)

## compute base model
bmod1 <- glm(success ~ trt, data = dat, family = binomial)
```
One-factor analysis

```r
## compute tree
(tr1 <- pmtree(bmod1, data = dat))

## compute log-likelihood
logLik(tr1)
objfun(tr1, newdata = dat, sum = TRUE)
objfun(tr1, sum = TRUE)

## log-likelihood contributions of first
## 5 observations
nd <- dat[1:5, ]
objfun(tr1, newdata = nd)
```

---

### one_factor

Check if model has only one factor covariate.

**Description**

See [https://stackoverflow.com/questions/50504386/check-that-model-has-only-one-factor-covariate/50514499#50514499](https://stackoverflow.com/questions/50504386/check-that-model-has-only-one-factor-covariate/50514499#50514499)

**Usage**

```r
one_factor(object)
```

**Arguments**

- `object`: model.

**Value**

Returns `TRUE` if model has a single factor covariate, `FALSE` otherwise.

---

### pmforest

Compute model-based forest from model.

**Description**

Input a parametric model and get a forest.
Usage

pmforest(model, data = NULL, zformula = ~., ntree = 500L, 
    perturb = list(replace = FALSE, fraction = 0.632), mtry = NULL, 
    applyfun = NULL, cores = NULL, control = ctree_control(teststat = 
        "quad", testtype = "Univ", mincriterion = 0, saveinfo = FALSE, lookahead 
        = TRUE, ...), trace = FALSE, ...)

## S3 method for class 'pmforest'
gettree(object, tree = 1L, saveinfo = TRUE, 
    coeffun = coef, ...)

Arguments

model \hspace{1cm} a model object. The model can be a parametric model with a single binary 
    covariate.
data \hspace{1cm} data. If NULL the data from the model object are used.zformula \hspace{1cm} formula describing which variable should be used for partitioning. Default is to 
    use all variables in data that are not in the model (i.e. ~ .).
ntree \hspace{1cm} number of trees.perturb \hspace{1cm} a list with arguments replace and fraction determining which type of resampling 
    with replace = TRUE referring to the n-out-of-n bootstrap and replace = FALSE 
    to sample splitting. fraction is the number of observations to draw without re-
    placement.mtry \hspace{1cm} number of input variables randomly sampled as candidates at each node (Default 
    NULL corresponds to ceiling(sqrt(nvar))). Bagging, as special case of a 
    random forest without random input variable sampling, can be performed by 
    setting mtry either equal to Inf or equal to the number of input variables.applyfun \hspace{1cm} see cforest.cores \hspace{1cm} see cforest.control \hspace{1cm} control parameters, see ctree_control.trace \hspace{1cm} a logical indicating if a progress bar shall be printed while the forest grows.
... \hspace{1cm} additional parameters passed on to model fit such as weights.object \hspace{1cm} an object returned by pmforest.tree \hspace{1cm} an integer, the number of the tree to extract from the forest.saveinfo \hspace{1cm} logical. Should the model info be stored in terminal nodes?coeffun \hspace{1cm} function that takes the model object and returns the coefficients. Useful when 
    coef() does not return all coefficients (e.g. survreg).

Value

cforest object

See Also

gettree
Examples

library("model4you")

if(require("mvtnorm") & require("survival")) {
  ## function to simulate the data
  sim_data <- function(n = 500, p = 10, beta = 3, sd = 1) {
    ## treatment
    lev <- c("C", "A")
    a <- rep(factor(lev, labels = lev, levels = lev), length = n)
    ## correlated z variables
    sigma <- diag(p)
    sigma[sigma == 0] <- 0.2
    ztemp <- rmvnorm(n, sigma = sigma)
    z <- (pnorm(ztemp) * 2 * pi) - pi
    colnames(z) <- paste0("z", 1:ncol(z))
    z1 <- z[, 1]
    ## outcome
    y <- 7 + 0.2 * (a %in% "A") + beta * cos(z1) * (a %in% "A") + rnorm(n, 0, sd)
    data.frame(y = y, a = a, z)
  }
  simdata <- simdata_s <- sim_data(p = 5, beta = beta, n = ntrain)
  tsimdata <- tsimdata_s <- sim_data(p = 5, beta = beta, n = ntest)
  simdata_s$cens <- rep(1, ntrain)
  tsimdata_s$cens <- rep(1, ntest)
  ## base model
  basemodel_lm <- lm(y ~ a, data = simdata)
  ## forest
  frst_lm <- pmforest(basemodel_lm, ntree = 20,
                      perturb = list(replace = FALSE, fraction = 0.632),
                      control = ctree_control(mincriterion = 0))
  ## personalised models
  # (1) return the model objects
  pmmodels_lm <- pmodel(x = frst_lm, newdata = tsimdata, fun = identity)
  class(pmmodels_lm)
  # (2) return coefficients only (default)
  coefs_lm <- pmodel(x = frst_lm, newdata = tsimdata)
  # compare predictive objective functions of personalised models versus
# base model
sum(objfun(pmodels_lm)) # -RSS personalised models
sum(objfun(basemodel_lm, newdata = tsimdata)) # -RSS base model

if(require("ggplot2")) {
  ## dependence plot
dp_lm <- cbind(coefs_lm, tsimdata)
ggplot(tsimdata) +
  stat_function(fun = function(z1) 0.2 + beta * cos(z1),
    aes(color = "true treatment\neffect")) +
  geom_point(data = dp_lm,
    aes(y = aA, x = z1, color = "estimates lm"),
    alpha = 0.5) +
  ylab("treatment effect") +
  xlab("patient characteristic z1")
}

---

**pmodel**

**Personalised model**

**Description**

Compute personalised models from cforest object.

**Usage**

```r
pmodel(x = NULL, model = NULL, newdata = NULL, OOB = TRUE,
    fun = coef, return_attr = c("modelcall", "data", "similarity"))
```

**Arguments**

- **x**: cforest object or matrix of weights.
- **model**: model object. If NULL the model in `x$info$model` is used.
- **newdata**: new data. If NULL cforest learning data is used. Ignored if x is a matrix.
- **OOB**: In case of using the learning data, should patient similarities be computed out of bag?
- **fun**: function to apply on the personalised model before returning. The default `coef` returns a matrix of personalised coefficients. For returning the model objects use `identity`.
- **return_attr**: which attributes to add to the object returned. If it contains "modelcall" the call of the base model is returned, if it contains "data" the data, and if it contains "similarity" the matrix of similarity weights is added.
Value

depends on fun.

Examples

library("model4you")

if(require("mvtnorm") & require("survival")) {

  ## function to simulate the data
  sim_data <- function(n = 500, p = 10, beta = 3, sd = 1){

    ## treatment
    lev <- c("C", "A")
    a <- rep(factor(lev, labels = lev, levels = lev), length = n)

    ## correlated z variables
    sigma <- diag(p)
    sigma[sigma == 0] <- 0.2
    ztemp <- rmvnorm(n, sigma = sigma)
    z <- (pnorm(ztemp) * 2 * pi) - pi
    colnames(z) <- paste0("z", 1:ncol(z))
    z1 <- z[,1]

    ## outcome
    y <- 7 + 0.2 * (a %in% "A") + beta * cos(z1) * (a %in% "A") + rnorm(n, 0, sd)

    data.frame(y = y, a = a, z)
  }

  ## simulate data
  set.seed(123)
  beta <- 3
  ntrain <- 500
  ntest <- 50
  simdata <- simdata_s <- sim_data(p = 5, beta = beta, n = ntrain)
  tsimdata <- tsimdata_s <- sim_data(p = 5, beta = beta, n = ntest)
  simdata_s$cens <- rep(1, ntrain)
  tsimdata_s$cens <- rep(1, ntest)

  ## base model
  basemodel_lm <- lm(y ~ a, data = simdata)

  ## forest
  frst_lm <- pmforest(basemodel_lm, ntree = 20,
                      perturb = list(replace = FALSE, fraction = 0.632),
                      control = ctree_control(mincriterion = 0))

  ## personalised models
  # (1) return the model objects
  pmodels_lm <- pmodel(x = frst_lm, newdata = tsimdata, fun = identity)
  class(pmodels_lm)
# (2) return coefficients only (default)
coefs_lm <- pmodel(x = frst_lm, newdata = tsimdata)

# compare predictive objective functions of personalised models versus
# base model
sum(objfun(pmodels_lm)) # -RSS personalised models
sum(objfun(basemodel_lm, newdata = tsimdata)) # -RSS base model

if(require("ggplot2")) {
  ## dependence plot
dp_lm <- cbind(coefs_lm, tsimdata)
ggplot(tsimdata) +
  stat_function(fun = function(z1) 0.2 + beta * cos(z1),
                aes(color = "true treatment\neffect")) +
  geom_point(data = dp_lm,
             aes(y = aA, x = z1, color = "estimates lm"),
             alpha = 0.5) +
  ylab("treatment effect") +
  xlab("patient characteristic z1")
}

pmtest

Test if personalised models improve upon base model.

Description

This is a rudimentary test if there is heterogeneity in the model parameters. The null-hypothesis is: the base model is the correct model.

Usage

pmtest(forest, pmodels = NULL, data = NULL, B = 100)

## S3 method for class 'heterogeneity_test'
plot(x, ...)

Arguments

forest pmforest object.
pmodels pmodel_identity object (pmodel(..., fun = identity)).
data data.
B number of bootstrap samples.
x object of class heterogeneity_test.
... ignored.
Value

list where the first element is the p-value and the second element is a data.frame with all necessary info to compute the p-value.

The test statistic is the difference in objective function between the base model and the personalised models. To compute the distribution under the Null we draw parametric bootstrap samples from the base model. For each bootstrap sample we again compute the difference in objective function between the base model and the personalised models. If the difference in the original data is greater than the difference in the bootstrap samples, we reject the null-hypothesis.

Examples

```r
## Not run:
set.seed(123)
n <- 160
trt <- factor(rep(0:1, each = n/2))
y <- 4 + (trt == 1) + rnorm(n)
z <- matrix(rnorm(n * 2), ncol = 2)

dat <- data.frame(y, trt, z)

mod <- lm(y ~ trt, data = dat)

## Note that ntree should usually be higher
frst <- pmforest(mod, ntree = 20)
pmods <- pmodel(frst, fun = identity)

## Note that B should be at least 100
## The low B is just for demonstration purposes.
tst <- pmtest(forest = frst,
              pmodels = pmods,
              B = 10)
tst$pvalue
tst
plot(tst)

## End(Not run)
```

pmtree

Compute model-based tree from model.

Description

Input a parametric model and get a model-based tree.

Usage

```r
pmtree(model, data = NULL, zformula = ~., control = ctree_control(),
       coeffun = coef, ...)```
Arguments

- **model**: a model object. The model can be a parametric model with a binary covariate.
- **data**: data. If NULL (default) the data from the model object are used.
- **zformula**: formula describing which variable should be used for partitioning. Default is to use all variables in data that are not in the model (i.e. ~ .).
- **control**: control parameters, see `ctree_control`.
- **coeffun**: function that takes the model object and returns the coefficients. Useful when `coef()` does not return all coefficients (e.g. `survreg`).
- **...**: additional parameters passed on to model fit such as weights.

Details

Sometimes the number of participant in each treatment group needs to be of a certain size. This can be accomplished by setting `control$converged`. See example below.

Value

ctree object

Examples

```r
if(require("TH.data") & require("survival")) {
  ## base model
  bmod <- survreg(Surv(time, cens) ~ horTh, data = GBSG2, model = TRUE)
  survreg_plot(bmod)

  ## partitioned model
  tr <- pmtree(bmod)
  plot(tr, terminal_panel = node_pmterminal(tr, plotfun = survreg_plot, confint = TRUE))
  summary(tr)
  summary(tr, node = 1:2)
  logLik(bmod)
  logLik(tr)

  ## Sometimes the number of participant in each treatment group needs to be of a certain size. This can be accomplished using converged

  ## Each treatment group should have more than 33 observations
  ctrl <- ctree_control(lookahead = TRUE)
  ctrl$converged <- function(mod, data, subset) {
    all(table(data$horTh[subset]) > 33)
  }
  tr2 <- pmtree(bmod, control = ctrl)
  plot(tr2, terminal_panel = node_pmterminal(tr, plotfun = survreg_plot, confint = TRUE))
}
```
summary(tr2[[5]]$data$horTh)
}

if(require("psychotools")) {
data("MathExam14W", package = "psychotools")

## scale points achieved to [0, 100] percent
MathExam14W$tests <- 100 * MathExam14W$tests/26
MathExam14W$pcorrect <- 100 * MathExam14W$nsolved/13

## select variables to be used
MathExam <- MathExam14W[, c("pcorrect", "group", "tests", "study",
"attempt", "semester", "gender")]

## compute base model
bmod_math <- lm(pcorrect ~ group, data = MathExam)
lm_plot(bmod_math, densest = TRUE)

## compute tree
(tr_math <- pmtree(bmod_math, control = ctree_control(maxdepth = 2)))
plot(tr_math, terminal_panel = node_pmterminal(tr_math, plotfun = lm_plot,
confint = FALSE))
plot(tr_math, terminal_panel = node_pmterminal(tr_math, plotfun = lm_plot,
densest = TRUE,
confint = TRUE))

## predict
newdat <- MathExam[1:5, ]

# terminal nodes
(nodes <- predict(tr_math, type = "node", newdata = newdat))

# response
(pr <- predict(tr_math, type = "pass", newdata = newdat))

# response including confidence intervals, see ?predict.lm
(pr1 <- predict(tr_math, type = "pass", newdata = newdat,
predict_args = list(interval = "confidence")))
}

---

**predict.pmtree**  

**pmtree predictions**

**Description**

Compute predictions from pmtree object.
Usage

```r
## S3 method for class 'pmtree'
predict(object, newdata = NULL, type = "node",
         predict_args = list(), perm = NULL, ...)
```

Arguments

- **object**: pmtree object.
- **newdata**: an optional data frame in which to look for variables with which to predict, if omitted, `object$data` is used.
- **type**: character denoting the type of predicted value. The terminal node is returned for "node". If `type = "pass"` the model predict method is used and arguments can be passed to it via `predict_args`. If `type = "coef"` the the model coefficients are returned.
- **predict_args**: If `type = "pass"` arguments can be passed on to the model predict function.
- **perm**: an optional character vector of variable names (or integer vector of variable location in newdata). Splits of nodes with a primary split in any of these variables will be permuted (after dealing with surrogates). Note that surrogate split in the perm variables will no be permuted.
- **...**: passed on to `predict.party` (e.g. `perm`).

Value

- **predictions**

Examples

```r
if(require("psychotools")) {
  data("MathExam14W", package = "psychotools")

  ## scale points achieved to [0, 100] percent
  MathExam14W$tests <- 100 * MathExam14W$tests/26
  MathExam14W$pcorrect <- 100 * MathExam14W$nsolved/13

  ## select variables to be used
  MathExam <- MathExam14W[, c("pcorrect", "group", "tests", "study",
                             "attempt", "semester", "gender")]

  ## compute base model
  bmod_math <- lm(pcorrect ~ group, data = MathExam)
  lm_plot(bmod_math, densest = TRUE)

  ## compute tree
  (tr_math <- pmtree(bmod_math, control = ctree_control(maxdepth = 2)))
  plot(tr_math, terminal_panel = node_pmterminal(tr_math, plotfun = lm_plot,
                                                confint = FALSE))
  plot(tr_math, terminal_panel = node_pmterminal(tr_math, plotfun = lm_plot,
```
print.pmtree

## predict
newdat <- MathExam[1:5, ]

# terminal nodes
(nodes <- predict(tr_math, type = "node", newdata = newdat))

# response
(pr <- predict(tr_math, type = "pass", newdata = newdat))

# response including confidence intervals, see ?predict.lm
(pr1 <- predict(tr_math, type = "pass", newdata = newdat,
                predict_args = list(interval = "confidence")))

---

print.pmtree  

Methods for pmtree

### Description

Print and summary methods for pmtree objects.

### Usage

```r
## S3 method for class 'pmtree'
print(x, node = NULL, FUN = NULL,
      digits = getOption("digits") - 4L, footer = TRUE, ...)

## S3 method for class 'pmtree'
summary(object, node = NULL, ...)

## S3 method for class 'summary.pmtree'
print(x, digits = 4, ...)

## S3 method for class 'pmtree'
coef(object, node = NULL, ...)
```

### Arguments

- **x**: object.
- **node**: node number, if any.
- **FUN**: formatinfo function.
- **digits**: number of digits.
- **footer**: should footer be included?
- **...**: further arguments passed on to `print.party`.
- **object**: object.
Value

print

### Description

Returns the sum of the squared residuals for a given object.

### Usage

```r
rss(object, ...)
```

#### Default S3 method:
```r
rss(object, ...)
```

### Arguments

- `object` : model object.
- `...` : passed on to specific methods.

### Value

sum of the squared residuals.

### Examples

```r
# example from ?lm
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)
lm.D9 <- lm(weight ~ group)
rss(lm.D9)
```
**survreg_plot**

Survival plot for a given survreg model with one binary covariate.

**Description**

Can be used on its own but is also usable as plotfun in `node_pmterminal`.

**Usage**

```
survreg_plot(mod, data = NULL, theme = theme_classic(), yrange = NULL)
```

**Arguments**

- **mod**: A model of class survreg.
- **data**: optional data frame. If NULL the data stored in mod is used.
- **theme**: A ggplot2 theme.
- **yrange**: Range of the y variable to be used for plotting. If NULL it will be 0 to max(y).

**Examples**

```r
if(require("survival")) {
  survreg_plot(survreg(Surv(futime, fustat) ~ factor(rx), ovarian))
}
```

---

**varimp.pmforest**

Variable Importance for pmforest

**Description**

See `varimp.cforest`.

**Usage**

```
## S3 method for class 'pmforest'
varimp(object, nperm = 1L, OOB = TRUE, risk = function(x, ...) -objfun(x, sum = TRUE, ...), conditional = FALSE, threshold = 0.2, ...)
```
Arguments

object DESCRIPTION.
nperm the number of permutations performed.
OOB a logical determining whether the importance is computed from the out-of-bag sample or the learning sample (not suggested).
risk the risk to be evaluated. By default the objective function (e.g. log-Likelihood) is used.
conditional a logical determining whether unconditional or conditional computation of the importance is performed.
threshold the value of the test statistic or 1 - p-value of the association between the variable of interest and a covariate that must be exceeded inorder to include the covariate in the conditioning scheme for the variable of interest (only relevant if conditional = TRUE).
... passed on to objfun.

Value

A vector of ’mean decrease in accuracy’ importance scores.
Index

.add_modelinfo, 2
.modelfit, 3
.prepare_args, 4

binomial_glm_plot, 4

cforest, 14
coef.pmtree (print.pmtree), 23
coeftable.survreg, 5, 9
coxph_plot, 6
ctree_control, 4, 14, 20

data

geom_density, 7
gettree, 14
gettree.pmforest (pmforest), 13
glm, 10

lm, 9, 10
lm_plot, 7
logLik, 8
logLik.pmodel_identity
    (objfun.pmodel_identity), 11
logLik.pmtree, 8

node_pmterminal, 4, 6, 7, 9, 25

objfun, 10, 11, 26
objfun.pmodel_identity, 11
objfun.pmtree, 9, 12
one_factor, 13

plot.heterogeneity_test (pmtest), 18
pmforest, 13
pmodel, 12, 16
pmtest, 18
pmtree, 19
predict.party, 12
predict.pmtree, 21
print.party, 23
print.pmtree, 23
print.summary.pmtree (print.pmtree), 23

rss, 24
summary.pmtree (print.pmtree), 23
survreg, 6
survreg_plot, 25

varimp, 8, 12
varimp.cforest, 25
varimp.pmforest, 25