Package ‘medflex’

September 3, 2018

Title Flexible Mediation Analysis Using Natural Effect Models
Version 0.6-6
Date 2018-09-03

Depends R (>= 3.1.2), multcomp (>= 1.3-6)
License GPL-2
URL https://github.com/jmpsteen/medflex
LazyData true
Imports boot (>= 1.3-8), car (>= 2.0-21), Matrix (>= 1.1-4), graphics (>= 3.1.2), sandwich (>= 2.3-2), stats (>= 3.1.2), utils (>= 3.1.2)
Suggests arm (>= 1.7-05), gam (>= 1.09.1), glmnet (>= 1.9-8), mice (>= 2.22), mitools (>= 2.3), rpart (>= 4.1-8), SuperLearner (>= 2.0-15), VGAM (>= 1.0-0)
RoxygenNote 6.1.0
ByteCompile yes
NeedsCompilation no
Author Johan Steen [aut, cre], Tom Loeys [aut], Beatrijs Moerkerke [aut], Stijn Vansteelandt [aut], Joris Meys [ctb] (technical support), Theis Lange [ctb] (valuable suggestions), Joscha Legewie [ctb], Paul Fink [ctb]
Maintainer Johan Steen <johan.steen@gmail.com>
Repository CRAN
Date/Publication 2018-09-03 07:20:07 UTC
**Description**

Expanded dataset including either ratio-of-mediator probability weights or imputed nested counterfactual outcomes.

**Value**

A data frame, resulting from applying `neWeight` or `neImpute` on an original dataset `data`. This data frame has `nRep * length(data)` rows, containing all original variables (except the original exposure variable) and two variables reflecting observed and hypothetical values of the exposure for each observation unit.

These auxiliary variables (x and x*) are named after the exposure variable and carry integers as suffixes. Suffixes 0 and 1 are used for variables whose corresponding parameters in the final natural effect model index natural direct and indirect effects, respectively.

This object also stores some additional attributes, which are used as input for `neModel`, such as

- `model` the fitted working model object
- `data` original dataset
- `call` the matched call
- `terms` the `neTerms` (internal class) object used
- `weights` ratio-of-mediator probability weights (only stored if object inherits from class `weightData`
Note

If the weighting-based approach (neWeight) is applied, the original outcome values are copied for the nested counterfactual outcomes and the object stores an additional attribute, "weights", containing a vector with ratio-of-mediator probability weights.

If the imputation-based approach (neImpute) is applied, the nested counterfactual outcomes are imputed by predictions from the imputation model.

In the former case, this object inherits from classes c("data.frame", "expData", "impData"), whereas in the latter case it inherits from classes c("data.frame", "expData", "weightData").

See Also

neImpute, neWeight

expData-methods  Methods for expanded datasets

Description

Regression weights, residuals and residual plots for expanded datasets.

Usage

## S3 method for class 'expData'
residuals(object, ...)

## S3 method for class 'expData'
residualPlot(object, ...)

## S3 method for class 'expData'
residualPlots(object, ...)

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

Arguments

object  an expanded dataset (of class "expData").
...
    additional arguments.

Details

weights extracts regression weights (to be used in the natural effect model) for each observation of an expanded dataset.

residuals extracts residuals from the working model which is stored as an attribute of the expanded dataset. These can be used to assess normality of the residuals of the mediator working model when using the weighting-based approach (see example).
residualPlot and residualPlots are convenience functions from the car package. These can be used to assess the adequacy of the working model.

See Also
expData, neWeight, residualPlot, residualPlots, residuals, weights

Examples

data(UPBdata)

weightData <- neWeight(negaff ~ att + gender + educ + age,
                        data = UPBdata, nRep = 2)

## extract regression weights for natural effect model
head(weights(weightData))

## assess normality
qqnorm(residuals(weightData))

## assess model adequacy
library(car)
residualPlots(weightData)

neImpute

Expand the dataset and impute nested counterfactual outcomes

Description
This function both expands the data along hypothetical exposure values and imputes nested counterfactual outcomes.

Usage
neImpute(object, ...)

Arguments
object an object used to select a method.
... additional arguments.

Details
Generic function that both expands the data along hypothetical exposure values (for each observation unit i) and imputes nested counterfactual outcomes in this expanded dataset in a single run. Imputed counterfactual outcomes

\[ \hat{E}(Y_i|X_i = x, M_i, C_i) \]

are predictions from the imputation model that can be specified either externally as a fitted model object (neimpute.default) or internally (neimpute.formula).
Value

A data frame of class c("data.frame", "expData", "impData"). See expData for its structure.

References


See Also

neImpute.default, neImpute.formula, neModel, expData

**Description**

This function both expands the data along hypothetical exposure values and imputes nested counterfactual outcomes.

**Usage**

```r
# Default S3 method:
neImpute(object, formula, data, nMed = 1, nRep = 5,
          xSampling = c("quantiles", "random"), xFit, percLim = c(0.05, 0.95),
          ...)```

**Arguments**

- **object**: fitted model object representing the imputation model.
- **formula**: a formula object providing a symbolic description of the imputation model. Redundant if already specified in call for fitted model specified in object (see details).
- **data**: data, as matrix or data frame, containing the exposure (and other relevant) variables. Redundant if already specified in call for fitted model specified in object (see details).
- **nMed**: number of mediators.
- **nRep**: number of replications or hypothetical values of the exposure to sample for each observation unit.
- **xSampling**: character string indicating how to sample from the conditional exposure distribution. Possible values are "quantiles" or "random" (see details).
**Details**

Imputed counterfactual outcomes are predictions from the imputation model that needs to be specified as a fitted object in the `object` argument.

If the model-fitting function used to fit the imputation model does not require specification of a formula or data argument (when using e.g. `SuperLearner`), these need to be specified explicitly in order to enable `neImpute.default` to extract pointers to variable types relevant for mediation analysis.

Whether a formula is specified externally (in the call for the fitted imputation model object which is specified in `object`) or internally (via the `formula` argument), it always needs to be of the form $Y \sim X + M1 + M2 + M3 + C1 + C2$, with the same outcome as in the final natural effect model and with predictor variables entered in the following prespecified order:

1. exposure $X$: The first predictor is coded as exposure or treatment.
2. mediator(s) $M$: The second predictor is coded as mediator. In case of multiple mediators ($nMed > 1$), then predictors $2:(nMed + 1)$ are coded as mediators.
3. baseline covariates $C$: All remaining predictor variables are automatically coded as baseline covariates.

It is important to adhere to this prespecified order to enable `neImpute` to create valid pointers to these different types of predictor variables. This requirement extends to the use of operators different than the + operator, such as the : and * operators (when e.g. adding interaction terms).

For instance, the formula specifications $Y \sim X * M + C1 + C2$, $Y \sim X + M + X:M + C1 + C2$ and $Y \sim X + X:M + M + C1 + C2$ will create identical pointers to the different types of variables, as the order of the unique predictor variables is identical in all three specifications.

Furthermore, categorical exposures that are not coded as factors in the original dataset, should be specified as factors in the formula, using the `factor` function, e.g. $Y \sim factor(X) + M + C1 + C2$. Quadratic or higher-order polynomial terms can be included as well, by making use of the `I` function or by using the `poly` function. For instance, $Y \sim X + I(X^2) + M + C1 + C2$ and $Y \sim poly(X, 2, raw = TRUE) + M + C1 + C2$ are equivalent and result in identical pointers to the different types of variables.

The command `terms(object, "vartype")` (with object replaced by the name of the resulting expanded dataset) can be used to check whether valid pointers have been created.

If multiple mediators are specified ($nMed > 1$), the natural indirect effect parameter in the natural effect model captures the joint mediated effect. That is, the effect of the exposure on the outcome via these mediators considered jointly. The remaining effect of the exposure on the outcome (not mediated through the specified mediators) is then captured by the natural indirect effect parameter.

In contrast to imputation models with categorical exposures, additional arguments need to be specified if the exposure is continuous. All of these additional arguments are related to the sampling procedure for the exposure.
Whereas the number of replications \( n_{\text{Rep}} \) for categorical variables equals the number of levels for the exposure coded as a factor (i.e., the number of hypothetical exposure values), the number of desired replications needs to be specified explicitly for continuous exposures. Its default is 5.

If \( x_{\text{Fit}} \) is left unspecified, the hypothetical exposure levels are automatically sampled from a linear model for the exposure, conditional on a linear combination of all covariates. If one wishes to use another model for the exposure, this default model specification can be overruled by referring to a fitted model object in the \( x_{\text{Fit}} \) argument. Misspecification of this sampling model does not induce bias in the estimated coefficients and standard errors of the natural effect model.

The \( x_{\text{Sampling}} \) argument allows to specify how the hypothetical exposure levels should be sampled from the conditional exposure distribution (which is either entered explicitly using the \( x_{\text{Fit}} \) argument or fitted automatically as described in the previous paragraph). The “random” option randomly samples \( n_{\text{Rep}} \) draws from the exposure distribution, whereas the “quantiles” option (default) samples \( n_{\text{Rep}} \) quantiles at equal-sized probability intervals. Only the latter hence yields fixed exposure levels given \( n_{\text{Rep}} \) and \( x_{\text{Fit}} \).

In order to guarantee that the entire support of the distribution is being sampled (which might be a concern if \( n_{\text{Rep}} \) is chosen to be small), the default lower and upper sampled quantiles are the 5th and 95th percentiles. The intermittent quantiles correspond to equal-sized probability intervals. So, for instance, if \( n_{\text{Rep}} = 4 \), then the sampled quantiles will correspond to probabilities 0.05, 0.35, 0.65 and 0.95. These default ‘outer’ quantiles can be changed by specifying the \( \text{perclim} \) argument accordingly. By specifying \( \text{perclim} = \text{NULL} \), the standard quantiles will be sampled (e.g., 0.2, 0.4, 0.6 and 0.8 if \( n_{\text{Rep}} = 4 \)).

**Value**

A data frame of class \("\text{data.frame}\", \"expData\", \"impData\") See \expData \ for its structure.

**See Also**

\( \text{neImpute}, \text{neImpute.formula}, \text{neModel}, \expData \)

**Examples**

data(UPBdata)

### example using glm imputation model with binary exposure
fit(glm <- glm(UPB ~ factor(attbin) + negaff + gender + educ + age,

family = binomial, data = UPBdata)
impData <- neImpute(fitglm)
head(impData)

### example using glm imputation model with continuous exposure
fit(glm <- glm(UPB ~ att + negaff + gender + educ + age,

family = binomial, data = UPBdata)
impData <- neImpute(fitglm, nRep = 2)
head(impData)

### example using vglm (yielding identical results as with glm)
library(VGAM)
fit.vglm <- vglm(UPB ~ att + negaff + gender + educ + age,
family = binomialff, data = UPBdata)
impData2 <- neImpute(fit.vglm, nRep = 2)
head(impData2)

## Not run:
## example using SuperLearner
library(matrix)
library(SuperLearner)
SL.library <- c("SL.glm", "SL.glm.interaction", "SL.rpart",
    "SL.step", "SL.stepAIC", "SL.step.interaction",
    "SL.bayesglm", "SL.glmmnet")
pred <- c("att", "negaff", "gender", "educ", "age")
fit.SL <- SuperLearner(Y = UPBdata$UPB, X = subset(UPBdata, select = pred),
    SL.library = SL.library, family = binomial())
impSL <- neImpute(fit.SL,
    formula = UPB ~ att + negaff + gender + educ + age,
    data = UPBdata)
head(impSL)

## End(Not run)

neImpute.formula

Expand the dataset and impute nested counterfactual outcomes

Description

This function both expands the data along hypothetical exposure values and imputes nested counterfactual outcomes.

Usage

## S3 method for class 'formula'
neImpute(object, family, data, FUN = glm, nMed = 1,
    nRep = 5, xSampling = c("quantiles", "random"), xFit,
    perclim = c(0.05, 0.95), ...)

Arguments

object a formula object providing a symbolic description of the imputation model (see details).

family a description of the error distribution and link function to be used in the model. Consult the help files of the model-fitting function specified in FUN for more details on appropriate argument specification.

data data, as matrix or data frame, containing the exposure (and other relevant) variables. Redundant if already specified in call for fitted model specified in object (see details).
fun
function used to fit model specified in formula.
nMed	number of mediators.
nRep
number of replications or hypothetical values of the exposure to sample for each observation unit.
xSampling
character string indicating how to sample from the conditional exposure distribution. Possible values are "quantiles" or "random" (see details).
xFit
an optional fitted object (preferably glm) for the conditional exposure distribution (see details).
perclim
a numerical vector of the form c(lower, upper) indicating the extreme percentiles to sample when using "quantiles" as sampling method to sample from the conditional exposure distribution (see details).
... additionally arguments (passed to FUN).

Details
Imputed counterfactual outcomes are predictions from the imputation model that is fitted internally by extracting information from the arguments object, family, data, FUN and . . . .
For imputation model specification via the object argument, use a formula of the form
Y ~ X + M1 + M2 + M3 + C1 + C2,
with the same outcome as in the final natural effect model and with predictor variables entered in the following prespecified order:

1. exposure X: The first predictor is coded as exposure or treatment.
2. mediator(s) M: The second predictor is coded as mediator. In case of multiple mediators (nMed > 1), then predictors 2:(nMed + 1) are coded as mediators.
3. baseline covariates C: All remaining predictor variables are automatically coded as baseline covariates.

It is important to adhere to this prespecified order to enable neImpute to create valid pointers to these different types of predictor variables. This requirement extends to the use of operators different than the + operator, such as the : and * operators (when e.g. adding interaction terms).
For instance, the formula specifications Y ~ X * M + C1 + C2, Y ~ X + X:M + M + C1 + C2 and Y ~ X + X:M + M + C1 + C2 will create identical pointers to the different types of variables, as the order of the unique predictor variables is identical in all three specifications.

Furthermore, categorical exposures that are not coded as factors in the original dataset, should be specified as factors in the formula, using the factor function, e.g. Y ~ factor(X) + M + C1 + C2. Quadratic or higher-order polynomial terms can be included as well, by making use of the I function or by using the poly function. For instance, Y ~ X + I(X^2) + M + C1 + C2 and Y ~ poly(X, 2, raw = TRUE) + M + C1 + C2 are equivalent and result in identical pointers to the different types of variables.

The command terms(object, "vartype") (with object replaced by the name of the resulting expanded dataset) can be used to check whether valid pointers have been created.

If multiple mediators are specified (nMed > 1), the natural indirect effect parameter in the natural effect model captures the joint mediated effect. That is, the effect of the exposure on the outcome
via these mediators considered jointly. The remaining effect of the exposure on the outcome (not mediated through the specified mediators) is then captured by the natural indirect effect parameter.

The type of imputation model can be defined by specifying an appropriate model-fitting function via the FUN argument (its default is \texttt{glm}). This method can only be used with model-fitting functions that require a \texttt{formula} argument (so not when using e.g. \texttt{Super Learner}).

In contrast to imputation models with categorical exposures, additional arguments need to be specified if the exposure is continuous. All of these additional arguments are related to the sampling procedure for the exposure.

Whereas the number of replications \texttt{nrep} for categorical variables equals the number of levels for the exposure coded as a factor (i.e. the number of hypothetical exposure values), the number of desired replications needs to be specified explicitly for continuous exposures. Its default is 5.

If \texttt{xFit} is left unspecified, the hypothetical exposure levels are automatically sampled from a linear model for the exposure, conditional on a linear combination of all covariates. If one wishes to use another model for the exposure, this default model specification can be overruled by referring to a fitted model object in the \texttt{xFit} argument. Misspecification of this sampling model does not induce bias in the estimated coefficients and standard errors of the natural effect model.

The \texttt{xSampling} argument allows to specify how the hypothetical exposure levels should be sampled from the conditional exposure distribution (which is either entered explicitly using the \texttt{xFit} argument or fitted automatically as described in the previous paragraph). The "random" option randomly samples \texttt{nrep} draws from the exposure distribution, whereas the "quantiles" option (default) samples \texttt{nRep} quantiles at equal-sized probability intervals. Only the latter hence yields fixed exposure levels given \texttt{nRep} and \texttt{xFit}.

In order to guarantee that the entire support of the distribution is being sampled (which might be a concern if \texttt{nRep} is chosen to be small), the default lower and upper sampled quantiles are the 5th and 95th percentiles. The intermittent quantiles correspond to equal-sized probability intervals. So, for instance, if \texttt{nRep} = 4, then the sampled quantiles will correspond to probabilities 0.05, 0.35, 0.65 and 0.95. These default 'outer' quantiles can be changed by specifying the \texttt{perclim} argument accordingly. By specifying \texttt{perclim = NULL}, the standard quantiles will be sampled (e.g., 0.2, 0.4, 0.6 and 0.8 if \texttt{nRep} = 4).

**Value**

A data frame of class \texttt{cHBdataNframe, expdata, impdata}). See \texttt{expData} for its structure.

**See Also**

\texttt{neImpute, neImpute.default, neModel, expData}

**Examples**

```r
data(UPBdata)

## example using glm imputation model with binary exposure
impData <- neImpute(UPB ~ factor(attbin) + negaff + gender + educ + age,
                     family = binomial, data = UPBdata)
head(impData)
```

A data frame of class \texttt{cHBdataNframe, expData, impData}). See \texttt{expData} for its structure.
## Example using glm imputation model with continuous exposure

```r
impData <- neImpute(UPB ~ att + negaff + gender + educ + age,
                     family = binomial, data = UPBdata, nRep = 2)
head(impData)
```

## Example using vglm (yielding identical results as with glm)

```r
library(VGAM)
impData2 <- neImpute(UPB ~ att + negaff + gender + educ + age,
                      family = binomialff, data = UPBdata,
                      nRep = 2, FUN = vglm)
head(impData2)
```

---

### neLht

**Linear hypotheses for natural effect models**

#### Description

neLht allows to calculate linear combinations of natural effect model parameter estimates. neEffdecomp automatically extracts relevant causal parameter estimates from a natural effect model.

#### Usage

```r
neEffdecomp(model, xRef, covlev, ...)
```

**## S3 method for class 'neModel'**

```r
neEffdecomp(model, xRef, covlev, ...)
```

```r
neLht(model, ...)
```

**## S3 method for class 'neModel'**

```r
neLht(model, ...)
```

#### Arguments

- `model`: a fitted natural effect model object.
- `xRef`: a vector including reference levels for the exposure, \( x^* \) and \( x \), at which natural effect components need to be evaluated (see details).
- `covlev`: a vector including covariate levels at which natural effect components need to be evaluated (see details).
- `...`: additional arguments (passed to `glht`).

#### Details

neLht is a wrapper of `glht` and offers the same functionality (see ‘Details’ section of `glht` for details on argument specification). It returns objects that inherit from the class "neLht" in order to make output of their corresponding methods (see `neLht-methods`) more compatible for natural effect models containing bootstrap variance-covariance matrices and standard errors.
neEffdecomp is a convenience function that automatically extracts causal parameter estimates from a natural effect model and derives natural effect components. That is, natural direct, natural indirect and total causal effect estimates are returned if no exposure-mediator interaction is modelled (i.e. two-way decomposition). If mediated interaction is allowed for in the natural effect model, there are two ways of decomposing the total effect into (natural) direct and indirect effects components: either as the sum of the pure direct and the total indirect effect or as the sum of the pure indirect and the total direct effect (i.e. three-way decomposition). In total, five causal effect estimates are returned in this case.

For continuous exposures, default exposure levels at which natural effect components are evaluated are $x^* = 0$ and $x = 1$. For multicategorical exposures, default levels are the reference level of the factor that encodes the exposure variable and the level corresponding to its first dummy variable for $x^*$ and $x$, respectively. If one wishes to evaluate natural effect components at different reference levels (e.g. if the natural effect model includes mediated interaction, quadratic or higher-order polynomial terms for the exposure; see examples), these can be specified as a vector of the form $c(x^*, x)$ via the xRef argument.

If applicable, covariate levels at which natural effect components are evaluated can also be specified. This is particularly useful for natural effect models encoding effect modification by baseline covariates (e.g. moderated mediation). By default, these levels are set to 0 for continuous covariates and to the reference level for categorical covariates coded as factors. Different covariate levels can be specified via the covLev argument, which requires a vector including valid levels for covariates that are specified in the natural effect model (or a subset of covariates that are specified as modifiers of either the natural direct or indirect effect or both). Levels need to be preceded by the name of the corresponding covariate, e.g., covLev = c(gender = "M", age = 30). Covariates for which the levels are left unspecified are set to their default levels (see examples). The print and summary functions for neEffdecomp objects return the covariate levels at which natural effect components are evaluated. No specific levels are returned for covariates that are not specified as modifier since effect decomposition is independent of the level of these covariates (see examples).

Value

An object of class c("neLht", "glht") (see glht). If the bootstrap is used for obtaining standard errors when fitting the neModel object, the returned object additionally inherits from class "neLhtBoot". neEffdecomp returns an object that additionally inherits from class "neEffdecomp".

See neLht-methods for methods for neLht objects (and glht-methods for additional methods for glht objects).

Note

neEffdecomp is internally called by plot.neModel to create confidence interval plots for neModel objects.

See Also

plot.neLht, neLht-methods, glht, glht-methods, neModel, plot.neModel, summary

Examples

data(UPBdata)
impData <- neImpute(UPB ~ att * negaff + gender + educ + age,
  family = binomial, data = UPBdata)
neMod <- neModel(UPB ~ att0 * att1 + gender + educ + age,
  family = binomial, expData = impData, se = "robust")

lht <- neLht(neMod, linfct = c("att0 = 0", "att0 + att0:att1 = 0",
  "att1 = 0", "att1 + att0:att1 = 0",
  "att0 + att1 + att0:att1 = 0"))

summary(lht)

## or obtain directly via neEffdecomp
eff <- neEffdecomp(neMod)
summary(eff)

## changing reference levels for multicategorical exposures
UPBData$attcat <- factor(cut(UPBData$att, 3), labels = c("L", "M", "H"))
impData <- neImpute(UPB ~ attcat * negaff + gender + educ + age,
  family = binomial, data = UPBData)
neMod <- neModel(UPB ~ attcat0 * attcat1 + gender + educ + age,
  family = binomial, expData = impData, se = "robust")

neEffdecomp(neMod)
neEffdecomp(neMod, xRef = c("L", "H"))
neEffdecomp(neMod, xRef = c("M", "H"))

## changing reference levels for continuous exposures
impData <- neImpute(UPB ~ (att + I(att^2)) * negaff + gender + educ + age,
  family = binomial, data = UPBData)
neMod <- neModel(UPB ~ (att0 + I(att0^2)) * (att1 + I(att1^2)) + gender + educ + age,
  family = binomial, expData = impData, se = "robust")

neEffdecomp(neMod)
neEffdecomp(neMod, xRef = c(-1, 0))

## changing covariate levels when allowing for modification
## of the indirect effect by baseline covariates
impData <- neImpute(UPB ~ (att + negaff + gender + educ + age)^2,
  family = binomial, data = UPBData)
neMod <- neModel(UPB ~ att0 * att1 + gender + educ + age + att1:gender + att1:age,
  family = binomial, expData = impData, se = "robust")

neEffdecomp(neMod)
neEffdecomp(neMod, covLev = c(gender = "F", age = 0)) # default covariate levels
neEffdecomp(neMod, covLev = c(gender = "M", age = 40))
neEffdecomp(neMod, covLev = c(gender = "M", age = 40, educ = "L"))
neEffdecomp(neMod, covLev = c(gender = "M", age = 40, educ = "M"))
neEffdecomp(neMod, covLev = c(gender = "M", age = 40, educ = "H"))
# effect decomposition is independent of education level
neEffdecomp(neMod, covLev = c(gender = "M"))
# age is set to its default level when left unspecified
Methods for linear hypotheses in natural effect models

Description

Obtain confidence intervals and statistical tests for linear hypotheses in natural effect models.

Usage

```r
## S3 method for class 'neLhtBoot'
confint(object, parm, level = 0.95, type = "norm",
        ...)  
## S3 method for class 'neLht'
confint(object, parm, level = 0.95,
        calpha = univariate_calpha(), ...)
## S3 method for class 'neLht'
summary(object, test = univariate(), ...)
```

Arguments

- `object`: an object of class `neLht`.
- `parm`: a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
- `level`: the confidence level required.
- `type`: the type of bootstrap intervals required. The default "norm" returns normal approximation bootstrap confidence intervals. Currently, "norm", "basic", "perc" and "bca" are supported (see `boot.ci`).
- `...`: additional arguments.
- `calpha`: a function computing the critical value. The default `univariate_calpha()` returns unadjusted confidence intervals, whereas `adjusted_calpha()` returns adjusted confidence intervals.
- `test`: a function for computing p-values. The default `univariate()` does not apply a multiple testing correction. The function `adjusted()` allows to correct for multiple testing (see `summary.glht` and `adjusted`) and `chisquare()` allows to test global linear hypotheses.

Details

`confint` yields bootstrap confidence intervals or confidence intervals based on the sandwich estimator (depending on the type of standard errors requested when fitting the `neModel` object). Bootstrap confidence intervals are internally called via the `boot.ci` function from the `boot` package. Confidence intervals based on the sandwich estimator are internally called via the corresponding
confint.glht function from the multcomp package. The default confidence level specified in level (which corresponds to the conf argument in boot.ci) is 0.95 and the default type of bootstrap confidence interval, "norm", is based on the normal approximation. Bias-corrected and accelerated ("bca") bootstrap confidence intervals require a sufficiently large number of bootstrap replicates (for more details see boot.ci).

A summary table with large sample tests, similar to that for glht, can be obtained using summary. In contrast to summary.glht, which by default returns p-values that are adjusted for multiple testing, the summary function returns unadjusted p-values. Adjusted p-values can also be obtained by specifying the test argument (see adjusted for more details).

Global Wald tests considering all linear hypotheses simultaneously (i.e. testing the global null hypothesis) can be requested by specifying test = chisqtest().

See glht-methods for additional methods for glht objects.

Note

For the bootstrap, z-values in the summary table are simply calculated by dividing the parameter estimate by its corresponding bootstrap standard error. Corresponding p-values in the summary table are only indicative, since the null distribution for each statistic is assumed to be approximately standard normal. Therefore, whenever possible, it is recommended to focus mainly on bootstrap confidence intervals for inference, rather than the provided p-values.

See Also

neLht, plot.neLht, glht, glht-methods

Examples

data(UPBdata)

impData <- neImpute(UPB ~ att * negaff + gender + educ + age,
family = binomial, data = UPBdata)
neMod <- neModel(UPB ~ att0 + att1 + gender + educ + age,
family = binomial, expData = impData, se = "robust")

lht <- neLht(neMod, linfct = c("att0 = 0", "att0 + att0:att1 = 0",
"att1 = 0", "att1 + att0:att1 = 0",
"att0 + att1 + att0:att1 = 0")

## obtain confidence intervals
confint(lht)
confint(lht, parm = c("att0", "att0 + att0:att1"))
confint(lht, parm = 1:2, level = 0.90)

## summary table
summary(lht)

## summary table with omnibus Chisquare test
summary(lht, test = Chisqtest())
neModel

Fit a natural effect model

Description

neModel is used to fit a natural effect model on the expanded dataset.

Usage

neModel(formula, family = gaussian, expData, xFit, se = c("bootstrap", "robust"), nBoot = 1000, parallel = c("no", "multicore", "snow"), ncpus = getOption("boot.ncpus", 1L), progress = TRUE, ...)

Arguments

formula a formula object providing a symbolic description of the natural effect model.

family a description of the error distribution and link function to be used in the model. For glm this can be a character string naming a family function, a family function or the result of a call to a family function. For glm.fit only the third option is supported. (See family for details of family functions.)

expData the expanded dataset (of class "expData").

xFit fitted model object representing a model for the exposure (used for inverse treatment (exposure) probability weighting).

se character string indicating the type of standard errors to be calculated. The default type is based on the bootstrap (see details).

nBoot number of bootstrap replicates (see argument of boot).

parallel (only for bootstrap) The type of parallel operation to be used (if any). If missing, the default is taken from the option "boot.parallel" (and if that is not set, "no").

ncpus (only for bootstrap) integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs (see details).

progress (only for bootstrap) logical value indicating whether or not a progress bar should be displayed. Progress bars are automatically disabled for multicore processing.

... additional arguments (passed to glm).

Details

This function is a wrapper for glm, providing unbiased bootstrap (se = "bootstrap", the default) or robust (se = "robust") standard errors for the parameter estimates (see below for more details).

The formula argument requires to be specified in function of the variables from the expanded dataset (specified in expData) whose corresponding parameters index the direct and indirect effect. Stratum-specific natural effects can be estimated by additionally modeling the relation between the outcome and baseline covariates. If the set of baseline covariates adjusted for in the formula
argument is not sufficient to control for confounding (e.g. when fitting a population-average natural effect model), an adequate model for the exposure (conditioning on a sufficient set of baseline covariates) should be specified in the `xfit` argument. In this case, such a model for the exposure distribution is needed to weight by the reciprocal of the probability (density) of the exposure (i.e. inverse probability weighting) in order to adjust for confounding. Just as for ratio-of-mediator probability weighting (see paragraph below), this kind of weighting is done internally.

Quadratic or higher-order polynomial terms can be included in the `formula` by making use of the `I` function or by using the `poly` function. However, we do not recommend the use of orthogonal polynomials (i.e. using the default argument specification `raw = FALSE` in `poly`), as these are not compatible with the `neEffdecomp` function.

In contrast to `glm`, the `expData` argument (rather than `data` argument) requires specification of a data frame that inherits from class "expData", which contains additional information about e.g. the fitted working model, the variable types or terms of this working model and possibly ratio-of-mediator probability weights. The latter are automatically extracted from the `expData` object and weighting is done internally.

As the default `glm` standard errors fail to reflect the uncertainty inherent to the working model(s) (i.e. either a model for the mediator or an imputation model for the outcome and possibly a model for the exposure), bootstrap standard errors (using the `boot` function from the `boot` package) or robust standard errors are calculated. The default type of standard errors is bootstrap standard errors. Robust standard errors (based on the sandwich estimator) can be requested (to be calculated) instead by specifying `se = "robust"`.

Value

An object of class "neModel" (which additionally inherits from class "neModelBoot" if the bootstrap is used) consisting of a list of 3 objects:

- `nemodelfit` the fitted natural model object (of class "glm") with downwardly biased standard errors
- `bootRes`, `vcov` the bootstrap results (of class "boot"; if `se = "bootstrap"`) or the robust variance-covariance matrix (if `se = "robust"`)
- `terms` the `neTerms` (internal class) object used. This object is equivalent to the `terms` object returned by the `glm` function, but has an additional "vartype" attribute, a list including pointers to the names of the outcome variable (Y), exposure (X), mediator (M), covariates (C) and auxiliary hypothetical variables x and x* (Xexp).

See `neModel-methods` for methods for `neModel` objects.

Bootstrap standard errors

The bootstrap procedure entails refitting all working models on each bootstrap sample, reconstructing the expanded dataset and subsequently refitting the specified natural effect model on this dataset. In order to obtain stable standard errors, the number of bootstrap samples (specified via the `nboot` argument) should be chosen relatively high (default is 1000).

To speed up the bootstrap procedure, parallel processing can be used by specifying the desired type of parallel operation via the `parallel` argument (for more details, see `boot`). The number of parallel processes (ncpus) is suggested to be specified explicitly (its default is 1, unless the global option `options("boot.cpus")` is specified). The function `detectCores` from the `parallel` package can
be helpful at determining the number of available cores (although this may not always correspond to the number of allowed cores).

Robust standard errors

Robust variance-covariance matrices for the model parameters, based on the sandwich estimator, are calculated using core functions from the `sandwich` package. Additional details and derivations for the sandwich estimator for natural effect models can be found in the corresponding vignette that can be obtained by the command `vignette("sandwich", package = "medflex").`

Note

It is important to note that the original mediator(s) should not be specified in the `formula` argument, as the natural indirect effect in natural effect models should be captured solely by parameter(s) corresponding to the auxiliary hypothetical variable $x^*$ in the expanded dataset (see `expData`).

References


See Also

`neModel-methods`, `plot.neModel`, `neImpute`, `neWeight`, `neLht`, `neEffdecomp`

Examples

data(UPBdata)

`#` weighting-based approach `##`
`#` weighting-based approach `##`

`weightData <- neWeight(negaff ~ att + gender + educ + age,`
`  data = UPBdata)`

`## stratum-specific natural effects`
`# bootstrap SE`
`## Not run:`
`weightFit1b <- neModel(UPB ~ att0 * att1 + gender + educ + age,`
`  family = binomial, expData = weightData)`
`summary(weightFit1b)`

`## End(Not run)`
`## robust SE`
`weightFit1r <- neModel(UPB ~ att0 * att1 + gender + educ + age,`
family = binomial, expData = weightData, se = "robust")

summary(weightFit1r)

## population-average natural effects
expFit <- glm(att - gender + educ + age, data = UPBdata)
## bootstrap SE
## Not run:
weightFit2b <- neModel(UPB ~ att0 * att1, family = binomial,
                     expData = weightData, xFit = expFit)
summary(weightFit2b)

## End(Not run)
## robust SE
weightFit2r <- neModel(UPB ~ att0 * att1, family = binomial,
                     expData = weightData, xFit = expFit, se = "robust")
summary(weightFit2r)

#################################################################
## imputation-based approach ##
#################################################################
impData <- neImpute(UPB ~ att + negaff + gender + educ + age,
                     family = binomial, data = UPBdata)

## stratum-specific natural effects
## bootstrap SE
## Not run:
impFit1b <- neModel(UPB ~ att0 * att1 + gender + educ + age,
                     family = binomial, expData = impData)
summary(impFit1b)

## End(Not run)
## robust SE
impFit1r <- neModel(UPB ~ att0 * att1 + gender + educ + age,
                     family = binomial, expData = impData, se = "robust")
summary(impFit1r)

## population-average natural effects
## bootstrap SE
## Not run:
impFit2b <- neModel(UPB ~ att0 * att1, family = binomial,
                     expData = impData, xFit = expFit)
summary(impFit2b)

## End(Not run)
## robust SE
impFit2r <- neModel(UPB ~ att0 * att1, family = binomial,
                     expData = impData, xFit = expFit, se = "robust")
summary(impFit2r)
Methods for natural effect models

Description
Extractor functions, confidence intervals, residual plots and statistical tests for natural effect models.

Usage

## S3 method for class 'neModel'
coef(object, ...)

## S3 method for class 'neModelBoot'
confint(object, parm, level = 0.95,
        type = "norm", ...)

## S3 method for class 'neModel'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'neModel'
residualPlot(object, ...)

## S3 method for class 'neModel'
residualPlots(object, ...)

## S3 method for class 'neModel'
summary(object, ...)

## S3 method for class 'neModel'
vcov(object, ...)

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

Arguments

- **object**: a fitted natural effect model object.
- **...**: additional arguments.
- **parm**: a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
- **level**: the confidence level required.
- **type**: the type of bootstrap intervals required. The default "norm" returns normal approximation bootstrap confidence intervals. Currently, "norm", "basic", "perc" and "bca" are supported (see boot.ci).
Details

confint yields bootstrap confidence intervals or confidence intervals based on the sandwich estimator (depending on the type of standard errors requested when fitting the `neModel` object). Bootstrap confidence intervals are internally called via the `boot.ci` function from the `boot` package. Confidence intervals based on the sandwich estimator are internally called via `confint.default`. The default confidence level specified in `level` (which corresponds to the `conf` argument in `boot.ci`) is 0.95 and the default type of bootstrap confidence interval, “norm”, is based on the normal approximation. Bias-corrected and accelerated (“bca”) bootstrap confidence intervals require a sufficiently large number of bootstrap replicates (for more details see `boot.ci`).

A summary table with large sample tests, similar to that for `glm` output, can be obtained using `summary`.

`vcov` returns either the bootstrap variance-covariance matrix (calculated from the bootstrap samples stored in `object$bootRes`; see `neModel`) or the robust variance-covariance matrix (which is a diagonal block matrix of the original sandwich covariance matrix).

`weights` returns a vector containing the regression weights used to fit the natural effect model. These weights can be based on

1. ratio-of-mediator probability (density) weights (only if the weighting-based approach is used)
2. inverse probability of treatment (exposure) weights (only if `xFit` was specified in `neModel`)

`residualPlot` and `residualPlots` are convenience functions from the `car` package. These can be used to assess model adequacy.

Note

For the bootstrap, z-values in the summary table are calculated by dividing the parameter estimate by its corresponding bootstrap standard error. Corresponding p-values in the summary table are indicative, since the null distribution for each statistic is assumed to be approximately standard normal. Therefore, whenever possible, it is recommended to focus mainly on bootstrap confidence intervals for inference, rather than the provided p-values.

See Also

`neModel`, `plot.neModel`, `residualPlot`, `residualPlots`, `weights`

Examples

data(UPBdata)

weightData <- neWeight(negaff ~ att + educ + gender + age,
                        data = UPBdata)
neMod <- neModel(UPB ~ att0 * att1 + educ + gender + age,
                 family = binomial, expData = weightData, se = "robust")

## extract coefficients
coeff(neMod)

## extract variance-covariance matrix
neWeight

Expand the dataset and calculate ratio-of-mediator probability weights

Description

This function both expands the data along hypothetical exposure values and calculates ratio-of-mediator probability weights.

Usage

neWeight(object, ...)

Arguments

object an object used to select a method.
... additional arguments.

Details

Generic function that both expands the data along hypothetical exposure values and calculates ratio-of-mediator probability weights

\[
\frac{P(M_i | X_i = x^*, C_i)}{P(M_i | X_i = x, C_i)}
\]

for each observation unit \(i\) in this expanded dataset in a single run. These weights are ratios of probabilities or probability densities from the mediator model distribution, which can be specified either externally as a fitted model object \(\text{neWeight.default}\) or internally \(\text{neWeight.formula}\).
Value

A data frame of class c("data.frame", "expData", "weightData"). See expData for its structure.

References


See Also

newweight.default, newweight.formula, expData

Description

This function both expands the data along hypothetical exposure values and calculates ratio-of-mediator probability weights.

Usage

## Default S3 method:
newWeight(object, formula, data, nRep = U,
xsampling = c("quantiles", "random"), xFit, perclim = c(0.05, 0.95),
...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>fitted model object representing the mediator model.</td>
</tr>
<tr>
<td>formula</td>
<td>a formula object providing a symbolic description of the mediator model.</td>
</tr>
<tr>
<td>data</td>
<td>data, as matrix or data frame, containing the exposure (and other relevant) variables. Redundant if already specified in call for fitted model specified in object (see details).</td>
</tr>
<tr>
<td>nRep</td>
<td>number of replications or hypothetical values of the exposure to sample for each observation unit.</td>
</tr>
<tr>
<td>xsampling</td>
<td>character string indicating how to sample from the conditional exposure distribution. Possible values are &quot;quantiles&quot; or &quot;random&quot; (see details).</td>
</tr>
</tbody>
</table>
xFit an optional fitted object (preferably glm) for the conditional exposure distribution (see details).

derclim a numerical vector of the form c(lower, upper) indicating the extreme percentiles to sample when using "quantiles" as sampling method to sample from the conditional exposure distribution (see details).

... additional arguments.

Details

The calculated weights are ratios of fitted probabilities or probability densities from the distribution of the mediator model. This model needs to be specified as a fitted object in the object argument.

If the model-fitting function used to fit the mediator model does not require specification of a formula or data argument, these need to be specified explicitly in order to enable newWeight.default to extract pointers to variable types relevant for mediation analysis.

Whether a formula is specified externally (in the call for the fitted mediator model object which is specified in object) or internally (via the formula argument), it always needs to be of the form \( M \sim X + C1 + C2 \), with predictor variables entered in the following prespecified order:

1. exposure \( X \): The first predictor is coded as exposure or treatment.
2. baseline covariates \( C \): All remaining predictor variables are automatically coded as baseline covariates.

It is important to adhere to this prespecified order to enable newWeight to create valid pointers to these different types of predictor variables. This requirement extends to the use of operators different than the + operator, such as the : and * operators (when e.g. adding interaction terms). For instance, the formula specifications \( M \sim X \times C1 + C2 \), \( M \sim X + C1 + X:C1 + C2 \) and \( Y \sim X + X:C1 + C1 + C2 \) will create identical pointers to the different types of variables, as the order of the unique predictor variables is identical in all three specifications.

Furthermore, categorical exposures that are not coded as factors in the original dataset, should be specified as factors in the formula, using the factor function, e.g. \( M \sim \text{factor}(X) + C1 + C2 \). Quadratic or higher-order polynomial terms can be included as well, by making use of the I function or by using the poly function. For instance, \( M \sim X + I(X^2) + C1 + C2 \) and \( M \sim \text{poly}(X, 2, \text{raw} = \text{TRUE}) + C1 + C2 \) are equivalent and result in identical pointers to the different types of variables.

The command terms(object, "vartype") (with object replaced by the name of the resulting expanded dataset) can be used to check whether valid pointers have been created.

In contrast to imputation models with categorical exposures, additional arguments need to be specified if the exposure is continuous. All of these additional arguments are related to the sampling procedure for the exposure.

Whereas the number of replications nRep for categorical variables equals the number of levels for the exposure coded as a factor (i.e. the number of hypothetical exposure values), the number of desired replications needs to be specified explicitly for continuous exposures. Its default is 5.

If xFit is left unspecified, the hypothetical exposure levels are automatically sampled from a linear model for the exposure, conditional on a linear combination of all covariates. If one wishes to use another model for the exposure, this default model specification can be overruled by referring to a fitted model object in the xFit argument. Misspecification of this sampling model does not induce bias in the estimated coefficients and standard errors of the natural effect model.
The `xSampling` argument allows to specify how the hypothetical exposure levels should be sampled from the conditional exposure distribution (which is either entered explicitly using the `xFit` argument or fitted automatically as described in the previous paragraph). The "random" option randomly samples `nRep` draws from the exposure distribution, whereas the "quantiles" option (default) samples `nRep` quantiles at equal-sized probability intervals. Only the latter hence yields fixed exposure levels given `nRep` and `xFit`.

In order to guarantee that the entire support of the distribution is being sampled (which might be a concern if `nRep` is chosen to be small), the default lower and upper sampled quantiles are the 5th and 95th percentiles. The intermittent quantiles correspond to equal-sized probability intervals. So, for instance, if `nRep = 4`, then the sampled quantiles will correspond to probabilities 0.05, 0.35, 0.65 and 0.95. These default 'outer' quantiles can be changed by specifying the `perclim` argument accordingly. By specifying `perclim = NULL`, the standard quantiles will be sampled (e.g., 0.2, 0.4, 0.6 and 0.8 if `nRep = 4`).

Value

A data frame of class `c("data.frame", "expData", "weightData")`. See `expData` for its structure.

See Also

`newWeight`, `newWeight.formula`, `expData`

Examples

data(UPBdata)

## example using glm
fit.glm <- glm(negaff ~ att + gender + educ + age, data = UPBdata)
weightData <- newWeight(fit.glm, nRep = 2)

---

**neWeight.formula**

Expand the dataset and calculate ratio-of-mediator probability weights

Description

This function both expands the data along hypothetical exposure values and calculates ratio-of-mediator probability weights.

Usage

```r
## S3 method for class 'formula'
neWeight(object, family, data, FUN = glm, nRep = 5,
  xSampling = c("quantiles", "random"), xFit, perclim = c(0.05, 0.95),
  ...)  
```
Arguments

object
  a formula object providing a symbolic description of the mediator model (see details).

family
  a description of the error distribution and link function to be used in the model. Consult the help files of the model-fitting function specified in FUN for more details on appropriate argument specification.

data
  data, as matrix or data frame, containing the exposure (and other relevant) variables. Redundant if already specified in call for fitted model specified in object (see details).

FUN
  function used to fit model specified in formula.

nRep
  number of replications or hypothetical values of the exposure to sample for each observation unit.

xBampling
  character string indicating how to sample from the conditional exposure distribution. Possible values are "quantiles" or "random" (see details).

xFit
  an optional fitted object (preferably glm) for the conditional exposure distribution (see details).

perclim
  a numerical vector of the form \(c_{lower} \rightarrow upper\) indicating the extreme percentiles to sample when using "quantiles" as sampling method to sample from the conditional exposure distribution (see details).

... additional arguments (passed to FUN).

Details

The calculated weights are ratios of fitted probabilities or probability densities from the distribution of the mediator model. This model is fitted internally by extracting information from the arguments object, family, data, FUN and ....

For mediation model specification via the object argument, use a formula of the form
\[ M \sim X + C1 + C2, \]
with predictor variables entered in the following prespecified order:

1. exposure X: The first predictor is coded as exposure or treatment.
2. baseline covariates C: All remaining predictor variables are automatically coded as baseline covariates.

It is important to adhere to this prespecified order to enable newWeight to create valid pointers to these different types of predictor variables. This requirement extends to the use of operators different than the + operator, such as the : and * operators (when e.g. adding interaction terms).

For instance, the formula specifications
\[ M \sim X + C1 + C2, \]
\[ M \sim X + C1 + X:C1 + C2 \]
\[ Y \sim X + X:C1 + C1 + C2 \]
will create identical pointers to the different types of variables, as the order of the unique predictor variables is identical in all three specifications.

Furthermore, categorical exposures that are not coded as factors in the original dataset, should be specified as factors in the formula, using the factor function, e.g. \(M \sim \text{factor}(X) + C1 + C2\). Quadratic or higher-order polynomial terms can be included as well, by making use of the I function or by using the poly function. For instance, \(M \sim X + I(X^2) + C1 + C2\) and \(M \sim \text{poly}(X, 2, \text{raw = TRUE}) + C1 + C2\) are equivalent and result in identical pointers to the different types of variables.
The command terms(object, "vartype") (with object replaced by the name of the resulting expanded dataset) can be used to check whether valid pointers have been created.

The type of mediator model can be defined by specifying an appropriate model-fitting function via the FUN argument (its default is glm). This method can only be used with model-fitting functions that require a formula argument.

In contrast to imputation models with categorical exposures, additional arguments need to be specified if the exposure is continuous. All of these additional arguments are related to the sampling procedure for the exposure.

Whereas the number of replications nRep for categorical variables equals the number of levels for the exposure coded as a factor (i.e. the number of hypothetical exposure values), the number of desired replications needs to be specified explicitly for continuous exposures. Its default is 5.

If xFit is left unspecified, the hypothetical exposure levels are automatically sampled from a linear model for the exposure, conditional on a linear combination of all covariates. If one wishes to use another model for the exposure, this default model specification can be overruled by referring to a fitted model object in the xFit argument. Misspecification of this sampling model does not induce bias in the estimated coefficients and standard errors of the natural effect model.

The xsampling argument allows to specify how the hypothetical exposure levels should be sampled from the conditional exposure distribution (which is either entered explicitly using the xFit argument or fitted automatically as described in the previous paragraph). The "random" option randomly samples nRep draws from the exposure distribution, whereas the "quantiles" option (default) samples nRep quantiles at equal-sized probability intervals. Only the latter hence yields fixed exposure levels given nRep and xFit.

In order to guarantee that the entire support of the distribution is being sampled (which might be a concern if nRep is chosen to be small), the default lower and upper sampled quantiles are the 5th and 95th percentiles. The intermittent quantiles correspond to equal-sized probability intervals. So, for instance, if nRep = 4, then the sampled quantiles will correspond to probabilities 0.05, 0.35, 0.65 and 0.95. These default ‘outer’ quantiles can be changed by specifying the perclim argument accordingly. By specifying perclim = NULL, the standard quantiles will be sampled (e.g., 0.2, 0.4, 0.6 and 0.8 if nRep = 4).

Value

A data frame of class c("data.frame", "expData", "weightData"). See expData for its structure.

See Also

newWeight.default, expData

Examples

data(UPBdata)

## example using glm
weightData <- newWeight(negaff ~ att + gender + educ + age,
                         data = UPBdata, nRep = 2)
plot.neLht

Confidence interval plots for linear hypotheses in natural effect models

Description

Confidence interval plots for linear hypotheses in natural effect models.

Usage

```r
## S3 method for class 'neEffdecomp'
plot(x, level = 0.95, transf = identity, ylabels, yticks.at, ...)

## S3 method for class 'neLht'
plot(x, level = 0.95, transf = identity, ylabels, yticks.at, ...)

## S3 method for class 'neLhtBoot'
plot(x, level = 0.95, ci.type = "norm", transf = identity, ylabels, yticks.at, ...)
```

Arguments

- `x`: an object of class `neLht`.
- `level`: the confidence level required.
- `transf`: transformation function to be applied internally on the (linear hypothesis) estimates and their confidence intervals (e.g. `exp` for logit or Poisson regression). The default is identity (i.e. no transformation).
- `ylabels`: character vector containing the labels for the (linear hypothesis) estimates to be plotted on the y-axis.
- `yticks.at`: numeric vector containing the y-coordinates (from 0 to 1) to draw the tick marks for the different estimates and their corresponding confidence intervals.
- `...`: additional arguments.
- `ci.type`: the type of bootstrap intervals required (see `type` argument in `neModel-methods`).

Details

This function is an adapted version of `plot.glht` from the `multcomp` package and yields confidence interval plots for each of the linear hypothesis parameters.

See Also

`neModel`, `neLht`, `neEffdecomp`
Examples

data(UPBdata)

impData <- neImpute(UPB ~ att * negaff + gender + educ + age,
  family = binomial, data = UPBdata)
neMod <- neModel(UPB ~ att0 * att1 + gender + educ + age,
  family = binomial, expData = impData, se = "robust")

lht <- neLht(neMod, linfct = c("att0 = 0", "att0 + att0:att1 = 0",
  "att1 = 0", "att1 + att0:att1 = 0",
  "att0 + att1 + att0:att1 = 0")

## all pairs return identical output
plot(confint(lht), transf = exp)
plot(lht, transf = exp)

plot(neEffdecomp(neMod), transf = exp)
plot(neMod, transf = exp)

plot.neModel

Confidence interval plots for natural effect components

Description

Obtain effect decomposition confidence interval plots for natural effect models.

Usage

## S3 method for class 'neModel'
plot(x, xRef, covLev, level = 0.95,
  transf = identity, ylabels, yticks.at, ...)

## S3 method for class 'neModelBoot'
plot(x, xRef, covLev, level = 0.95,
  ci.type = "norm", transf = identity, ylabels, yticks.at, ...)

Arguments

x a fitted natural effect model object.

xRef a vector including reference levels for the exposure, x* and x, at which natural
effect components need to be evaluated (see details).

covLev a vector including covariate levels at which natural effect components need to
be evaluated (see details).

level the confidence level required.
transf transformation function to be applied internally on the (linear hypothesis) estimates and their confidence intervals (e.g., exp for logit or Poisson regression). The default is identity (i.e., no transformation).

ylabels character vector containing the labels for the (linear hypothesis) estimates to be plotted on the y-axis.

yticks.at numeric vector containing the y-coordinates (from 0 to 1) to draw the tick marks for the different estimates and their corresponding confidence intervals.

... additional arguments.

ci.type the type of bootstrap intervals required (see type argument in neModel-methods).

Details

This function yields confidence interval plots for the natural effect components. These causal parameter estimates are first internally extracted from the neModel object by applying the effect decomposition function neEffdecomp(x, xRef, covLev).

Examples

data(UPBdata)

impData <- neImpute(UPB ~ att * negaff + educ + gender + age,
                   family = binomial, data = UPBdata)
neMod <- neModel(UPB ~ att0 * att1 + educ + gender + age,
                 family = binomial, expData = impData, se = "robust")

plot(neMod)
plot(neMod, transf = exp,
     ylabels = c("PDE", "TDE", "PIE", "TIE", "TE"))
plot(neMod, level = 0.9, xRef = c(-1, 0))

UPBdata UPB data

Description

Data from a survey study that was part of the Interdisciplinary Project for the Optimization of Separation trajectories (IPOS). This large-scale project involved the recruitment of individuals who divorced between March 2008 and March 2009 in four major courts in Flanders. It aimed to improve the quality of life in families during and after the divorce by translating research findings into practical guidelines for separation specialists and by promoting evidence-based policy. This dataset involves a subsample of 385 individuals, namely those who responded to a battery of questionnaires related to romantic relationship and breakup characteristics (De Smet, 2012).
UPBdata  31

Format

A data frame with 385 rows and 9 variables:

att  self-reported anxious attachment level (standardized)
attbin binary version of self-reported anxious attachment level: 1 = higher than sample mean, 0 = lower than sample mean
attcat mult cat egorical version of self-reported anxious attachment level: L = low, M = intermediate, H = high
ergaff level of self-reported experienced negative affectivity (standardized)
initiator initiator of the divorce
gender gender: F = female, M = male
educ education level: either H = high (at least a bachelor’s degree), M = intermediate (having finished secondary school) or L = low (otherwise)
age age (in years)
UPB binary variable indicating whether the individual reported having displayed unwanted pursuit behavior(s) towards the ex-partner

Source


References

Index

adjusted, 14, 15

boot, 16, 17
boot.ci, 14, 15, 20, 21

coeff.neModel (neModel-methods), 20
confint.default, 21
confint.glht, 15
confint.neLht (neLht-methods), 14
confint.neLhtBoot (neLht-methods), 14
confint.neModel (neModel-methods), 20
confint.neModelBoot (neModel-methods), 20

detectCores, 17

detectCores, 17
expData, 2, 3–5, 7, 10, 16–18, 23, 25, 27
expData-methods, 3

factor, 6, 9, 24, 26
family, 16
formula, 5, 6, 8, 9, 16, 23, 24, 26

glht, 11, 12, 15
glm, 10, 16, 17, 21, 27
I, 6, 9, 17, 24, 26

neEffdecomp, 17, 18, 28, 30
neEffdecomp (neLht), 11
neImpute, 2, 3, 4, 7, 10, 18
neImpute.default, 4, 5, 5, 10
neImpute.formula, 4, 5, 7, 8
neLht, 11, 15, 18, 28
neLht-methods, 14
neModel, 2, 5, 7, 10, 12, 14, 16, 17, 21, 28
neModel-methods, 20
newWeight, 2–4, 18, 22, 25
newWeight.default, 22, 23, 23, 27
newWeight.formula, 22, 23, 25, 25

plot.glht, 28
plot.neEffdecomp (plot.neLht), 28
plot.neLht, 12, 15, 28
plot.neLhtBoot (plot.neLht), 28
plot.neModel, 12, 18, 21, 29
plot.neModelBoot (plot.neModel), 29
poly, 6, 9, 17, 24, 26
print, 12

residualPlot, 4, 21
residualPlot.expData (expData-methods), 3
residualPlot.neModel (neModel-methods), 20
residualPlots, 4, 21
residualPlots.expData (expData-methods), 3
residualPlots.neModel (neModel-methods), 20

summary, 12
summary.glht, 14, 15
summary.neLht (neLht-methods), 14
summary.neModel (neModel-methods), 20
SuperLearner, 6, 10

terms, 17

UPBdata, 30

vcov.neModel (neModel-methods), 20

weights, 4, 21
weights.expData (expData-methods), 3
weights.neModel (neModel-methods), 20