Package "hopit"

February 21, 2019

`anova.hopit` *Likelihood Ratio Test Tables*

Description

Compute likelihood ratio test for two or more `hopit` objecs.

Usage

```
## S3 method for class 'hopit'
anova(object, ..., method = c("sequential",
"with.most.complex", 'with.least.complex'),
direction = c("decreasing", "increasing"))
```

Arguments

`object`	an object containing the results returned by a `hopit`.
`...`	additional objects of the same type.
`method`	the method of model comparison. Choose `"sequential"` for 1-2, 2-3, 3-4, ... comparisons or `"with.most.complex"` for 1-2, 1-3, 1-4, ... comparisons, where 1 is the most complex model.
`direction`	determine if complexity of listed models is `"increasing"` or `"decreasing"` (default).

Value

a vector or a matrix with results of the test(s).

Author(s)

Maciej J. Danko

See Also

`print.anova.hopit`, `print.lrt.hopit`, `lrt.hopit`, `hopit`.

Examples

```
# DATA
data(healthsurvey)

# the order of response levels is decreasing (from the best health to
# the worst health)
levels(healthsurvey$health)

# Example 1 ---------------------

# fitting two nested models
model1 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                 heart_attack_or_stroke + poor_mobility + very_poor_grip +
                 depression + respiratory_problems +
                 IADL_problems + obese + diabetes + other_diseases,
               thresh.formula = ~ sex + ageclass + country,
               decreasing.levels = TRUE,
               control = list(trace = FALSE),
               data = healthsurvey)

# model with interaction between hypertension and high_cholesterol
model2 <- hopit(latent.formula = health ~ hypertension * high_cholesterol +
                 heart_attack_or_stroke + poor_mobility + very_poor_grip +
                 depression + respiratory_problems +
                 IADL_problems + obese + diabetes + other_diseases,
               thresh.formula = ~ sex + ageclass + country,
               decreasing.levels = TRUE,
               control = list(trace = FALSE),
               data = healthsurvey)

# Likelihood ratio test
lrt1 <- anova(model1, model2)
lrt1

# print results in shorter form
print(lrt1, short = TRUE)

# equivalently
lrt.hopit(model2, model1)

# Example 2 ---------------------

# fitting additional nested models
model3 <- hopit(latent.formula = health ~ hypertension * high_cholesterol +
                 heart_attack_or_stroke + poor_mobility + very_poor_grip +
                 depression + respiratory_problems +
                 IADL_problems + obese * diabetes + other_diseases,
               thresh.formula = ~ sex + ageclass + country,
               decreasing.levels = TRUE,
               control = list(trace = FALSE),
               data = healthsurvey)
```

```
model4 <- hopit(latent.formula = health ~ hypertension * high_cholesterol +
                  heart_attack_or_stroke + poor_mobility + very_poor_grip +
                  depression + respiratory_problems +
                  IADL_problems + obese * diabetes + other_diseases,
                thresh.formula = ~ sex * ageclass + country,
                decreasing.levels = TRUE,
                control = list(trace = FALSE),
                data = healthsurvey)

# sequential likelihood ratio tests
# model complexity increases so direction = "increasing"
anova(model1, model2, model3, model4,
      direction = "increasing", method = "sequential")

# likelihood ratio tests of the most complex model with the rest
anova(model1, model2, model3, model4,
      direction = "increasing", method = "with.most.complex")

# likelihood ratio tests of the least complex model with the rest
anova(model1, model2, model3, model4,
      direction = "increasing", method = "with.least.complex")
```

---

boot_hopit *Bootstrapping hopit model*

Description

boot_hopit performs bootstrap of a function dependent on fitted model. In each of the bootstrap repetitions a set of new model coefficients is drawn from the multivariate normal distribution, assuming originally estimated model coefficients (see coef.hopit) as a mean and using model variance-covariance matrix (see vcov.hopit). The drawn coefficients are then used to calculate the measure of interest using a function delivered by func parameter.

Usage

```
boot_hopit(model, data, func, nboot = 500, unlist = TRUE,
  boot.only.latent = TRUE, robust.vcov = TRUE, ...)
```

Arguments

model	a fitted hopit model.
data	data used to fit the model.
func	function to be bootstrapped of the form func(model, data, ...).
nboot	number of bootstrap replicates.
unlist	logical indicting if to unlist boot object.
boot.only.latent	logical indicating if to perform the bootstrap only on latent variables.
robust.vcov	see vcov.hopit.
...	other parameters passed to the func.

Value

a list with bootstrapped elements.

Author(s)

Maciej J. Danko

See Also

percentile_CI, getLevels, getCutPoints, latentIndex, standardiseCoef, hopit.

Examples

```
# DATA
data(healthsurvey)

# the order of response levels is decreasing (from the best health to the worst health)
levels(healthsurvey$health)

# fit a model
model1 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility + very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases,
              thresh.formula = ~ sex + ageclass + country,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)

# Example 1 ---------------------
# bootstrapping cut-points

# Function to be bootstrapped
cutpoints <-  function(model, data) getCutPoints(model, plotf = FALSE)$cutpoints
B <- boot_hopit(model = model1, data = healthsurvey,
                func = cutpoints, nboot = 100)

# calculate lower and upper bounds using percentile method
cutpoints.CI <- percentile_CI(B)

# print estimated cutpoints and their confidence intervals
cutpoints(model1, healthsurvey)
cutpoints.CI

# Example 2 ---------------------
# bootstrapping health levels differences

# the function to be bootstrapped
diff_BadHealth <- function(model, data) {
  hl <- getLevels(model = model, formula=~ sex + ageclass, data = data,
                  sep=' ', plotf=FALSE)
  hl$original[,1] + hl$original[,2] - hl$adjusted[,1]- hl$adjusted[,2]
```

```
}

# estimate of the difference
est.org <- diff_BadHealth(model = model1, data = healthsurvey)

# perform the bootstrap
B <- boot_hopit(model = model1, data = healthsurvey,
                func = diff_BadHealth, nboot = 100)

# calculate lower and upper bounds using percentile method
est.CI <- percentile_CI(B)

# plot the difference and its (asymmetrical) confidence intervals
pmar <- par('mar'); par(mar = c(9.5,pmar[2:4]))
m <- max(abs(est.CI))
pos <- barplot(est.org, names.arg = names(est.org), las = 3, ylab = 'Original - Adjusted',
              ylim=c(-m, m), density = 20, angle = c(45, -45), col = c('blue', 'orange'))
for (k in seq_along(pos)) lines(c(pos[k,1],pos[k,1]), est.CI[,k], lwd = 2, col = 2)
abline(h = 0); box(); par(mar = pmar)
```

`getCutPoints` *Calculate threshold cut-points and individual ajusted responses using Jurges' method*

Description

Calculate threshold cut-points and individual ajusted responses using Jurges' method

Usage

```
getCutPoints(model, subset = NULL, plotf = TRUE, mar = c(4, 4, 1, 1),
  oma = c(0, 0, 0, 0), XLab = "Health index", XLab.cex = 1.1,
  YLab = "Counts", YLab.cex = 1.1, decreasing.levels = TRUE,
  group.labels.type = c("middle", "border", "none"))
```

Arguments

`model`	a fitted `hopit` model.
`subset`	an optional vector specifying a subset of observations.
`plotf`	a logical indicating if to plot the results.
`mar, oma`	see `par`.
`XLab, XLab.cex`	label and size of the label for x axis.
`YLab, YLab.cex`	label and size of the label for y axis.
`decreasing.levels`	logical indicating if self-reported health classes are ordered in decreasing order.
`group.labels.type`	position of the legend. One of `middel`, `border`, or `none`.

Value

a list with following components:

`cutpoints` cutpoints for adjusted categorical response levels with corresponding percentiles of latent index.

`adjusted.levels` adjusted categorical response levels for each individual.

Author(s)

Maciej J. Danko

References

\insertRefJurges2007hopit

See Also

`latentIndex`, `standardiseCoef`, `getLevels`, `hopit`.

Examples

```
# DATA
data(healthsurvey)

# the order of response levels is decreasing (from the best health to the
# worst health)
levels(healthsurvey$health)

# Example 1 ---------------------

# fitting a model
model1 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility + very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases,
              thresh.formula = ~ sex + ageclass + country,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)

# health index cut-points
z <- getCutPoints(model = model1)
z$cutpoints

# adjusted health levels for individuals: Jurges method
rev(table(z$adjusted.levels))

# original health levels for individuals
table(model1$y_i)

# adjusted health levels for individuals: Estimated model thresholds
```

```
table(model1$Ey_i)
```

---

getLevels *Summarize adjusted and original self-rated response levels*

Description

Summarize adjusted and original self-rated response levels.

Usage

```
getLevels(model, formula = model$thresh.formula,
  data = environment(model$thresh.formula), decreasing.levels = TRUE,
  sort.flag = FALSE, plotf = TRUE, sep = "_", mar = c(7, 2, 1.5,
  0.5), oma = c(0, 3, 0, 0), YLab = "Fraction [%]", YLab.cex = 1.1,
  legbg = grDevices::adjustcolor("white", alpha.f = 0.4), legbty = "o")
```

Arguments

`model`	a fitted `hopit` model.
`formula`	a formula containing the grouping variables. It is by default set to threshold formula.
`data`	data used to fit the model.
`decreasing.levels`	logical indicating if self-reported health classes are ordered in increasing order.
`sort.flag`	logical indicating if to sort the levels.
`plotf`	a logical indicating if to plot the results.
`sep`	separator for levels names.
`mar, oma`	see `par`.
`YLab, YLab.cex`	label and size of the label for y axis.
`legbg`	legend background color. See `bg` parameter in `legend`.
`legbty`	legend box type. See `bty` parameter in `legend`.

Value

a list with following components:

`original`	frequencies of original response levels for selected groups/categories.
`adjusted`	frequencies of adjusted response levels (Jurges 2007 method) for selected groups/categories.
`N.original`	numbers of original response levels for selected groups/categories.
`N.adjusted`	numbers of adjusted response levels (Jurges 2007 method) for selected groups/categories.

categories	selected groups/categories used in summary.
tab	original vs. adjusted contingency table.
mat	a matrix with columns: grouping variables, original response levels, adjusted response levels. Each row corresponds to a single individual from the data used to fit the model.

Author(s)

Maciej J. Danko

References

\insertRefJurges2007hopit

See Also

getCutPoints, latentIndex, standardiseCoef, hopit.

Examples

```
# DATA
data(healthsurvey)

# the order of response levels is decreasing (from the best health to the
# worst health)
levels(healthsurvey$health)

# fitting a model
model1 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility + very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases,
              thresh.formula = ~ sex + ageclass + country,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)

# Example 1 ---------------------

# summary by country
hl <- getLevels(model=model1, formula=~ country,
                data = healthsurvey,
                sep=' ', plotf=TRUE)

# differences in frequencies between original and adjusted health levels
round(100*(hl$original - hl$adjusted),2)

# extract good and bad health (combine levels)
Org <- cbind(bad = rowSums(hl$original[,1:2]),
             good = rowSums(hl$original[,4:5]))
Adj <- cbind(bad = rowSums(hl$adjusted[,1:2]),
             good = rowSums(hl$adjusted[,4:5]))
```

```
round(100*(Org - Adj),2)

# plot the differences
barplot(t(Org - Adj), beside = TRUE, density = 20, angle = c(-45, 45),
        col = c('pink4', 'green2'),
        ylab = 'Original - adjusted reported health frequencies')
abline(h = 0); box()
legend('top', c('Bad health','Good health'),
       density = 20, angle = c(-45, 45),
       fill = c('pink4', 'green2'), bty = 'n', cex = 1.2)

# in country X the bad health seems to be over-reported and good health
# under reported, in country Z the good health is highly over-reported.

# Example 2 ---------------------

# summary by gender and age
hl <- getLevels(model = model1, formula=~ sex + ageclass,
                data = healthsurvey,
                sep=' ', plotf=TRUE)

# differences in frequencies between original and adjusted health levels
round(100*(hl$original - hl$adjusted),2)

# extract good health levels (combined "Very good" and "Excelent" levels)
Org <- rowSums(hl$original[,4:5])
Adj <- rowSums(hl$adjusted[,4:5])
round(100*(Org - Adj),2)

pmar <- par('mar'); par(mar = c(9.5, pmar[2:4]))
barplot(Org-Adj,
        ylab = 'Original - adjusted reported good health frequencies',
        las = 3,
        density = 20, angle = c(45, -45), col = c('blue', 'orange'))
abline(h = 0); box(); par(mar = pmar)
legend('top', c('Man','Woman'), density = 20, angle = c(-45, 45),
       fill = c('blue', 'orange'), bty = 'n', cex = 1.2)

# the results show that women in general tend to over-report good health,
# while men in ages 50-59 greatly under-report good health.

# more examples can be found in the description of boot.hopit() function.
```

---

getTheta *Extract Theta parameter from the* hopit *model*

Description

Extract Theta parameter from the hopit model

Usage

```
getTheta(model)
```

Arguments

`model`	a fitted `hopit` model.

Author(s)

Maciej J. Danko

`%c%` *Check if one set is a subset of an another subset*

Description

Check if one set is a subset of an another subset

Usage

```
x %c% y
```

Arguments

`x, y`	numeric vectors

Author(s)

Maciej J. Danko

`%notc%` *Not %c% function*

Description

Not %c% function

Usage

```
x %notc% y
```

Arguments

`x, y`	numeric vectors

Author(s)

Maciej J. Danko

`%notin%` *Not %in% function*

Description

Not %in% function

Usage

```
x %notin% y
```

Arguments

`x, y`	numeric vectors

Author(s)

Maciej J. Danko

`healthsurvey` *Artificially generated health survey data*

Description

A dataset containing artificially generated survey data

Usage

```
healthsurvey
```

Format

A data frame with 10000 rows and 11 variables:

ID personal identification number.
health reported health, 5 levels.
diabetes has diabetes? "yes" or "no"?
obese has obese? "yes" or "no"?
IADL_problems problems in Instrumental Activities of Daily Living? "yes" or "no"?
hypertension has hypertension? "yes" or "no"?
high_cholesterol has high cholesterol? "yes" or "no"?
respiratory_problems has respirator problems? "yes" or "no"?
heart_attack_or_stroke had stroke or heart attack? "yes" or "no"?
poor_mobility has poor mobility? "yes" or "no"?

very_poor_grip cannot perform grip strength? "yes" or "no"?

depression has depression? "yes" or "no"?

other_diseases has other diseases? "yes" or "no"?

sex sex/gender: woman or man.

ageclass categorized age: [50,60), [60,70), [70,80), [80,120).

education two levels of education: primary or lower and secondary or higher.

country country: X, Y, or Z.

csw cross-sectional survey weights.

psu primary statistical unit.

Source

Data was randomly generated using probabilities of occurrence of particular combination of diseases, conditions, sex, age, education, reported health, etc. The structure of the data and some probabilities were inspired by WAVE1 SHARE database (DOIs: 10.6103/SHARE.w1.600), see Börsch-Supan et al for methodological details (Börsch-Supan et al. 2013).

The SHARE data collection has been primarily funded by the European Commission through FP5 (QLK6-CT-2001-00360), FP6 (SHARE-I3: RII-CT-2006-062193, COMPARE: CIT5-CT-2005-028857, SHARELIFE: CIT4-CT-2006-028812) and FP7 (SHARE-PREP: N°211909, SHARE-LEAP: N°227822, SHARE M4: N°261982). Additional funding from the German Ministry of Education and Research, the Max Planck Society for the Advancement of Science, the U.S. National Institute on Aging (U01_AG09740-13S2, P01_AG005842, P01_AG08291, P30_AG12815, R21_AG025169, Y1-AG-4553-01, IAG_BSR06-11, OGHA_04-064, HHSN271201300071C) and from various national funding sources is gratefully acknowledged (see www.share-project.org).

None of the individuals (records) represent any true individual (record) of the SHARE database

References

Börsch-Supan A, Brandt M, Hunkler C, et al (2013) Data resource profile: The survey of health, ageing and retirement in europe (share). Int J Epidemiol 42:992–1001. doi: 10.1093/ije/dyt088

Examples

```
# load *healthsurvey* dataset
data(healthsurvey)

# horizontal view on the dataset (omitting ID)
print(t(healthsurvey[1:6,-1]), quote=FALSE, na.print='NA', right=TRUE)
```

`hopit.control` *Auxiliary for controlling the fitting of* `hopit` *model*

Description

Auxiliary function for controlling the fitting of `hopit` model. Use this function to set control parameters of the `hopit` and other related functions.

Usage

```
hopit.control(grad.eps = 3e-05, bgfs.maxit = 10000, cg.maxit = 10000,
  nlm.maxit = 150, bgfs.reltol = 5e-10, cg.reltol = 5e-10,
  nlm.gradtol = 1e-07, nlm.steptol = 1e-07, fit.methods = "BFGS",
  quick.fit = TRUE, trace = TRUE, transform.latent = "none",
  transform.thresh = "none")
```

Arguments

`grad.eps` epsilon for numerical Hessian function.

`bgfs.maxit, cg.maxit, nlm.maxit`
: the maximum number of iterations. See `optim` and `nlm` for details.

`bgfs.reltol, cg.reltol`
: relative convergence tolerance. See `optim` for details.

`nlm.gradtol, nlm.steptol`
: tolerance at which the scaled gradient is considered close enough to zero and minimum allowable relative step length. See `nlm`.

`fit.methods` 'CG', 'BFGS' or both. If both then CG is run first and then BFGS. See `optim`.

`quick.fit` logical, if TRUE extensive `nlm` optimization method is ignored and only BFGS and CG methods are run.

`trace` logical, if to trace model fitting.

`transform.latent, transform.thresh`
: type of the transformation apllied to the all latent or all threshold numerical variables. Possible values:

- "none" - no transformation
- "min" - subtract minimum from a variable
- "scale_01" - ransform variable to fit the range from 0 to 1
- "standardize" or "standardise" - subtract mean from a variable then divide it by it's standard deviation
- "standardize_trunc" or "standardise_trunc" - subtract minimum from a variable then divide it by it's standard deviation

Author(s)

Maciej J. Danko

See Also

hopit

---

hopit *Generalized hierarchical ordered threshold models.*

Description

The ordered response data classifies a measure of interest into ordered categories collected during a survey. For example, if the dependent variable were a happiness rating, then a respondent typically answers a question like: “Taking all things together, would you say you are ... ?” and then selects from response options along the lines of: ”very happy”, ”pretty happy”, ”not too happy”, ”very unhappy” \insertCiteLiao2005hopit. Similarly if interviewees are asked to evaluate their health in general (e.g. “Would you say your health is ... ?”) they may choose among several categories, such as ”very good”, ”good”, ”fair”, ”bad”, and ”very bad” \insertCiteKing2004,Jurges2007,Rebelo2014hopit. In political sciences a respondent may be asked for an opinion about recent legislation (e.g. “Rate your feelings about the proposed legislation.”) and asked to choose among categories like: ”strongly oppose”, ”mildly oppose”, ”indifferent”, ”mildly support”, ”strongly support” \insertCiteGreeneHensher2010hopit. It is easy to imagine other multi-level ordinal variables that might be used during a survey and to which the methodology described below could be applied to.

Practically, it is assumed that when responding to a survey question about their general happiness, health, feeling, attitude or other status, participants assess their true value of this unobserved continuous variable, and project it to a provided discrete scale. The thresholds that each individual uses to categorize their true status into a specific response option may be affected by the choice of a reference group, earlier life experiences, and cross-cultural differences in using scales, and thus, may differ across individuals depending on their gender, age, cultural background, education, and personality traits, among other factors.

From the reporting behavior modeling perspective, one of the main tasks is to compute this continuous estimate of individuals’ underlying, latent measure based on several specific characteristics of the considered response (e.g. health variables or happiness variables) and accounting also for variations in reporting across socio-demographic and cultural groups. More specifically, to build the latent, underlying measure a generalized hierarchical ordered threshold model is fitted, which regresses the reported status/attitude/feeling on two sets of independent variables \insertCiteBoes2006,Green2014hopit. When a dependent reported ordered variable is self-rated health status then the first set of variables - health variables - assesses individuals’ specific aspects of health, and might include chronic conditions, mobility level, difficulties with a range of daily activities, performance on grip strength test, anthropometric measures, lifestyle behaviors, etc. Using the second set of independent variables (threshold variables), the model also adjusts for the differences across socio-demographic and cultural groups like cultural background, gender, age, education, etc. \insertCiteKing2004,Jurges2007hopit.

Ordered threshold models are used to fit ordered categorical dependent variables. The generalized ordered threshold models \insertCiteTerza1985,Boes2006,Green2014hopit are an extension to the ordered threshold models \insertCiteMcKelvey1975hopit. In the latter models, the thresholds are constant, whereas generalized models allows thresholds to be dependent on covariates. \insertCiteGreeneHensher2010,Green2014;textualhopit pointed out that also thresholds must be ordered so that a model has a sense. This motivated Greene and coauthors to call this models *HOPIT*, which stands for hierarchical ordered probit models.

The fitted *hopit* model is used to analyse heterogeneity in reporting behavior. See `standardizeCoef`, `latentIndex`, `getCutPoints`, and `getLevels`.

Usage

```
hopit(latent.formula, thresh.formula = ~1, data, decreasing.levels,
  start = NULL, overdispersion = FALSE, design = list(),
  weights = NULL, link = c("probit", "logit"), control = list(),
  na.action = na.fail)
```

Arguments

`latent.formula`	formula used to model latent variable. It should not contain any threshold variable. To specify interactions between latent and threshold variables see details.
`thresh.formula`	formula used to model threshold variable. It should not contain any latent variable. To specify interactions between latent and threshold variables see details. Any dependent variable (left side of "~" in the formula) will be ignored.
`data`	a data frame including all modeled variables.
`decreasing.levels`	logical indicating if self-reported health classes are ordered in decreasing order.
`start`	a vector with starting coefficient values in the form `c(latent_parameters, threshold_lambdas, th` or `c(latent_parameters, threshold_lambdas, threshold_gammas, logTheta)` if the `overdispersion == TRUE`.
`overdispersion`	logical indicting if to fit additional parameter theta, which models a variance of the error term.
`design`	an optional survey design. Use `svydesign` function to specify the design. The design cannot be specified together with parameter `weights`.
`weights`	optional model weights. Use design to construct survey weights.
`link`	the link function. The possible values are `"probit"` (default) and `"logit"`.
`control`	a list with control parameters. See `hopit.control`.
`na.action`	a function which indicates what should happen when the `data` contain NAs. The default is `na.fail`, which generates an error if missing values are found. The alternative is `na.omit` (or `na.exclude` equivalently), which removes rows with missing values from the `data`. Using `na.pass` will lead to an error.

Details

The function fits generelaized hierarchical ordered threshold models.

`latent.formula` models latent variable. if the response variable is self-rated health then latent measure can depend on different health conditions and diseases (latent variables are called health variables). Latent variables are modeled with parallel regression assumption. According to the assumption, coefficients, which describe the relationship between lowest and all higher response categories, are the same as those coefficients, which describe the relationship between another (e.g. adjacent) lowest and the remaining higher response categories. The predicted latent variable is modeled as a linear function of health variables and corresponding coefficients.

`thresh.formula` models threshold variable. The thresholds (cut points, `alpha`) are modeled by threshold variables `gamma` and intercepts `lambda`. It is assumed that they model contextual characteristics of the respondent (e.g. country, gender, age, etc.). Threshold variables are modeled without parallel regression assumption, thus each threshold is modeled by a variable independently \insertCiteBoes2006,Green2014hopit. `hopit`() function uses parametrization of tresholds proposed by \insertCiteJurges2007;textualhopit.

`decreasing.levels` it is the logical that determines the ordering of levels of the categorical response variable. It is always good to check first the ordering of the levels before starting (see example 1)

It is possible to model interactions, including interactions beetwen latent and threshold variables. Interactions added to the latent formula models only the latent measure and interactions modeled in threshold formula models only thresholds. The general rule for modeling any kind of interactions is to use "*" to specify interactions within latent (or threshold) formula and to use ':' to specify interactions between latent and threshold variables. In the latter case the main effects of an interaction must be also speciffied, i.e. main latent effects must be speciffied in the latent formula and main threshold effect must be speciffied in the threshold formula. See also `Example 3` below.

For more details please see the package vignette: "introduction_to_hopit".

Value

a `hopit` object used by other functions and methods. The object is a list with following components:

`control`	a list with control parameters. See `hopit.control`.
`link`	the used link funtion.
`hasdisp`	logical, was overdispersion modeled?
`use.weights`	logical indicating if any weights were used.
`weights`	vector with model weights.
`latent.formula`	used latent formula. It is updated by cross-interactions if crossinter.formula is delivered.

`latent.mm`	latent model matrix.
`latent.terms`	used latent variables and their interactions.
`cross.inter.latent`	part of the latent formula modeling cross-interactions in the latent model
`thresh.formula`	used threshold formula.
`thresh.mm`	threshold model matrix.
`thresh.extd`	threshold extended model matrix.
`thresh.terms`	used threshold variables and their interactions.
`cross.inter.thresh`	part of the threshold formula modeling cross-interactions in the threshold model
`thresh.no.cov`	logical, are gamma parameters present?
`parcount`	3-element vector with number of parmeters for latent latent variable (beta), threshold intercept (lambda), and threshold covariates (gamma).
`coef`	a vector with coefficients.
`coef.ls`	coefficients as a list.
`start`	vector with starting vlues of coefficients.
`alpha`	estimated individual-specific thresholds.
`y_i`	the response variable.
`y_latent_i`	predicted latent measure.
`Ey_i`	predicted categorical response.
`J`	the number of response levels.
`N`	the number of observations.
`deviance`	deviance.
`LL`	log likelihood.
`AIC`	AIC for models without survey design.
`vcov`	variance-covariance matrix.
`vcov.basic`	variance-covariance matrix ignoring survey design.
`hessian`	a Hessian matrix.
`estfun`	gradient (vector of partial derivatives) of the log likelihood function at estimated coefficient values.
`YYY1,YYY2,YY3`	internal objects used for calculation of gradient and Hessian functions.

Author(s)

Maciej J. Danko

References

\insertAllCited

See Also

coef.hopit, profile.hopit, hopit.control, anova.hopit, vcov.hopit, logLik.hopit, AIC.hopit, summary.hopit, svydesign,

For heterogeneity in reporting behavior analysis see:
standardizeCoef, latentIndex, getCutPoints, getLevels.

Examples

```
# DATA
data(healthsurvey)

# first determine the order of levels of dependent variable
levels(healthsurvey$health)

# Example 1 ---------------------

# the order is decreasing (from the best health to the worst health)
# so we set: decreasing.levels = TRUE
# fitting the model:
model1 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility + very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases,
              thresh.formula = ~ sex + ageclass + country,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)

# summarize the fit:
summary(model1)

# extract parameters in a form of list
cm1 <- coef(model1, aslist = TRUE)

# names of returned coefficients
names(cm1)

# extracting latent health coefficients
cm1$latent.params

# check the fit
profile(model1)

# Example 2 ---------------------

# incorporating survey design
design <- svydesign(ids = ~ country + psu, weights = healthsurvey$csw,
data = healthsurvey)

model2 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                  heart_attack_or_stroke + poor_mobility +
```

```
                  very_poor_grip + depression + respiratory_problems +
                  IADL_problems + obese + diabetes + other_diseases,
                thresh.formula = ~ sex + ageclass + country,
                decreasing.levels = TRUE,
                design = design,
                control = list(trace = FALSE),
                data = healthsurvey)

# compare latent variables
cbind('No survey design' = coef(model1, aslist = TRUE)$latent.par,
'Has survey design' = coef(model2, aslist = TRUE)$latent.par)

# Example 3 ---------------------

# interactions beetween threshold and latent variables
# correctly defined interactions:
model3 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility * very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases +
                sex : depression + sex : diabetes + ageclass:obese,
              thresh.formula = ~ sex * ageclass + country + sex : obese,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)

# badly defined interactions:
## Not run:
# 1) lack of main effect of "other_diseases" in both formulas
# it can be solved by adding " + other_diseases" to the latent formula
model3a <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility + very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases : sex,
              thresh.formula = ~ sex + ageclass + country,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)

# 2) Main effect of sex present in both formulas.
# it can be solved by exchanging "*" into ":" in "other_diseases * sex"
model3b <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility + very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases * sex,
              thresh.formula = ~ sex + ageclass + country,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)

## End(Not run)

# Example 4 ---------------------
```

```
# model with overdispersion
model4 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility + very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases,
              thresh.formula = ~ sex + ageclass + country,
              overdispersion = TRUE,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)

# estimated variance of the error term:
getTheta(model4)

# compare fit of model1 and model4
# Likelihood Ratio Test
print(anova(model1, model4), short = TRUE)

# Example 5 ----------------------

## Not run:
# construct a naive continuous variable:
hs <- healthsurvey
hs$cont_var <- sample(5000:5020,nrow(hs),replace=TRUE)

latent.formula = health ~ hypertension + high_cholesterol +
  heart_attack_or_stroke + poor_mobility + very_poor_grip +
  depression + respiratory_problems +
  IADL_problems + obese + diabetes + other_diseases

# in some cases, when continouse variables are used, the start.glm() function
# may not find starting parameters (R version 3.4.4 (2018-03-15)):
model5 <- hopit(latent.formula = latent.formula,
                thresh.formula = ~ sex + cont_var,
                decreasing.levels = TRUE,
                data = hs)

# one of the solutions is to transform one or more continuous variables:
hs$cont_var_t <- hs$cont_var-min(hs$cont_var)

model5b <- hopit(latent.formula = latent.formula,
                 thresh.formula = ~ sex + cont_var_t,
                 decreasing.levels = TRUE,
                 data = hs)

# this can also be done automatically using control parameter
model5c <- hopit(latent.formula = latent.formula,
                 thresh.formula = ~ sex + cont_var,
                 decreasing.levels = TRUE,
                 control = list(transform.thresh = 'min',
                                transform.latent = 'none'),
                 data = hs)
```

```
model5d <- hopit(latent.formula = latent.formula,
                 thresh.formula = ~ sex + cont_var,
                 decreasing.levels = TRUE,
                 control = list(transform.thresh = 'scale_01',
                                transform.latent = 'none'),
                 data = hs)

model5e <- hopit(latent.formula = latent.formula,
                 thresh.formula = ~ sex + cont_var,
                 decreasing.levels = TRUE,
                 control = list(transform.thresh = 'standardize',
                                transform.latent = 'none'),
                 data = hs)

model5f <- hopit(latent.formula = latent.formula,
                 thresh.formula = ~ sex + cont_var,
                 decreasing.levels = TRUE,
                 control = list(transform.thresh = 'standardize_trunc',
                                transform.latent = 'none'),
                 data = hs)

round(t(rbind(coef(model5b),
              coef(model5c),
              coef(model5d),
              coef(model5e),
              coef(model5f))),4)


## End(Not run)
```

`latentIndex` *Calculate latent index*

Description

Calculate latent index from the fitted model. Latent index is a standardized latent measure, it takes values from 0 to 1, where zero is prescribed to the worse predicted state (maximal observed value for the latent measure) and 1 is prescribed to the best predicted health (minimal observed value for the latent measure).

Usage

```
latentIndex(model, decreasing.levels = TRUE, subset = NULL,
  plotf = FALSE, response = c("data", "fitted", "Jurges"),
  ylab = "Latent index", ...)

healthIndex(model, decreasing.levels = TRUE, subset = NULL,
  plotf = FALSE, response = c("data", "fitted", "Jurges"),
  ylab = "Latent index", ...)
```

Arguments

model	a fitted hopit model.
decreasing.levels	logical indicating if self-reported (e.g. health) classes are ordered in decreasing order.
subset	an optional vector specifying a subset of observations.
plotf	logical indicating if to plot summary figure.
response	X axis plotting option, choose 'data' for raw responses and 'fitted' for model reclassified responses
ylab	a label of y axis.
...	further parameters passed to the plot function.

Value

a vector with latent index for each individual.

Author(s)

Maciej J. Danko

References

\insertRefJurges2007hopit

See Also

standardizeCoef, getCutPoints, getLevels, hopit.

Examples

```
# DATA
data(healthsurvey)

# the order of response levels is decreasing (from the best health to the
# worst health)
levels(healthsurvey$health)

# Example 1 ---------------------

# fitting a model
model1 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility + very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases,
              thresh.formula = ~ sex + ageclass + country,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)
```

```
# calculate health index and plotting reported health status
# vs. health index.
hi <- latentIndex(model1, plotf = TRUE, response = "data",
                  ylab = 'Health index', col='deepskyblue3')

# a simple histogram of the function output
hist(hi)

# calculate health index and plotting adjusted health status (Jurges 2007)
# vs. health index.
latentIndex(model1, plotf = TRUE, response = "Jurges",
                 ylab = 'Health index', col='deepskyblue3')

# calculate health index and plotting predicted health status
# vs. health index.
latentIndex(model1, plotf = TRUE, response = "fitted",
                 ylab = 'Health index', col='deepskyblue3')
```

`percentile_CI` *Calculating confidence intervals of bootstrapped function using percentile method*

Description

Calculating confidence intervals of bootstrapped function using percentile method

Usage

```
percentile_CI(boot, alpha = 0.05, bounds = c("both", "lo", "up"))
```

Arguments

`boot`	a matrix or list of vectors with bootstrapped elements. If a list then each element of the list is one replication.
`alpha`	significance level.
`bounds`	one of `"both"`, `"lo"`, `"up"`.

Author(s)

Maciej J. Danko

See Also

`boot_hopit`, `getLevels`, `getCutPoints`, `latentIndex`, `standardiseCoef`, `hopit`.

Examples

```
# see examples in boot_hopit() function.
```

`standardizeCoef` *Standardization of coefficients*

Description

Calculate standardized coefficients (e.g. disability weights for health variables) using the predicted latent measure obtained from the model.
In the self-rated health example the standardized coefficients are called disability weights \insertCiteJurges2007;textualhopit and are calculated for each health variable to provide information about the impact of a specific health measure on the latent index (see `latentIndex`). The disability weight for a health variable is equal to the ratio of corresponding health coefficient and the difference between the lowest and highest values of predicted latent health. In other words, disability weight reduces latent index by some given amount or percentage (i.e. of every individual is reduced by the same amount if heart attack or other heart problems are present)\insertCiteJurges2007;textualhopit.

Usage

```
standardizeCoef(model, ordered = TRUE, plotf = FALSE,
  plotpval = FALSE, mar = c(15, 4, 1, 1), oma = c(0, 0, 0, 0),
  YLab = "Disability weight", YLab.cex = 1.1, namesf = identity, ...)

standardiseCoef(model, ordered = TRUE, plotf = FALSE,
  plotpval = FALSE, mar = c(15, 4, 1, 1), oma = c(0, 0, 0, 0),
  YLab = "Disability weight", YLab.cex = 1.1, namesf = identity, ...)

disabilityWeights(model, ordered = TRUE, plotf = FALSE,
  plotpval = FALSE, mar = c(15, 4, 1, 1), oma = c(0, 0, 0, 0),
  YLab = "Disability weight", YLab.cex = 1.1, namesf = identity, ...)
```

Arguments

`model`	a fitted `hopit` model.
`ordered`	logical indicating if to order the disability weights.
`plotf`	logical indicating if to plot results.
`plotpval`	logical indicating if to plot p-values.
`mar, oma`	see `par`.
`YLab, YLab.cex`	label and `cex` of y axis.
`namesf`	a vector of names of coefficients or one argument function that modifies names of coefficients.
`...`	arguments passed to `boxplot`.

Value

a vector with standardized coefficients.

Author(s)

Maciej J. Danko

References

\insertRefJurges2007hopit

See Also

latentIndex, getCutPoints, getLevels, hopit.

Examples

```
# DATA
data(healthsurvey)

# the order of response levels is decreasing (from the best health to the worst health)
levels(healthsurvey$health)

# Example 1 ----------------------

# fitting a model
model1 <- hopit(latent.formula = health ~ hypertension + high_cholesterol +
                heart_attack_or_stroke + poor_mobility + very_poor_grip +
                depression + respiratory_problems +
                IADL_problems + obese + diabetes + other_diseases,
              thresh.formula = ~ sex + ageclass + country,
              decreasing.levels = TRUE,
              control = list(trace = FALSE),
              data = healthsurvey)

# A function that modifies coefficient names.
txtfun <- function(x) gsub('_',' ',substr(x,1,nchar(x)-3))

# Calculate and plot disability weights
sc <- standardizeCoef(model1, plotf = TRUE, namesf = txtfun)
sc
```

`svy.varcoef.hopit` *Calculation of variance-covariance matrix for specified survey design (experimental function)*

Description

This is a modification of `survey:::svy.varcoef`. In the original approach `estfun` is calcualted from glm's working residuals:
`estfun <- model.matrix(glm.object) * resid(glm.object, "working") * glm.object$weights`
In the hopit package estfun is directly calculated as a gradient (vector of partial derivatives) of log likelihood function.

Usage

```
svy.varcoef.hopit(Ainv, estfun, design)
```

Arguments

`Ainv`	a variance-covariance matrix.
`estfun`	LL gradient function.
`design`	a `survey.design` object.

Details

Based on the survey v3.35 package.

Author(s)

Thomas Lumley, modified by Maciej J. Danko

Index