Package ‘DAMisc’

April 23, 2020

Type Package

Title Dave Armstrong’s Miscellaneous Functions

Version 1.5.4

Imports lattice, grid, car, effects, rstan, gdata, ggplot2, MASS, nnet, splines, xtable, boot, optiscale, fANCOVA, AICcmodavg, latticeExtra, coda, glue, clarkeTest

Suggests carData

Description Miscellaneous set of functions I use in my teaching either at the University of Western Ontario or the Inter-university Consortium for Political and Social Research (ICPSR) Summer Program in Quantitative Methods. Broadly, the functions help with presentation and interpretation of LMs and GLMs, but also implement some new tools like Alternating Least Squares Optimal Scaling for dependent variables, a Bayesian analog to the ALSOS algorithm. There are also tools to help understand interactions in both LMs and binary GLMs.

License GPL (>= 2)

LazyLoad yes

RoxygenNote 7.0.2

NeedsCompilation no

Author Dave Armstrong [aut, cre]

Maintainer Dave Armstrong <dave@quantoid.net>

Repository CRAN

Date/Publication 2020-04-23 14:20:02 UTC

R topics documented:

  DAMisc-package .................................................. 3
  aclp ................................................................. 4
  alsosDV ............................................................ 5
  aveEffPlot .......................................................... 6
  balsos .............................................................. 7
  BGMtest ............................................................. 8
  binfit .............................................................. 9
R topics documented:

btscs ......................................................... 10
cat2Table ..................................................... 11
changeSig ..................................................... 12
combTest ...................................................... 13
crSpanTest ................................................... 13
crTest ........................................................ 15
cv.lo2 ........................................................ 16
DAintfun ....................................................... 17
DAintfun2 ...................................................... 18
DAintfun3 ...................................................... 20
effect_logistf ............................................... 22
france .......................................................... 23
glmChange .................................................... 23
glmChange2 ................................................... 25
inspect ........................................................ 26
intEff ........................................................... 27
InteractionEx ............................................... 28
intQualQuant ............................................... 28
loessDeriv ................................................... 31
logit_cc ....................................................... 31
makeHypSurv ............................................... 32
mnlAveEffPlot ............................................... 33
mnlChange ................................................... 34
mnlChange2 .................................................. 35
mnlfit ......................................................... 36
mnlSig ........................................................ 37
NKnots ........................................................ 38
NKnotsTest ................................................... 40
oc2plot ....................................................... 41
ordAveEffPlot ............................................... 42
ordChange ..................................................... 44
ordChange2 ................................................... 45
ordfit ........................................................ 46
outEff ........................................................ 47
outXT .......................................................... 48
panel.2cat .................................................... 49
panel.ci ....................................................... 50
panel.doublerug ............................................ 51
panel.transci ............................................... 52
pgumbel ....................................................... 52
plot.balsos .................................................. 53
plot.loess .................................................... 54
plot.secdiff ................................................ 55
poisfit ........................................................ 55
poisGOF ...................................................... 56
powerTrans .................................................. 57
pre ............................................................. 57
prepanel.ci ................................................... 59
These are functions that help present linear model results. Largely, they represent alternatives in presentation to other R packages. For example, the `factorplot` function offers an alternative to David Firth’s `qvcalc` package. This function calculates and presents exact variances of all simple contrasts. Both `DAintfun` and `DAintfun2` are alternative ways of presenting interactions between two continuous variables. `DAintfun2` gives results in line with the suggestions in Brambor, Clark and Goldner (2006).
Author(s)

Dave Armstrong
Maintainer: Dave Armstrong <davearmstrong.ps@gmail.com>

References


aclp Example data for btscs function

Description

A subset of data from Alvarez et. al. (1996).

Format

A data frame with 4126 observations on the following 7 variables.

cname Country name
country Numeric country identifier
year Year of observation
reg A dichotomous variable coded 1 for dictatorship, 0 for democracy
gdpw GDP/worker, 1985 prices
popg Population growth
democ A dichotomous variable coded 1 for democracy, 0 for dictatorship, (1-reg)

References

alsosDV

Alternating Least Squares Optimal Scaling

Description

Estimates the Alternating Least Squares Optimal Scaling (ALSOS) solution for qualitative dependent variables.

Usage

alsosDV(form, data, maxit = 30, level = 2, process = 1, starts = NULL, ...)

Arguments

- **form**: A formula with a dependent variable that will be optimally scaled
- **data**: A data frame.
- **maxit**: Maximum number of iterations of the optimal scaling algorithm.
- **level**: Measurement level of the dependent variable 1=Nominal, 2=Ordinal
- **process**: Nature of the measurement process: 1=discrete, 2=continuous. Basically identifies whether tied observations will continue to be tied in the optimally scaled variable (1) or whether the algorithm can untie the points (2) subject to the overall measurement constraints in the model.
- **starts**: Optional starting values for the optimal scaling algorithm.
- **...**: Other arguments to be passed down to `lm`.

Details

alsosDV estimates the Alternating Least Squares Optimal Scaling solution on the dependent variable.

Value

A list with the following elements:

- **result**: The result of the optimal scaling process
- **data**: The original data frame with additional columns adding the optimally scaled DV
- **iterations**: The iteration history of the algorithm
- **form**: Original formula

Author(s)

Dave Armstrong
aveEffPlot

Average Effect Plot for Generalized Linear Models

Description

For objects of class \texttt{glm}, it calculates the change the average predicted probability (like the one calculated by \texttt{glmChange2}) for a hypothetical candidate set of values of a covariate.

Usage

\begin{verbatim}
aveEffPlot(
  obj, 
  varname, 
  data, 
  R = 1500, 
  nvals = 25, 
  plot = TRUE, 
  returnSim = FALSE, 
  ...
)
\end{verbatim}

Arguments

- \texttt{obj} A model object of class \texttt{glm}.
- \texttt{varname} Character string giving the variable name for which average effects are to be calculated.
- \texttt{data} Data frame used to fit \texttt{object}.
- \texttt{R} Number of simulations to perform.
- \texttt{nvals} Number of evaluation points at which the average probability will be calculated.
- \texttt{plot} Logical indicating whether plot should be returned, or just data (if \texttt{FALSE}).
- \texttt{returnSim} Logical indicating whether simulated predicted probabilities should be returned.
- \texttt{...} Other arguments to be passed down to \texttt{xyplot}.

Details

The function plots the average effect of a model covariate, for objects of class \texttt{glm}. The function does not work with \texttt{poly} unless the coefficients are provided as arguments to the command in the model (see example below).

References


Young, Forrest, Jan de Leeuw and Yoshio Takane. 1976. ‘Regression with Qualitative and Quantitative Variables: An Alternating Least Squares Method with Optimal Scaling Features’ Psychometrika, 41:502-529.
Value
A plot or a data frame

Author(s)
Dave Armstrong

Examples

```r
data(france)
p <- poly(france$lrself, 2)
left.mod <- glm(voteleft ~ male + age + retnat +
poly(lrself, 2, coefs=attr(p, "coefs")), data=france, family=binomial)
aveEffPlot(left.mod, "age", data=france, plot=FALSE)
```

---

### balsos

**Bayesian Alternating Least Squares Optimal Scaling**

**Description**

Estimates a Bayesian analog to the Alternating Least Squares Optimal Scaling (ALSOS) solution for qualitative dependent variables.

**Usage**

```r
balsos(
  formula,
  data,
  iter = 2500,
  chains = 1,
  alg = c("NUTS", "HMC", "Fixed_param"),
  ...
)
```

**Arguments**

- `formula`  A formula with a dependent variable that will be optimally scaled
- `data`  A data frame.
- `iter`  Number of samples for the MCMC sampler.
- `chains`  Number of parallel chains to be run.
- `alg`  Algorithm used to do sampling. See `stan` for more details.
- `...`  Other arguments to be passed down to `stanfit`.
Details

balsos estimates a Bayesian analog to the Alternating Least Squares Optimal Scaling solution on the dependent variable. This permits testing linearity assumptions on the original scale of the dependent variable.

Value

A list with the following elements:

* fit The fitted stan output
* y The dependent variable values used in the regression.
* X The design matrix for the regression

Author(s)

Dave Armstrong

References


Young, Forrest, Jan de Leeuw and Yoshio Takane. 1976. ‘Regression with Qualitative and Quantitative Variables: An Alternating Least Squares Method with Optimal Scaling Features’ Psychometrika, 41:502-529.

BGMtest

Tests the five Berry, Golder and Milton (2012) Interactive Hypothesis

Description

This function tests the five hypotheses that Berry, Golder and Milton identify as important when two quantitative variables are interacted in a linear model.

Usage

```
BGMtest(obj, vars, digits = 3, level = 0.05, two.sided = TRUE)
```

Arguments

* obj An object of class `lm`.
* vars A vector of two variable names giving the two quantitative variables involved in the interaction. These variables must be involved in one, and only one, interaction.
* digits Number of digits to be printed in the summary.
* level Type I error rate for the tests.
* two.sided Logical indicating whether the tests should be two-sided (if `TRUE`, the default) or one-sided (if `FALSE`).
Value

A matrix giving five t-tests.

Author(s)

Dave Armstrong

Examples

data(Duncan, package="carData")
mod <- lm(prestige ~ income*education + type, data=Duncan)
BGMtest(mod, c("income", "education"))

---

binfit

Scalar Measures of Fit for Binary Variable Models

Description

Calculates scalar measures of fit for models with binary dependent variables along the lines described in Long (1997) and Long and Freese (2005).

Usage

binfit(mod)

Arguments

mod

A model of class glm with family=binomial.

Details

binfit calculates scalar measures of fit (many of which are pseudo-R-squared measures) to describe how well a model fits data with a binary dependent variable.

Value

A named vector of scalar measures of fit

Author(s)

Dave Armstrong
References


Examples

```r
data(france)
left.mod <- glm(voteleft ~ male + age + retnat +
poly(lrself, 2), data=france, family=binomial)
binfit(left.mod)
```

---

**btscs**

*Generate Spells for Binary Variables*

Description

Beck et. al. (1998) identified that binary time-series cross-section data are discrete-time duration data and time dependence can be modeled in a logistic regression by including a flexible function (e.g., cubic spline) of time since the last event as a covariate. This function creates the variable identifying time since last event.

Usage

```r
btscs(data, event, tvar, csunit, pad.ts = FALSE)
```

Arguments

- `data`: A data frame.
- `event`: Character string giving the name of the dichotomous variable identifying the event (where an event is coded 1 and the absence of an event is coded 0).
- `tvar`: Character string giving the name of the time variable.
- `csunit`: Character string giving the name of the cross-sectional unit.
- `pad.ts`: Logical indicating whether the time-series should be filled in, when panels are unbalanced.

Value

The original data frame with one additional variable. The `spell` variable identifies the number of observed periods since the last event.

Author(s)

Dave Armstrong
References


Examples

library(splines)
## Data from Alvarez et. al. (1996)
data(aclp)
newdat <- btscs(aclp, "democ", "year", "country")

# Estimate Model with and without spell
full.mod <- glm(democ ~ log(gdpw) + popg + bs(spell, df=4), data=newdat, family=binomial)
restricted.mod <- glm(democ ~ log(gdpw) + popg, data=newdat, family=binomial)

# Incremental F-test of time dependence
anova(restricted.mod, full.mod, test='Chisq')

---

cat2Table

Fitted Values and CIs for 2-Categorical Interactions

Description

This function makes a table of fitted values and confidence intervals for all of the combinations of two categorical variables in an interaction.

Usage

cat2Table(eff.obj, digits = 2, rownames = NULL, colnames = NULL)

Arguments

eff.obj An object generated by effect from the effects package where the effect is calculated for two factors involved in an interaction.
digits Number of digits of the fitted values and confidence intervals to print.
rownames An optional vector of row names for the table, if NULL, the levels of the factor will be used
colnames An optional vector of column names for the table, if NULL, the levels of the factor will be used

Value

A matrix of fitted values and confidence intervals
changeSig

Author(s)

Dave Armstrong

Examples

library(effects)
data(Duncan, package="carData")
Duncan$inc.cat <- cut(Duncan$income, 3)
mod <- lm(prestige ~ inc.cat*type + income, data=Duncan)
e1 <- effect("inc.cat*type", mod)
cat2Table(e1)

---

changeSig

Regions of Statistical Significance in Interactions

Description

Calculates the regions of statistical significance in interaction and identifies the points at which the statistical significance of conditional coefficients changes.

Usage

changeSig(obj, vars, alpha = 0.05)

Arguments

obj A model of class glm or class lm.
vars A character vector of the names of the two variables involved in the interaction.
alpha Critical p-value of the test.

Value

Printed output that identifies the change-points in statistical significance.

Author(s)

Dave Armstrong
combTest

Test for Combining Categories in Multinomial Logistic Regression Models.

Description
Tests the null hypothesis that categories can be combined in Multinomial Logistic Regression Models.

Usage
combTest(obj)

Arguments
obj An object of class multinom.

Value
A matrix of test statistics and p-values.

Author(s)
Dave Armstrong

Examples
library(nnet)
data(france)
mnl.mod <- multinom(vote ~ age + male + retnat + lrself, data=france)
combTest(mnl.mod)

---

crSpanTest

Test of Span Parameter in linearity for Component + Residual Plots

Description
This function performs crTest for a user-defined range of span parameters, optionally allowing for multiple testing corrections in the p-values.
Usage

```r
crSpanTest(
  model, 
  spfromto, 
  n = 10, 
  adjust.method = "none", 
  adjust.type = c("none", "across", "within", "both")
)
```

Arguments

- `model`: A model object of class `lm`
- `spfromto`: A vector of two values across which a range of `n` span values will be generated and tested.
- `n`: Number of span parameters to test.
- `adjust.type`: String giving the values over which the multiple testing correction will be performed. Here, ‘both’ refers to a multiple testing correction done over all span parameters and all variables in the model. ‘within’ means the multiple testing correction should be done within each model, but not across the span parameters and ‘across’ means that the multiple testing correction should be for each variable across the various span parameters, but not across variables within the same model. ‘none’ refers to a pass-through option of no multiple testing procedure.

Value

A list with two elements:

- `x`: Sequence of span values used in testing
- `y`: p-values for each variable for each span parameter

Author(s)

Dave Armstrong

Examples

```r
data(Prestige, package="carData")
mod <- lm(prestige ~ income + education + women, data=Prestige)
tmp <- crSpanTest(mod, c(.1, .9), adjust.method="holm", adjust.type="both")
matplot(tmp$x, tmp$y, type="l")
```
Test of linearity for Component + Residual Plots

Description

This function estimates a linear model and a loess model on the component-plus-residual plot (i.e., a partial residual plot) for each quantitative variable in the model. The residual sums of squares for each are used to calculate an F-test for each quantitative variable.

Usage

```r
crTest(
  model, 
  adjust.method = "none", 
  cat = 5, 
  var = NULL, 
  span.as = TRUE, 
  span = 0.75, 
  ...
)
```

Arguments

- `model`: A model object of class `lm`
- `cat`: Number of unique values below which numeric variables are considered categorical for the purposes of the smooth.
- `var`: Character string indicating the term desired for testing. If left `NULL`, the default value, all numeric variables will be tested.
- `span.as`: Logical indicating whether the span should be automatically selected through AICC or GCV.
- `span`: Span to be passed down to the `loess` function if `span.as` = `FALSE`.
- `...`: Other arguments to be passed down to the call to `loess`.

Value

A matrix with the following columns for each variable:

- `RSSp`: Residual sum-of-squares for the parametric (linear) model.
- `RSSnp`: Residual sum-of-squares for the non-parametric (loess) model.
- `DFnum`: Numerator degrees of freedom for the F-test: tr(S)-(k+1).
- `DFdenom`: Denominator degrees of freedom for the F-test: n-tr(S).
- `F`: F-statistic
- `p`: p-value, potentially adjusted for multiple comparisons.
cv.lo2

Cross-validating Loess curve

Description
Function provides the cross-validation error for the loess curve in a manner that is amenable to optimization of the span.

Usage
cv.lo2(
  span,
  form,
  data,
  cost = function(y, yhat) mean((y - yhat)^2, na.rm = TRUE),
  K = n,
  numiter = 100,
  which = c("corrected", "raw")
)

Arguments

  span          The span of the loess smoother.
  form          The formula that identifies the model
  data          A data frame containing the required variables.
  cost          Cost function to be passed down to loess.
  K             Number of folds for the cross-validation
  numiter       Number of times over which the cv error will be aggregated
  which         Return raw or corrected cv error

Value
The cross-validation error from the loess curve.

Author(s)
Dave Armstrong
DAintfun

Surface Plots for Two-Way Interactions

Description

Makes surface plots to display interactions between two continuous variables

Usage

DAintfun(
  obj,  
  varnames,  
  theta = 45,  
  phi = 10,  
  xlab = NULL,  
  ylab = NULL,  
  zlab = NULL,  
  hcols = NULL,  
  ...  
)

Arguments

obj A model object of class lm
varnames A two-element character vector where each element is the name of a variable involved in a two-way interaction.
theta Angle defining the azimuthal viewing direction to be passed to persp
phi Angle defining the colatitude viewing direction to be passed to persp
xlab Optional label to put on the x-axis, otherwise if NULL, it will take the first element of varnames
ylab Optional label to put on the y-axis, otherwise if NULL, it will take the second element of varnames
zlab Optional label to put on the z-axis, otherwise if NULL, it will be ‘Predictions’
hcols Vector of four colors to color increasingly high density areas
...

Other arguments to be passed down to the initial call to persp

Details

This function makes a surface plot of an interaction between two continuous covariates. If the model is

\[ y_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + b_3 x_{i1} \times x_{i2} + \ldots + e_i, \]

this function plots \( b_1 x_{i1} + b_2 x_{i2} + b_3 x_{i1} \times x_{i2} \) for values over the range of \( X_1 \) and \( X_2 \). The highest 75%, 50% and 25% of the bivariate density of \( X_1 \) and \( X_2 \) (as calculated by \texttt{sm.density} from the \texttt{sm} package) are colored in with colors of increasing gray-scale.
Value

x1  Values of the first element of varnames used to make predictions.
x2  Values of the second element of varnames used to make predictions.
pred  The predictions based on the values x1 and x2.
graph  A graph is produced, but no other information is returned.

Author(s)

Dave Armstrong

Examples

data(InteractionEx)
mod <- lm(y ~ x1*x2 + z, data=InteractionEx)
DAintfun(mod, c("x1", "x2"))

DAintfun2  Conditional Effects Plots for Interactions in Linear Models

Description

Generates two conditional effects plots for two interacted continuous covariates in linear models.

Usage

DAintfun2(
  obj,
  varnames,
  varcov = NULL,
  rug = TRUE,
  ticksize = -0.03,
  hist = FALSE,
  hist.col = "gray75",
  nclass = c(10, 10),
  scale.hist = 0.5,
  border = NA,
  name.stem = "cond_eff",
  xlab = NULL,
  ylab = NULL,
  plot.type = "screen"
)
Arguments

- **obj**: A model object of class `lm`.
- **varnames**: A two-element character vector where each element is the name of a variable involved in a two-way interaction.
- **varcov**: A variance-covariance matrix with which to calculate the conditional standard errors. If `NULL`, it is calculated with `vcov(obj)`.
- **rug**: Logical indicating whether a rug plot should be included.
- **ticksize**: A scalar indicating the size of ticks in the rug plot (if included) positive values put the rug inside the plotting region and negative values put it outside the plotting region.
- **hist**: Logical indicating whether a histogram of the x-variable should be included in the plotting region.
- **hist.col**: Argument to be passed to `polygon` indicating the color of the histogram bins.
- **nclass**: Vector of two integers indicating the number of bins in the two histograms, which will be passed to `hist`.
- **scale.hist**: A scalar in the range (0,1] indicating how much vertical space in the plotting region the histogram should take up.
- **border**: Argument passed to `polygon` indicating how the border of the histogram bins should be printed (NA for no border).
- **name.stem**: A character string giving filename to which the appropriate extension will be appended.
- **xlab**: Optional vector of length two giving the x-labels for the two plots that are generated. The first element of the vector corresponds to the figure plotting the conditional effect of the first variable in `varnames` given the second and the second element of the vector corresponds to the figure plotting the conditional effect of the second variable in `varnames` conditional on the first.
- **ylab**: Optional vector of length two giving the y-labels for the two plots that are generated. The first element of the vector corresponds to the figure plotting the conditional effect of the first variable in `varnames` given the second and the second element of the vector corresponds to the figure plotting the conditional effect of the second variable in `varnames` conditional on the first.
- **plot.type**: One of ‘pdf’, ‘png’, ‘eps’ or ‘screen’, where the one of the first three will produce two graphs starting with `name.stem` written to the appropriate file type and the third will produce graphical output on the screen.

Details

This function produces graphs along the lines suggested by Brambor, Clark and Golder (2006) and Berry, Golder and Milton (2012), that show the conditional effect of one variable in an interaction given the values of the conditioning variable. This is an alternative to the methods proposed by John Fox in his `effects` package, upon which this function depends heavily.

Specifically, if the model is

\[ y_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + b_3 x_{i1} \times x_{i2} + \ldots + e_i, \]
this function plots calculates the conditional effect of $X_1$ given $X_2$

$$\frac{\partial y}{\partial X_1} = b_1 + b_3 X_2$$

and the variances of the conditional effects

$$V(b_1 + b_3 X_2) = V(b_1 + X_2^2 V(b_3) + 2(1)(X_2)V(b_1, b_3))$$

for different values of $X_2$ and then switches the places of $X_1$ and $X_2$, calculating the conditional effect of $X_2$ given a range of values of $X_1$. 95% confidence bounds are then calculated and plotted for each conditional effects along with a horizontal reference line at 0.

Value

graphs Either a single graph is printed on the screen (using \texttt{par(mfrow=c(1,2))}) or two figures starting with \texttt{name.stem} are produced where each gives the conditional effect of one variable based on the values of another.

Author(s)

Dave Armstrong

References


Examples

```r
data(InteractionEx)
mod <- lm(y ~ x1*x2 + z, data=InteractionEx)
DAintfun2(mod, c("x1", "x2"), hist=TRUE, scale.hist=.3)
```

**Description**

Generates two conditional effects plots for two interacted continuous covariates in linear models.
Usage

DAintfun3(
  obj,
  varnames,
  varcov = NULL,
  name.stem = "cond_eff",
  xlab = NULL,
  ylab = NULL,
  plot.type = "screen"
)

Arguments

obj A model object of class `lm`
varnames A two-element character vector where each element is the name of a variable involved in a two-way interaction.
varcov A variance-covariance matrix with which to calculate the conditional standard errors. If NULL, it is calculated with `vcov(obj)`.
name.stem A character string giving filename to which the appropriate extension will be appended
xlab Optional vector of length two giving the x-labels for the two plots that are generated. The first element of the vector corresponds to the figure plotting the conditional effect of the first variable in `varnames` given the second and the second element of the vector corresponds to the figure plotting the conditional effect of the second variable in `varnames` conditional on the first.
ylab Optional vector of length two giving the y-labels for the two plots that are generated. The first element of the vector corresponds to the figure plotting the conditional effect of the first variable in `varnames` given the second and the second element of the vector corresponds to the figure plotting the conditional effect of the second variable in `varnames` conditional on the first.
plot.type One of ‘pdf’, ‘png’, ‘eps’ or ‘screen’, where the one of the first three will produce two graphs starting with `name.stem` written to the appropriate file type and the third will produce graphical output on the screen.

Details

This function does the same thing as `DAintfun2`, but presents effects only at the mean of the conditioning variable and the mean +/- 1 standard deviation.

Value
graphs Either a single graph is printed on the screen (using `par(mfrow=c(1,2))`) or two figures starting with `name.stem` are produced where each gives the conditional effect of one variable based on the values of another.

Author(s)

Dave Armstrong
References


Examples

data(InteractionEx)
mod <- lm(y ~ x1*x2 + z, data=InteractionEx)
DAintfun3(mod, c("x1", "x2"))

definitions

_description

Plots the effect of a variable in a model estimated with Firth Logit.

Usage

effect_logistf(var, obj, data, ...)

Arguments

<table>
<thead>
<tr>
<th>var</th>
<th>A character string giving the name of the variable whose effect is to be generated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>An object of class logistf.</td>
</tr>
<tr>
<td>data</td>
<td>A data frame.</td>
</tr>
<tr>
<td>...</td>
<td>Other arguments to be passed down to the Effect function.</td>
</tr>
</tbody>
</table>

Details

The effect.logistf function calculates the effect (predicted probabilities) of a variable in a Firth logit model estimated with the logistf function. The function estimates the analogous glm. It then replaces the coefficient vector in that model object with the Firth logit coefficients. It also puts the variance-covariance matrix from the Firth logit in the model object and uses a custom extractor function in the Effect function to extract that variance-covariance matrix rather than the one usually extracted with vcov. Note that variability and confidence intervals for the effects will not be calculated using profile likelihood as they are in the Firth logit, but will be calculated using the appropriate variance-covariance matrix.

Value

An object of class eff that can be used with other functions from the effects package.
Author(s)

Dave Armstrong

Example data for factorplot function

Description

A subset of data from the 1994 Eurobarometer for France

Format

A data frame with 542 observations on the following 5 variables.

- `lrself`: respondent’s left-right self-placement on a 1(left)-10(right) scale
- `male`: a dummy variable coded 1 for males and 0 for females
- `age`: respondent’s age
- `vote`: a factor indicating vote choice with levels PCF, PS, Green, RPR and UDF
- `retnat`: a factor indicating the respondent’s retrospective national economic evaluation with levels Better, Same and Worse
- `voteleft`: a dichotomous variable where 1 indicates a vote for a left party, 0 otherwise

References


Maximal First Differences for Generalized Linear Models

Description

For objects of class `glm`, it calculates the change in predicted responses, for maximal discrete changes in all covariates holding all other variables constant at typical values.

Usage

```r
glmChange(
  obj, 
  data, 
  typical.dat = NULL, 
  diffchange = c("range", "sd", "unit"), 
  sim = FALSE, 
  R = 1000
)
```
Arguments

- **obj**: A model object of class `glm`.
- **data**: Data frame used to fit `obj`.
- **typical.dat**: Data frame with a single row containing values at which to hold variables constant when calculating first differences. These values will be passed to `predict`, so factors must take on a single value, but have all possible levels as their levels attribute.
- **diffchange**: A string indicating the difference in predictor values to calculate the discrete change. `range` gives the difference between the minimum and maximum, `sd` gives plus and minus one-half standard deviation change around the median and `unit` gives a plus and minus one-half unit change around the median.
- **sim**: Logical indicating whether simulated confidence bounds on the difference should be calculated and presented.
- **R**: Number of simulations to perform if `sim` is `TRUE`.

Details

The function calculates the changes in predicted responses for maximal discrete changes in the covariates, for objects of class `glm`. This function works with polynomials specified with the `poly` function. It also works with multiplicative interactions of the covariates by virtue of the fact that it holds all other variables at typical values. By default, typical values are the median for quantitative variables and the mode for factors. The way the function works with factors is a bit different. The function identifies the two most different levels of the factor and calculates the change in predictions for a change from the level with the smallest prediction to the level with the largest prediction.

Value

A list with the following elements:

- **difs**: A matrix of calculated first differences
- **minmax**: A matrix of values that were used to calculate the predicted changes

Author(s)

Dave Armstrong

Examples

```r
data(france)
left.mod <- glm(voteleft ~ male + age + retnat + poly(lrself, 2), data=france, family=binomial)
typical.france <- data.frame(
  retnat = factor(1, levels=1:3, labels=levels(france$retnat)),
  age = 35, stringsAsFactors=TRUE )
glmChange(left.mod, data=france, typical.dat=typical.france)
```
**Description**

For objects of class `glm`, it calculates the change in predicted responses, for discrete changes in a covariate holding all other variables at their observed values.

**Usage**

```r
glmChange2(obj, varname, data, change = c("unit", "sd"), R = 1500)
```

**Arguments**

- `obj`: A model object of class `glm`.
- `varname`: Character string giving the variable name for which average effects are to be calculated.
- `data`: Data frame used to fit `object`.
- `change`: A string indicating the difference in predictor values to calculate the discrete change. `sd` gives plus and minus one-half standard deviation change around the median and `unit` gives a plus and minus one-half unit change around the median.
- `R`: Number of simulations to perform.

**Details**

The function calculates the average change in predicted probability for a discrete change in a single covariate with all other variables at their observed values, for objects of class `glm`. This function works with polynomials specified with the `poly` function.

**Value**

- `res`: A vector of values giving the average and 95 percent confidence bounds.
- `ames`: The average change in predicted probability (across all N observations) for each of the R simulations.
- `avesamp`: The average change in predicted probability for each of the N observation (across all of the R simulations).

**Author(s)**

Dave Armstrong
Examples

```r
data(france)
left.mod <- glm(voteleft ~ male + age + retnat +
poly(lrself, 2), data=france, family=binomial)
glmChange2(left.mod, "age", data=france, "sd")
```

---

**inspect**  
*Inspect a Variable in a Data Frame*

**Description**

Shows the variable label, factor levels (i.e., value labels) and frequency distribution for the specified variable.

**Usage**

```r
inspect(data, x, ...)
```

**Arguments**

- `data`  
  A data frame of class `data.frame` or `tbl_df`.

- `x`  
  A string identifying the name of the variable to be inspected.

- `...`  
  Other arguments to be passed down, currently unimplemented.

**Value**

A list with a variable label (if present), factor levels/value labels (if present) and a frequency distribution.

**Author(s)**

Dave Armstrong

**Examples**

```r
data(france)
inspect(france, "vote")
```
intEff

*Functions for Estimating Interaction Effects in Logit and Probit Models*

**Description**


**Usage**

```r
intEff(obj, vars, data)
```

**Arguments**

- `obj` A binary logit or probit model estimated with `glm`.
- `vars` A vector of the two variables involved in the interaction.
- `data` A data frame used in the call to `obj`.

**Value**

A list is returned with two elements - `byobs` and `atmean`. The `byobs` result gives the interaction effect evaluated at each observation. The `atmean` element has the marginal effect evaluated at the mean. Each eleement contains an element `int` which is a data frame with the following variable:

- `int_eff` The correctly calculated marginal effect.
- `linear` The incorrectly calculated marginal effect following the linear model analogy.
- `phat` Predicted Pr(Y=1|X).
- `se_int_eff` Standard error of `int_eff`.
- `zstat` The interaction effect divided by its standard error.

The `X` element of each returned result is the `X`-matrix used to generate the result.

**Author(s)**

Dave Armstrong

**References**


Examples

data(france)
mod <- glm(voteleft ~ age*lrself + retnat + male, data=france, family=binomial)
out <- intEff(obj=mod, vars=c("age", "lrself"), data=france)
out <- out$byobs$int
plot(out$phat, out$int_eff, xlab="Predicted Pr(Y=1|X)",
ylab = "Interaction Effect")
ag <- aggregate(out$linear, list(out$phat), mean)
lines(ag[,1], ag[,2], lty=2, col="red", lwd=2)
legend("topright", c("Correct Marginal Effect", "Linear Marginal Effect"),
pch=c(1, NA), lty=c(NA, 2), col=c("black", "red"), lwd=c(NA, 2), inset=.01)

---

InteractionEx   Example Data for DAintfun

Description

Data to execute example code for DAintfun

Format

A data frame with 500 observations on the following 4 variables.

- **y** a numeric vector
- **x1** a numeric vector
- **x2** a numeric vector
- **z** a numeric vector

Details

These are randomly generated data to highlight the functionality of DAintfun

---

intQualQuant   Predictions for Factor-Numeric Interactions in Linear Models

Description

This function works on linear models with a single interaction between a continuous (numeric) variable and a factor. The output is a data frame that gives the predicted effect of moving from each category to each other category of the factor over the range of values of the continuous conditioning variable.
**Usage**

```r
intQualQuant(
  obj,
  vars,
  level = 0.95,
  varcov = NULL,
  labs = NULL,
  n = 10,
  onlySig = FALSE,
  type = c("facs", "slopes"),
  plot = TRUE,
  vals = NULL,
  rug = TRUE,
  ci = TRUE,
  digits = 3,
  ...
)
```

**Arguments**

- `obj` An object of class `lm`.
- `vars` A vector of two variable names giving the two quantitative variables involved in the interaction. These variables must be involved in one, and only one, interaction.
- `level` Confidence level desired for lower and upper bounds of confidence interval.
- `varcov` A potentially clustered or robust variance-covariance matrix of parameters used to calculate standard errors. If `NULL`, the `vcov` function will be used.
- `labs` An optional vector of labels that will be used to identify the effects, if `NULL`, the factor levels will be used.
- `n` Number of values of the conditioning variable to use.
- `onlySig` Logical indicating whether only contrasts with significant differences should be returned. Significance is determined to exist if the largest lower bound is greater than zero or the smallest upper bound is smaller than zero.
- `type` String indicating whether the conditional partial effect of the factors is plotted (if `"facs"`), or the conditional partial effect of the quantitative variable (if `"slopes"`) is produced.
- `plot` Logical indicating whether graphical results (if `TRUE`) or numerical results (if `FALSE`) are produced.
- `vals` A vector of values at which the continuous variable will be held constant. If `NULL`, a sequence of length `n` across the variable’s range will be used.
- `rug` Logical indicating whether rug plots should be plotted in the panels.
- `ci` Logical indicating whether confidence bounds should be drawn.
- `digits` Number indicating how many decimal places to round the numeric output.
- `...` Other arguments to be passed down to `effect` if `plot.type = 'slopes'`. 
Value

For type = ‘facs’ and plot = FALSE, a data frame with the following values:

- **fit**: The expected difference between the two factor levels at the specified value of the conditioning variable.
- **se.fit**: The standard error of the expected differences.
- **x**: The value of the continuous conditioning variable.
- **contrast**: A factor giving the two values of the factor being evaluated.
- **lower**: The lower 95% confidence interval for fit.
- **upper**: The upper 95% confidence interval for fit.

For type = ‘facs’ and plot = TRUE, a lattice display is returned.

For type = ‘slopes’ and plot = FALSE, a character matrix with the following columns:

- **B**: The conditional effect of the quantitative variable for each level of the factor.
- **SE(B)**: The standard error of the conditional effect.
- **t-stat**: The t-statistic of the conditional effect.
- **Pr(>|t|)**: The two-sided p-value.

For type = ‘slopes’ and plot = TRUE, a lattice display is returned.

Author(s)

Dave Armstrong

Examples

data(Prestige, package="carData")
Prestige$income <- Prestige$income/1000
mod <- lm(prestige ~ income * type + education, data=Prestige)
intQualQuant(mod, c("income", "type"), n=10,
plot.type="none")
intQualQuant(mod, c("income", "type"), n=10,
plot.type="facs")
intQualQuant(mod, c("income", "type"), n=10,
plot.type="slopes")
**loessDeriv**

*Estimate Derivatives of LOESS Curve.*

**Description**

Estimates the first derivatives of the LOESS curve.

**Usage**

loessDeriv(obj, delta = 1e-05)

**Arguments**

- **obj**: An object of class `loess`.
- **delta**: Small change to be induced to estimate derivative.

**Value**

A vector of first derivative values evaluated at each original x-value.

**Author(s)**

Dave Armstrong

---

**logit_cc**

*Functions for Estimating Interaction Effects in Logit and Probit Models*

**Description**

Norton and Ai (2003) and Norton, Wang and Ai (2004) discuss methods for calculating the appropriate marginal effects for interactions in binary logit/probit models. These functions are direct translations of the Norton, Wang and Ai (2004) Stata code. These functions are not intended to be called by the user directly, rather they are called as needed by `intEff`.

**Usage**

logit_cc(obj = obj, int.var = int.var, vars = vars, b = b, X = X)

**Arguments**

- **obj**: A binary logit or probit model estimated with `glm`.
- **int.var**: The name of the interaction variable.
- **vars**: A vector of the two variables involved in the interaction.
- **b**: Coefficients from the `glm` object.
- **X**: Model matrix from the `glm` object.
Value
A data frame with the following variable:

- **int_eff**: The correctly calculated marginal effect.
- **linear**: The incorrectly calculated marginal effect following the linear model analogy.
- **phat**: Predicted Pr(Y=1|X).
- **se_int_eff**: Standard error of int_eff.
- **zstat**: The interaction effect divided by its standard error.

Author(s)
Dave Armstrong

References


---

**makeHypSurv**  
*Make Hypothetical Predictions for Survey Data*

Description
Calculates survival probabilities for hypothetical data.

Usage
```
makeHypSurv(1, obj, ...)
```

Arguments
- **1**: A named list where variable names are the names and values of the variables are the values. Combinations will be made with `expand.grid`.
- **obj**: A model object estimated with `survreg`.
- **...**: currently not implemented.

Value
A data frame.
mnlAveEffPlot

Author(s)
Dave Armstrong

---

mnlAveEffPlot  
*Average Effects Plot for Multinomial Logistic Regression*

Description

Produces a plot of average effects for one variable while holding the others constant at observed values.

Usage

```r
mnlAveEffPlot(obj, varname, data, R = 1500, nvals = 25, plot = TRUE, ...)
```

Arguments

- `obj`  
  An object of class `multinom`.
- `varname`  
  A string indicating the variable for which the plot is desired.
- `data`  
  The data used to estimate `obj`.
- `R`  
  Number of simulations used to generate confidence bounds.
- `nvals`  
  Number of evaluation points for the predicted probabilities.
- `plot`  
  Logical indicating whether a plot should be produced (if `TRUE`) or numerical results should be returned (if `FALSE`).
- `...`  
  Other arguments to be passed down to `xyplot`.

Value

Either a plot or a data frame with variables

- `mean`  
  The average effect (i.e., predicted probability)
- `lower`  
  The lower 95% confidence bound
- `upper`  
  The upper 95% confidence bound
- `y`  
  The values of the dependent variable being predicted
- `x`  
  The values of the independent variable being manipulated

Author(s)
Dave Armstrong

References

Examples

```r
library(nnet)
data(france)
mnl.mod <- multinom(vote ~ age + male + retnat + lrself, data=france)
## Not run: mnlAveEffPlot(mnl.mod, "lrself", data=france)
```

---

### mnlChange

**Maximal First Differences for Multinomial Logistic Regression Models**

**Description**

For objects of class `multinom`, it calculates the change in predicted probabilities, for maximal discrete changes in all covariates holding all other variables constant at typical values.

**Usage**

```r
mnlChange(
  obj, 
  data, 
  typical.dat = NULL, 
  diffchange = c("range", "sd", "unit"), 
  sim = TRUE, 
  R = 1500 
)
```

**Arguments**

- **obj** A model object of class `multinom`.
- **data** Data frame used to fit object.
- **typical.dat** Data frame with a single row containing values at which to hold variables constant when calculating first differences. These values will be passed to `predict`, so factors must take on a single value, but have all possible levels as their levels attribute.
- **diffchange** A string indicating the difference in predictor values to calculate the discrete change. `range` gives the difference between the minimum and maximum, `sd` gives plus and minus one-half standard deviation change around the median and `unit` gives a plus and minus one-half unit change around the median.
- **sim** Logical indicating whether simulated confidence bounds should be produced.
- **R** Number of simulations to perform if `sim = TRUE`
Details

The function calculates the changes in predicted probabilities for maximal discrete changes in the covariates for objects of class `multinom`. This function works with polynomials specified with the `poly` function. It also works with multiplicative interactions of the covariates by virtue of the fact that it holds all other variables at typical values. By default, typical values are the median for quantitative variables and the mode for factors. The way the function works with factors is a bit different. The function identifies the two most different levels of the factor and calculates the change in predictions for a change from the level with the smallest prediction to the level with the largest prediction.

Value

A list with the following elements:

- `diffs`: A matrix of calculated first differences
- `minmax`: A matrix of values that were used to calculate the predicted changes
- `minPred`: A matrix of predicted probabilities when each variable is held at its minimum value, in turn.
- `maxPred`: A matrix of predicted probabilities when each variable is held at its maximum value, in turn.

Author(s)

Dave Armstrong

Examples

```r
library(nnet)
data(france)
mnl.mod <- multinom(vote ~ age + male + retnat + lrself, data=france)
typical.france <- data.frame(
  age = 35,
  retnat = factor(1, levels=1:3, labels=levels(france$retnat)),
  stringsAsFactors=TRUE)
mnlChange(mnl.mod, data=france, typical.dat=typical.france)
```

Description

Calculates average effects of a variable in multinomial logistic regression holding all other variables at observed values.
Usage

```r
mnlChange2(obj, varnames, data, diffchange = c("unit", "sd"), R = 1500)
```

Arguments

- **obj**: An object of class `multinom`
- **varnames**: A string identifying the variable to be manipulated.
- **data**: Data frame used to fit object.
- **diffchange**: A string indicating the difference in predictor values to calculate the discrete change. `sd` gives plus and minus one-half standard deviation change around the median and `unit` gives a plus and minus one-half unit change around the median.
- **R**: Number of simulations.

Value

A list with elements:

- **mean**: Average effect of the variable for each category of the dependent variable.
- **lower**: Lower 95 percent confidence bound
- **upper**: Upper 95 percent confidence bound

Author(s)

Dave Armstrong

Examples

```r
library(nnet)
data(france)
mnl.mod <- multinom(vote ~ age + male + retnat + lrself, data=france)
mnlChange2(mnl.mod, "lrself", data=france, )
```

---

**mnlfit**

*Fit Statistics and Specification Test for Multinomial Logistic Regression*

Description

Provides fit statistics (pseudo R-squared values) and the Fagerland, Hosmer and Bonfi (2008) specification test for Multinomial Logistic Regression models.
Usage

mnlfit(obj, permute = FALSE)

Arguments

obj An object of class multinom
permute Logical indicating whether to check all base categories for the Fagerland et. al. specification test.

Value

A list with elements:

result Fit statistics.
permres The results of the base category permutation exercise.

Author(s)

Dave Armstrong

References


Examples

library(nnet)
data(france)
mnl.mod <- multinom(vote ~ age + male + retnat + lrself, data=france)
mnlfit(mnl.mod)

mnlSig Print Statistically Significant MNL Coefficients

Description

By default, the summary for objects of class multinom is not particularly helpful. It still requires a lot of work on the part of the user to figure out which coefficients are significantly different from zero and which ones are not. mnlSig solves this problem by either flagging significant coefficients with an asterisk or only printing significant coefficients, leaving insignificant ones blank.
Usage

mnlSig(
obj,
pval = 0.05,
two.sided = TRUE,
flag.sig = TRUE,
insig.blank = FALSE
)

Arguments

obj A model object of class multinom.
pval The desired Type I error rate to identify coefficients as statistically significant.
two.sided Logical indicating whether calculated p-values should be two-sided (if TRUE) or one-sided (if FALSE).
flag.sig Logical indicating whether an asterisk should be placed beside coefficients which are significant at the pval level.
insig.blank Logical indicating whether coefficients which are not significant at the pval level should be blank in the output.

Value

A data frame suitable for printing with the (optionally significance-flagged) coefficients from a multinomial logit model.

Author(s)

Dave Armstrong

Examples

library(nnet)
data(france)
mnl.mod <- multinom(vote ~ retnat + male + retnat + lrself, data=france)
mnlSig(mnl.mod)

NKnots

AIC and BIC selection of number of spline knots

Description

Calculates AIC and BIC for the selection of knots in a spline over values (potentially including polynomials) up to a user-defined maximum.
**Usage**

```r
NKnots(
  form,
  var,
  data,
  degree = 3,
  min.knots = 1,
  max.knots = 10,
  includePoly = FALSE,
  plot = FALSE,
  criterion = c("AIC", "BIC", "CV"),
  cvk = 10,
  cviter = 10
)
```

**Arguments**

- **form**: A formula detailing the model for which smoothing is to be evaluated.
- **var**: A character string identifying the variable for which smoothing is to be evaluated.
- **data**: Data frame providing values of all variables in `form`.
- **degree**: Degree of polynomial in B-spline basis functions.
- **min.knots**: Minimum number of internal B-spline knots to be evaluated.
- **max.knots**: Maximum number of internal B-spline knots to be evaluated.
- **includePoly**: Include linear and polynomial models up to, and including `degree`-th order polynomials.
- **plot**: Logical indicating whether a plot should be returned.
- **criterion**: Statistical criterion to minimize in order to find the best number of knots - AIC, BIC or Cross-validation.
- **cvk**: Number of groups for cross-validation.
- **cviter**: Number of iterations of cross-validation to average over. 10 is the default but in real-world applications, this should be somewhere around 200.

**Value**

A plot, if `plot=TRUE`, otherwise a data frame with the degrees of freedom and corresponding fit measure.

**Author(s)**

Dave Armstrong
Examples

data(Prestige, package="carData")
NKnots(prestige ~ education + type, var="income", data=na.omit(Prestige), plot=FALSE)

NKnotsTest

Test of functional form assumption using B-splines

Description

Estimate hypothesis test of lower- and higher-order non-linear relationships against an assumed target relationship.

Usage

NKnotsTest(
  form,
  var,
  data,
  targetdf = 1,
  degree = 3,
  min.knots = 1,
  max.knots = 10,
  adjust = "none"
)

Arguments

form A formula detailing the model for which smoothing is to be evaluated.
var A character string identifying the variable for which smoothing is to be evaluated.
data Data frame providing values of all variables in form.
targetdf The assumed degrees of freedom against which the tests will be conducted.
degree Degree of polynomial in B-spline basis functions.
min.knots Minimum number of internal B-spline knots to be evaluated.
max.knots Maximum number of internal B-spline knots to be evaluated.
adjust Method by which p-values will be adjusted (see p.adjust)

Value

A matrix with the following columns:

F F statistics of test of candidate models against target model
DF1 Numerator DF from F-test
oc2plot

DF2  Denominator DF from F-test
p(F)  p-value from the F-test
Clarke  Test statistic from the Clarke test
Pr(Better)  The Clarke statistic divided by the number of observations
p(Clarke)  p-value from the Clarke test. (T) means that the significant p-value is in favor of the Target model and (C) means the significant p-value is in favor of the candidate (alternative) model.
Delta_AIC  AIC(candidate model) - AIC(target model)
Delta_AICc  AICc(candidate model) - AICc(target model)
Delta_BIC  BIC(candidate model) - BIC(target model)

Author(s)
Dave Armstrong

Examples

data(Prestige, package="carData")
NKnotsTest(prestige ~ education + type, var="income", data=na.omit(Prestige), targetdf=3)

oc2plot  Plot First Differences from Ordinal DV Model

Description
Takes the output from ordChange and turns it into a plot.

Usage
oc2plot(ordc, plot = TRUE)

Arguments
ordc  The output from ordChange.
plot  Logical indicating whether a plot (if TRUE) or data (if FALSE) should be returned.

Value
Either a lattice plot or a data.frame depending on the specification of the plot argument.

Author(s)
Dave Armstrong
Examples

```r
library(MASS)
data(france)
polr.mod <- polr(vote ~ age + male + retnat + lrself, data=france)
typical.france <- data.frame(
  age = 35,
  retnat = factor(1, levels=1:3, labels=levels(france$retnat)),
stringsAsFactors=TRUE)
oc.res <- ordChange(polr.mod, data=france, typical.dat=typical.france, sim=TRUE)
oc2plot(oc.res)
```

---

**ordAveEffPlot**

*Plot Average Effects of Variables in Proportional Odds Logistic Regression*

Description

For objects of class `polr` the function plots the average effect of a single variable holding all other variables at their observed values.

Usage

```r
ordAveEffPlot(
  obj,
  varname,
  data,
  R = 1500,
  nvals = 25,
  plot = TRUE,
  returnInd = FALSE,
  returnMprob = FALSE,
  ...
)
```

Arguments

- **obj**: An object of class `polr`
- **varname**: A string providing the name of the variable for which you want the plot to be drawn.
- **data**: Data used to estimate `obj`.
- **R**: Number of simulations to generate confidence intervals.
- **nvals**: Number of evaluation points of the function.
- **plot**: Logical indicating whether or not the result should be plotted (if TRUE) or returned to the console (if FALSE).
- **returnInd**: Logical indicating whether average individual probabilities should be returned.
ordAveEffPlot

returnMprob Logical indicating whether marginal probabilities, averaged over individuals, should be returned.

... Arguments passed down to the call to xyplot

Details

Following the advice of Hanmer and Kalkan (2013) the function calculates the average effect of a variable holding all other variables at observed values and then plots the result.

Value

Either a plot or a list with a data frame containing the variables

mean The average effect (i.e., predicted probability)
lower The lower 95% confidence bound
upper The upper 95% confidence bound
y The values of the dependent variable being predicted
x The values of the independent variable being manipulated

and the elements Ind or Mprob, as requested.

Author(s)

Dave Armstrong

References


Examples

library(MASS)
data(france)
polr.mod <- polr(vote ~ age + male + retnat + lrself, data=france)
## Not run: ordAveEffPlot(polr.mod, "lrself", data=france)
ordChange

Maximal First Differences for Proportional Odds Logistic Regression Models

Description

For objects of class polr, it calculates the change in predicted probabilities, for maximal discrete changes in all covariates holding all other variables constant at typical values.

Usage

ordChange(
  obj,
  data,
  typical.dat = NULL,
  diffchange = c("range", "sd", "unit"),
  sim = TRUE,
  R = 1500
)

Arguments

  obj         A model object of class polr.
  data        Data frame used to fit object.
  typical.dat Data frame with a single row containing values at which to hold variables constant when calculating first differences. These values will be passed to predict, so factors must take on a single value, but have all possible levels as their levels attribute.
  diffchange  A string indicating the difference in predictor values to calculate the discrete change. range gives the difference between the minimum and maximum, sd gives plus and minus one-half standard deviation change around the median and unit gives a plus and minus one-half unit change around the median.
  sim         Logical indicating whether or not simulations should be done to generate confidence intervals for the difference.
  R           Number of simulations.

Details

The function calculates the changes in predicted probabilities for maximal discrete changes in the covariates for objects of class polr. This function works with polynomials specified with the poly function. It also works with multiplicative interactions of the covariates by virtue of the fact that it holds all other variables at typical values. By default, typical values are the median for quantitative variables and the mode for factors. The way the function works with factors is a bit different. The function identifies the two most different levels of the factor and calculates the change in predictions for a change from the level with the smallest prediction to the level with the largest prediction.
Value

A list with the following elements:

- **diffs**: A matrix of calculated first differences
- **minmax**: A matrix of values that were used to calculate the predicted changes
- **minPred**: A matrix of predicted probabilities when each variable is held at its minimum value, in turn.
- **maxPred**: A matrix of predicted probabilities when each variable is held at its maximum value, in turn.

Author(s)

Dave Armstrong

Examples

```r
library(MASS)
data(france)
polr.mod <- polr(vote ~ age + male + retnat + lrself, data=france)
typical.france <- data.frame(
  age = 35,
  retnat = factor(1, levels=1:3, labels=levels(france$retnat)),
  stringsAsFactors=TRUE)
ordChange(polr.mod, data=france, typical.dat=typical.france, sim=FALSE)
```

Description

For objects of class `polr`, it calculates the average change in predicted probabilities, for discrete changes in a covariate holding all other variables at their observed values.

Usage

```r
ordChange2(obj, varnames, data, diffchange = c("sd", "unit"), R = 1500)
```

Arguments

- **obj**: A model object of class `polr`.
- **varnames**: A vector of strings identifying the variable to be manipulated.
- **data**: Data frame used to fit object.
- **diffchange**: A string indicating the difference in predictor values to calculate the discrete change. `sd` gives plus and minus one-half standard deviation change around the median and `unit` gives a plus and minus one-half unit change around the median.
- **R**: Number of simulations.
Details

The function calculates the changes in predicted probabilities for maximal discrete changes in the covariates for objects of class `polr`. This function works with polynomials specified with the `poly` function. It also works with multiplicative interactions of the covariates by virtue of the fact that it holds all other variables at typical values. By default, typical values are the median for quantitative variables and the mode for factors. The way the function works with factors is a bit different. The function identifies the two most different levels of the factor and calculates the change in predictions for a change from the level with the smallest prediction to the level with the largest prediction.

Value

A list with the following elements:
- `diffs` A matrix of calculated first differences
- `minmax` A matrix of values that were used to calculate the predicted changes
- `minPred` A matrix of predicted probabilities when each variable is held at its minimum value, in turn.
- `maxPred` A matrix of predicted probabilities when each variable is held at its maximum value, in turn.

Author(s)

Dave Armstrong

Examples

```r
library(MASS)
data(france)
polr.mod <- polr(vote ~ age + male + retnat + lrself, data=france)
typical.france <- data.frame(
  age = 35,
  retnat = factor(1, levels=1:3, labels=levels(france$retnat)),
  stringsAsFactors=TRUE)
ordChange2(polr.mod, "age", data=france, diffchange="sd")
```

ordfit

Fit Statistics for Proportional Odds Logistic Regression Models

Description

For objects of class `polr`, it calculates a number of fit statistics and specification tests.

Usage

`ordfit(obj)"
outEff

Arguments

   obj  A model object of class polr.

Value

   An object of class ordfit which is a matrix containing statistics and specification tests.

Author(s)

   Dave Armstrong

References


Examples

   library(MASS)
   data(france)
   polr.mod <- polr(vote ~ age + male + retnat + lrself, data=france)
   ordfit(polr.mod)

outEff  

Plot Effects of Removing Outliers

Description

   Plots the effect of a variable sequentially removing outlying observations.

Usage

   outEff(
       obj,  
       var,  
       data,  
       stat = c("cooksD", "hat", "deviance", "pearson"),  
       nOut = 10,  
       whichOut = NULL,  
       cumulative = FALSE  
   )
Arguments

- **obj**: An object of class `glm` that will be used to plot the outlier removed lines.
- **var**: A character string giving the name of the variable to be used.
- **data**: A data frame.
- **stat**: Which statistic to use to evaluate ‘outlyingness’.
- **nOut**: Number of outliers to be removed.
- **whichOut**: If not `NULL`, a vector of observation numbers to be removed manually, rather than using `stat`.
- **cumulative**: Logical indicating whether the outliers should be removed cumulatively, or each one in turn, replacing the other outlying observations.

Author(s)

Dave Armstrong

---

**outXT**

*Create LaTeX or CSV versions of an Object Produced by CrossTable*

Description

`outXT` takes the output from `CrossTable` in the `gmodels` package and produces either LaTeX code or CSV file that can be imported into word processing software.

Usage

```r
outXT(
  obj,
  count = TRUE,
  prop.r = TRUE,
  prop.c = TRUE,
  prop.t = TRUE,
  col.marg = TRUE,
  row.marg = TRUE,
  digits = 3,
  type = "word",
  file = NULL
)
```

Arguments

- **obj**: A list returned by `CrossTable` from the `gmodels` package.
- **count**: Logical indicating whether the cell frequencies should be returned.
- **prop.r**: Logical indicating whether the row proportions should be returned.
- **prop.c**: Logical indicating whether the column proportions should be returned.
panel.2cat

<table>
<thead>
<tr>
<th>prop.t</th>
<th>Logical indicating whether the cell proportions should be returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>col.marg</td>
<td>Logical indicating whether the column marginals should be printed.</td>
</tr>
<tr>
<td>row.marg</td>
<td>Logical indicating whether the row marginals should be printed.</td>
</tr>
<tr>
<td>digits</td>
<td>Number of digits to use in printing the proportions.</td>
</tr>
<tr>
<td>type</td>
<td>String where \texttt{word} indicates a CSV file will be produced and \texttt{latex} indicates LaTeX code will be generated.</td>
</tr>
<tr>
<td>file</td>
<td>Connection where the file will be written, if \texttt{NULL} the output will only be written to the console</td>
</tr>
</tbody>
</table>

**Value**

A file containing LaTeX Code or CSV data to make a table

**Author(s)**

Dave Armstrong

---

**panel.2cat**

\textit{Lattice panel function for confidence intervals with capped bars}

**Description**

This panel function is defined to plot confidence intervals in a multi-panel lattice display where the x-variable is categorical. Note, both lower and upper must be passed directly to \texttt{xyplot} as they will be passed down to the panel function.

**Usage**

\texttt{panel.2cat(x, y, subscripts, lower, upper, length = 0.2)}

**Arguments**

- \texttt{x, y} Data from the call to \texttt{xyplot}.
- \texttt{subscripts} Variable used to create the juxtaposed panels.
- \texttt{lower, upper} 95\% lower and upper bounds of \texttt{y}.
- \texttt{length} Length of the arrow head lines.

**Author(s)**

Dave Armstrong
Examples

```r
library(lattice)
library(effects)
data(Duncan, package="carData")
Duncan$inc.cat <- cut(Duncan$income, 3)
mod <- lm(prestige ~ inc.cat * type + education, 
  data=Duncan)
e1 <- effect("inc.cat*type", mod)
update(plot(e1), panel=panel.2cat)
```

---

**panel.ci**  
Lattice panel function for confidence intervals

**Description**

This panel function is defined to plot confidence intervals in a multi-panel lattice display. Note, both lower and upper must be passed directly to `xyplot` as they will be passed down to the `prepanel` function.

**Usage**

```
panel.ci(x, y, subscripts, lower, upper, zl)
```

**Arguments**

- **x, y**  
  Data from the call to `xyplot`.
- **subscripts**  
  Variable used to created the juxtaposed panels.
- **lower, upper**  
  95% lower and upper bounds of `y`.
- **zl**  
  Logical indicating whether or not a horizontal dotted line at zero is desired.

**Author(s)**

Dave Armstrong
panel.doublerug

Description

This panel function is defined to plot two rugs, one on top of the other in a multi-panel lattice display.

Usage

panel.doublerug(
  xa = NULL,
  xb = NULL,
  regular = TRUE,
  start = if (regular) 0 else 0.97,
  end = if (regular) 0.03 else 1,
  x.units = rep("npc", 2),
  lty = 1,
  lwd = 1
)

Arguments

xa, xb Numerical vectors to be plotted.
regular Logical flag indicating whether rug is to be drawn on the usual side (bottom/left) as opposed to the other side (top/right).
start, end Start and end points for the rug ticks on the y-axis.
x.units Character vectors, replicated to be of length two. Specifies the (grid) units associated with start and end above. x.units are for the rug on the x-axis and y-axis respectively (and thus are associated with start and end values on the y and x scales respectively). See panel.rug for more details.
lty, lwd Line type and width arguments (see par for more details).

Author(s)

Dave Armstrong
panel.transci

*Lattice panel function for translucent confidence intervals*

**Description**

This panel function is defined to plot translucent confidence intervals in a single-panel, grouped (i.e., superposed) lattice display. Note, both lower and upper must be passed directly to `xyplot` as they will be passed down to the panel function.

**Usage**

```r
panel.transci(x, y, groups, lower, upper, ca = 0.25, ...)
```

**Arguments**

- `x, y` Data from the call to `xyplot`.
- `groups` Variable used to create the superposed panels.
- `lower, upper` 95% lower and upper bounds of `y`.
- `ca` Value of the alpha channel in [0,1]
- `...` Other arguments to be passed down to the plotting functions.

**Author(s)**

Dave Armstrong

---

pgumbel

*PDF of the Gumbel Distribution*

**Description**

Returns the PDF of the Gumbel distribution.

**Usage**

```r
pgumbel(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
```

**Arguments**

- `q` Vector of quantiles
- `location` The location parameter, \( \mu \). This is not the mean of the Gumbel distribution.
- `scale` The scale parameter \( \sigma \).
- `lower.tail` Logical, whether lower, if `TRUE` or upper, if `FALSE`, tail probabilities should be returned.
- `log.p` Logical, if `TRUE` probabilities are given in their log.
Details

This code and the details of the help file were taken from the VGAM package.

The Gumbel distribution is a special case of the generalized extreme value (GEV) distribution where the shape parameter $\xi = 0$. The latter has 3 parameters, so the Gumbel distribution has two. The Gumbel distribution function is

$$G(y) = \exp \left( - \exp \left[ \frac{y - \mu}{\sigma} \right] \right)$$

where $-\infty < y < \infty$, $-\infty < \mu < \infty$ and $\sigma > 0$. Its mean is

$$\mu - \sigma \gamma$$

and its variance is

$$\sigma^2 \pi^2 / 6$$

where $\gamma$ is Euler’s constant (which can be obtained as -digamma(1)).

Value

A vector of probabilities

Author(s)

Thomas Yee

Plot Results from BALSOS

Description

Plots the optimally scaled points with posterior 95% credible intervals.

Usage

```r
## S3 method for class 'balsos'
plot(x, ..., freq = TRUE, offset = 0.1)
```

Arguments

- `x` Object of class balsos.
- `...` Other arguments to be passed down, currently not implement and may conflict with the lattice figure. To change the figure the best advice would be to save the plot as an oject and use the update function to change its features.
- `freq` Logical indicating whether you want the frequentist result plotted alongside the Bayesian result.
- `offset` If `freq=TRUE`, the Bayesian points will be plotted at `x-offset` and the frequentist points will be plotted at `x+offset.`
Value

A lattice graph produce by a call to xyplot.

Author(s)

Dave Armstrong

Description

Plots the loess curve of the fitted values against a focal x-variable.

Usage

```r
## S3 method for class 'loess'
plot(
  x,
  ...
  ci = TRUE,
  level = 0.95,
  linear = FALSE,
  addPoints = FALSE,
  col.alpha = 0.5
)
```

Arguments

- `x` An object of class `loess`.
- `...` Other arguments to be passed down to `xyplot`.
- `ci` Logical indicating whether point-wise confidence intervals should be included around the fitted curve.
- `level` The confidence level of the confidence intervals
- `linear` Logical indicating whether the OLS line should also be included.
- `addPoints` Logical indicating whether or not points should be added to the figure identifying the postion of individual observations.
- `col.alpha` Value for alpha channel of the RGB color palette.

Details

Plots the fitted loess curve potentially with point-wise confidence bounds.

Value

A plot.
plot.secdiff

Author(s)
Dave Armstrong

Description
Plots the results of the secondDiff function.

Usage
## S3 method for class 'secdiff'
plot(x, level = 0.95, ...)

Arguments
x An object of class secdiff
level The confidence level of the confidence interval(s)
... Other arguments to be passed down, currently not implemented.

poisfit
Scalar Measures of Fit for Poisson GLMs Models

Description
Calculates scalar measures of fit for models with count dependent variables along the lines described in Long (1997) and Long and Freese (2005).

Usage
poisfit(obj)

Arguments
obj A model of class glm with family=poisson.

Details
 poisfit calculates scalar measures of fit (many of which are pseudo-R-squared measures) to describe how well a model fits data with a count dependent variable.

Value
A named vector of scalar measures of fit
Author(s)
Dave Armstrong

References

poisGOF

Deviance and Chi-squared Goodness-of-Fit Test for Poisson Models

Description
Deviance and Chi-squared goodness-of-fit test of the null hypothesis that poisson variance is appropriate to model the conditional dispersion of the data, given a particular model.

Usage
poisGOF(obj)

Arguments
obj A model object of class glm (with family=poisson).

Value
A 2x2 data frame with rows representing the different types of statistics (Deviance and Chi-squared) and columns representing the test statistic and p-value.

Examples
## Example taken from MASS help file for glm, identified to be
## Dobson (1990) Page 93: Randomized Controlled Trial:
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts, stringsAsFactors=TRUE))

## Example taken from MASS help file for glm, identified to be
## Dobson (1990) Page 93: Randomized Controlled Trial:
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts, stringsAsFactors=TRUE))
```r
glm.D93 <- glm(counts ~ outcome + treatment, family=poisson())
poisGOF(glm.D93)
```

---

**powerTrans**  
*Power Transformation Function*

**Description**

Power transformation function that treats everything with absolute power transform < .01 as the log transform.

**Usage**

```r
powerTrans(x, transPower)
```

**Arguments**

- `x` Vector of values to be transformed
- `transPower` The power of the transformation

**Value**

A vector of transformed values

---

**pre**  
*Proportional and Expected Proportional Reductions in Error*

**Description**

Calculates proportional reduction in error (PRE) and expected proportional reduction in error (epre) from Herron (1999).

**Usage**

```r
pre(mod1, mod2 = NULL, sim = FALSE, R = 2500)
```

**Arguments**

- `mod1` A model of class `glm` (with family `binomial`), `polr` or `multinom` for which (e)PRE will be calculated.
- `mod2` A model of the same class as `mod1` against which proportional reduction in error will be measured. If `NULL`, the null model will be used.
- `sim` A logical argument indicating whether a parametric bootstrap should be used to calculate confidence bounds for (e)PRE. See `Details` for more information.
- `R` Number of bootstrap samples to be drawn if `sim=TRUE`. 
Details

Proportional reduction in error is calculated as a function of correct and incorrect predictions (and the probabilities of correct and incorrect predictions for ePRE). When sim=TRUE, a parametric bootstrap will be used that draws from the multivariate normal distribution centered at the coefficient estimates from the model and using the estimated variance-covariance matrix of the estimators as Sigma. This matrix is used to form R versions of XB and predictions are made for each of the R different versions of XB. Confidence intervals can then be created from the bootstrap sampled (e)PRE values.

Value

An object of class pre, which is a list with the following elements:

- pre: The proportional reduction in error
- epre: The expected proportional reduction in error
- m1form: The formula for model 1
- m2form: The formula for model 2
- pcp: The percent correctly predicted by model 1
- pmc: The percent correctly predicted by model 2
- epcp: The expected percent correctly predicted by model 1
- epmc: The expected percent correctly predicted by model 2
- pre.sim: A vector of bootstrapped PRE values if sim=TRUE
- epre.sim: A vector of bootstrapped ePRE values if sim=TRUE

Author(s)

Dave Armstrong

References


Examples

data(france)
left.mod <- glm(voteleft ~ male + age + retnat + poly(lrself, 2), data=france, family=binomial)
pre(left.mod)
prepanel.ci

Lattice prepanel function for confidence intervals

Description

This prepanel function is defined so as to allow room for all confidence intervals plotted in a lattice display. Note, both lower and upper must be passed directly to xyplot as they will be passed down to the prepanel function.

Usage

prepanel.ci(x, y, subscripts, lower, upper)

Arguments

x, y Data from the call to xyplot.
subscripts Variable used to created the juxtaposed panels.
lower, upper 95% lower and upper bounds of y.

Value

A list giving the ranges and differences in ranges of x and the lower and upper bounds of y.

Author(s)

Dave Armstrong

print.diffci

Print Confidence Intervals for Predicted Probabilities and Their Differences

Description

Print method for output from the probci function.

Usage

## S3 method for class 'diffci'
print(x, ..., digits = 4, filter = NULL, const = NULL, onlySig = FALSE)
Arguments

- **x**: A object of class `diffci` produced by `probci`.
- **...**: Other arguments to be passed down to print, currently unimplemented.
- **digits**: How many digits to round output.
- **filter**: A named list of values where the names indicate the variable to be filtered and the values in the vector indicate the values to include for the filtering variable.
- **const**: A string identifying the name of the variable to be held constant across comparisons.
- **onlySig**: Logical indicating whether all differences should be displayed or only those significant at the 95% two-tailed level.

Value

An data frame with the following variables:

- **variables**: The variables and the values at which they are held constant. For example, `tmp1` would be the first value of `tmp` used in the probability calculation and `tmp2` would be the second value of `tmp` used in the probability calculation.
- **pred_prob**: The difference in predicted probability given the following change in `X`: `tmp2-tmp1`.
- **lower, upper**: The lower and upper 95% confidence bounds.

Author(s)

Dave Armstrong

Examples

data(france)
left.mod <- glm(voteleft ~ male + age + retnat + poly(lrself, 2, raw=TRUE), data=france, family=binomial)
out <- probci(left.mod, france, numQuantVals=3, changeX=c("retnat", "lrself"))
print(out, filter=list(retnat=c("Better", "Worse")))
print(out, filter=list(retnat=c("Better", "Worse")), const="lrself")
print(out, const="retnat")
Description

Print method for object of class glmc2.

Usage

## S3 method for class 'glmc2'
print(x, ...)

Arguments

x Object of class glmc2
... Currently unimplemented.

print.iqq

Print method for intQualQuant objects.

Description

Print method for objects of class iqq calculated with the intQualQuant function.

Usage

## S3 method for class 'iqq'
print(x, ...)

Arguments

x Object of class iqq
... Currently unimplemented
print.ordChange  
*Print method for ordChange objects*

**Description**

Print methods for objects of class `ordChange`

**Usage**

```r
## S3 method for class 'ordChange'
print(x, ..., digits = 3)
```

**Arguments**

- `x`: Object of class `ordChange`
- `...`: Other arguments to be passed down to the function
- `digits`: Number of digits to print

print.pre  
*Print method for objects of class pre*

**Description**

Prints the output from an object of class `pre`. The function prints all components of the calculation and optionally simulated confidence bounds.

**Usage**

```r
## S3 method for class 'pre'
print(x, ..., sim.ci = 0.95)
```

**Arguments**

- `x`: An object of class `pre`.
- `...`: Other arguments passed to print, currently not implemented
- `sim.ci`: Coverage for the simulated confidence interval, if `sim=TRUE` in the call to `pre`.

**Author(s)**

Dave Armstrong

**See Also**

`pre`
Description

Calculates predicted probabilities for any combination of x-variable values holding all other variables constant at either typical values (average case approach) or at observed values (average effect approach).

Usage

probci(
  obj,  
  data, 
  .b = NULL, 
  .vcov = NULL, 
  changeX = NULL, 
  numQuantVals = 5, 
  xvals = NULL, 
  type = c("aveEff", "aveCase")
)

Arguments

obj  A model of class glm, particularly those with binomial family.
data  Data frame used to estimate obj.
.b  A vector of coefficients to be passed down to the simulation. If NULL, coef() will be used to obtain coefficients from obj.
.vcov  A parameter variance covariance matrix to be passed to the simulation. If NULL, vcov() will be used to obtain the variance-covariance matrix of the parameters.
changeX  A vector of strings giving the names of variables for which changes are desired.
numQuantVals  For quantitative variables, if no x-values are specified in xvals, then numQuantVals gives the number of values used across the range of the variable.
xvals  A named list of values used to make the predictions. The names in the list should correspond with the variable names specified in changeX.
type  Type of effect to be generated. aveEff produces the average first difference across all observed values of X, while aveCase gives the first difference holding all other variables constant at typical values.

Details

Calculates predicted probabilities for any combination of x-variable values holding all other variables constant at either typical values (average case approach) or at observed values (average effect approach). The function uses a parametric bootstrap to provide generate confidence bounds for predicted probabilities and their differences. The confidence intervals produced are raw percentile interviews (at the 5% level).
Value

An data frame with the following variables:

- **variables**: The variables and the values at which they are held constant. For example, \texttt{tmp1} would be the first value of \texttt{tmp} used in the probability calculation and \texttt{tmp2} would be the second value of \texttt{tmp} used in the probability calculation.
- **pred_prob**: The difference in predicted probability given the following change in \texttt{X}: \texttt{tmp2} - \texttt{tmp1}.
- **lower, upper**: The lower and upper 95\% confidence bounds.

Author(s)

Dave Armstrong

Examples

data(france)
left.mod <- glm(voteleft ~ male + age + retnat + poly(lrself, 2, raw=TRUE), data=france, family=binomial)
out <- probci(left.mod, france, changeX="retnat")
out
out2 <- probci(left.mod, france, changeX="lrself",
               xvals = list(lrself = c(1,10)))
out2
out3 <- probci(left.mod, france, changeX=c("lrself", "retnat"),
               xvals = list(lrself = c(1,10)))
out3

---

Probgroup

Plog Probabilities by Group

Description

Plots predicted probabilities by value of the dependent variable for proportional odds logistic regression and multinomial logistic regression models

Usage

\texttt{probgroup(obj, \ldots)}

Arguments

- **obj**: Object of class \texttt{polr} or \texttt{multinom} where appropraite.
- **\ldots**: Currently not implemented.
Details

Plots the predicted probabilities by value of the dependent variable. Each panel only includes the observations that had the identified value of the dependent variable.

Value

A plot.

Author(s)

Dave Armstrong

scaleDataFrame

Standardize quantitative variables in a data frame

Description

This function standardizes quantitative variables in a data frame while leaving the others untouched. This leaves not only factors, but also binary variables (those with values 0, 1, or NA).

Usage

scaleDataFrame(data)

Arguments

data A data frame.

Value

A data frame with standardized quantitative variables

Author(s)

Dave Armstrong
searchVarLabels  

**Search Variable Labels Attribute**

**Description**

Data imported from SPSS or Stata comes with the variable labels set (if they were set in the original dataset) as one of the dataframe’s attributes. This allows you to search the variable labels and returns the variable column number, name and label for all variables that have partially match the search term either in their labels or names.

**Usage**

```
searchVarLabels(dat, str)
```

**Arguments**

- `dat`  
  a data frame whose variable labels you want to search.

- `str`  
  string used to search variable labels.

**Details**

For an imported Stata dataset, variable labels are in the `var.labels` attribute of the dataset and in an SPSS dataset, they are in the `variable.labels` attribute. These are searched, ignoring case, for the desired string.

**Value**

- `matrix`  
  A matrix of dimensions n-matches x 2 is returned, where the first column is the column number of the matching variable and the second column is the variable label. The row names of the matrix are the variable names.

**Author(s)**

Dave Armstrong

---

secondDiff  

**Calculate Cross-Derivative and its Variability**

**Description**

Calculates the cross-derivative required to evaluate interactions in logistic/probit regression models.
Usage

secondDiff(
  obj,
  vars,
  data,
  method = c("AME", "MER"),
  vals = NULL,
  typical = NULL
)

Arguments

obj An object of class glm that will be used to find the cross-derivative.
vars A vector of two variables to be used in calculating the derivative.
data A data frame.
method Indicate whether you want to use average marginal effects (AME) or marginal effects at representative values (MER).
vals A named list of length 2 where each element gives the minimum and maximum values used in the calculation.
typical A named vector of values at which to hold variables constant.

Details

The function calculates the second difference as \((\Pr(Y=1|x_1=\text{max}, x_2=\text{max}) - \Pr(Y=1|x_1=\text{min}, x_2=\text{max})) - (\Pr(Y=1|x_1=\text{max}, x_2=\text{min}) - \Pr(Y=1|x_1=\text{min}, x_2=\text{min}))\). The function uses a parametric bootstrap to calculate the sampling distribution of the second difference.

Value

A list with two elements:

tave The average second difference in each iteration of the bootstrap.
ind If type == 'AME', ind is returned with the second difference and measures of uncertainty for each individual observation in the original dataset
probs If type == 'MER', probs is returned with the full matrix of simulated predicted probabilities for the four conditions.

Author(s)

Dave Armstrong
**simPredpolr**  
*Calculate Predictions for Proportional Odds Logistic Regression*

**Description**

Calculates predicted probabilities from models of class `polr` from a model object and a vector of coefficient values. This is an auxiliary function used in `pre` if `sim=TRUE`.

**Usage**

```r
simPredpolr(object, coefs, n.coef)
```

**Arguments**

- `object`  
  An object of class `polr`.
- `coefs`  
  A vector of coefficients where elements 1 to `n.coef` give model coefficients and elements `n.coef+1` to `k` have intercepts.
- `n.coef`  
  Number of coefficients (minus intercepts) for the `polr` model.

**Value**

An `n x m`-category matrix of predicted probabilities

**Author(s)**

Dave Armstrong

---

**summary.balsos**  
*Summary method for Bayesian ALSOS*

**Description**

Summary method for objects of class `balsos`

**Usage**

```r
## S3 method for class 'balsos'
summary(object, ...)
```

**Arguments**

- `object`  
  Object of class `balsos`
- `...`  
  Other arguments, currently unimplemented
**summary.secdiff**  
*Summary for Second Difference Objects*

**Description**

Summary method for objects of class `secdiff`.

**Usage**

```r
## S3 method for class 'secdiff'
summary(object, ..., level = 0.95, digits = 3)
```

**Arguments**

- `object`: An object of class `secdiff`
- `...`: Other arguments to be passed down to `summary`.
- `level`: Confidence level for the confidence intervals
- `digits`: Number of digits to print

---

**test.balsos**  
*Testing Measurement Level Assumptions.*

**Description**

Uses the algorithm discussed in Armstrong and Jacoby (2018) to test the intervality of a variable scaled by the Bayesian ALSOS method.

**Usage**

```r
test.balsos(obj, cor.type = c("pearson", "spearman"))
```

**Arguments**

- `obj`: An object of class `balsos`
- `cor.type`: Type of correlation to be used in the p-value calculation.

**Value**

Printed output giving the Bayesian p-value evaluating the null that there is no interesting difference between the original values and the optimally scaled values.

**Author(s)**

Dave Armstrong
testGAMint  

Simulated F-test for Linear Interactions

Description

Simulates the sampling distribution of the F statistic when comparing a linear interaction model to a generalized additive model with a smooth over the two variables in the interaction.

Usage

testGAMint(m1, m2, data, R = 1000, ranCoef = FALSE)

Arguments

- **m1**: An object of class `gam` estimated with the mgcv package. This model should be linear in the interaction of the two x-variables of interest.
- **m2**: An object of class `gam` estimated with the mgcv package. This model should contain a smooth interaction. For two continuous variables, this should be done with `te()` unless the variables are measured in the same units (e.g., spatial coordinates) in which case the usual thin-plate regression spline will work. For categorical moderators, you should use the `s(x,by=D0)` and `s(x,by=D)` (for a dummy variable moderator, D, where D0=1 when D=0. Remember to include D as a parametric term in the model as well to account for the intercept difference between the two smooth terms.)
- **data**: Data frame used to estimate both models
- **R**: Number of simulated F values to create.
- **ranCoef**: Logical indicating whether the coefficients should be treated as fixed or whether they should be drawn from their implied sampling distribution for each iteration of the simulation.

Details

In simple simulations, an F-test of a linear interaction relative to a smooth interaction with a GAM using a nominal .05 type I error rate, has an actual type I error rate of more than double the nominal rate (this tended to be in the low teens). This function tries to build the F-distribution using simulation. First, it uses the coefficients from the linear interaction model, multiplies them by the coefficients from the linear interaction model and for each iteration of the simulation, it creates the simulated dependent variable by adding a random error to the linear predictor with the same standard deviation as the residual standard deviation from the linear interaction model. All of that is to say that this model has all of the same features as the linear interaction model, except that we are certain that this is the right model. The algorithm then estimates both the linear interaction model and the GAM with a smooth interaction on the original X variables and the new simulated y variable. The F-test is performed and the F-statistic saved for each iteration. The algorithm then calculates the probability of being to the right of the observed F-statistic in the simulated F-distribution.
**testLoess**

**Value**
- **obsF** The observed F-statistic from the test on the original models.
- **Fdist** The R different F-statistics calculated at each iteration of the simulation.

**Author(s)**
- Dave Armstrong

---

**testLoess**  
*Significance Test for Loess vs. LM*

**Description**
Calculates an F test to evaluate significant differences between a LOESS model and a parametric alternative estimated with `lm`.

**Usage**
```
testLoess(lmobj, loessobj, alpha = 0.05)
```

**Arguments**
- **lmobj** An object of class `lm`.
- **loessobj** An object of class `loess`.
- **alpha** Desired Type I error rate of test.

**Value**
Printed output describing the results of the test.

**Author(s)**
- Dave Armstrong

**Examples**
```
data(Prestige, package="carData")
linmod <- lm(prestige ~ income, data=Prestige)
lomod <- loess(prestige ~ income, data=Prestige)
testLoess(linmod, lomod)
```
Description

Tests for model improvements for non-linear transformations and polynomials with Clarke’s (2007) distribution-free test for non-nested models.

Usage

testNL(obj, var, transPower, polyOrder, plot = FALSE, ...)

## S3 method for class 'glm'
testNL(obj, var, transPower, polyOrder, plot = FALSE, ...)

## S3 method for class 'lm'
testNL(obj, var, transPower, polyOrder, plot = FALSE, ...)

Arguments

obj Object of a supported class in which non-linear functional forms will be tested.
var String giving name of variable to be tested.
transPower The power used in the transformation. For transformations in the range (-0.01, 0.01), the log transformation is used.
polyOrder The order of the polynomial to be used.
plot Logical indicating whether the effects should be plotted
...

Details

Three hypotheses are tested with this function. The first is whether the original specification is preferred to the power transformation. The second is whether the original specification is preferred to the polynomial model. The third is whether the power transformation is preferred to the polynomial model. All tests are done with the Clarke test.

Value

A plot or a data frame giving the results of the tests identified above.

Author(s)

Dave Armstrong

References

**tscslag**  
*Lag a time-series cross-sectional variables*

**Description**
Lags (or leads) a variable in a time-series cross-sectional dataset.

**Usage**
```r
  tscslag(dat, x, id, time, lagLength = 1)
```

**Arguments**
- `dat`: A data frame.
- `x`: A string identifying variable to be lagged.
- `id`: A string identifying the name of the cross-sectional identifier.
- `time`: A string identifying the name of the time variable.
- `lagLength`: The length of the lag, use negative values for leading variables.

**Value**
A vector giving the lagged values of `x`.

**Author(s)**
Dave Armstrong

**yeo.johnson**  
*Yeo-Johnson Transformation*

**Description**
Computes the normalizing Yeo-Johnson transformation.  

```r
  yeo.johnson(  
    y,  
    lambda,  
    derivative = 0,  
    epsilon = sqrt(.Machine$double.eps),  
    inverse = FALSE  
  )
```

This code and the details of the help file were taken from the VGAM package.
Arguments

y  Numeric, a vector or matrix.
lambda  Numeric. It is recycled to the same length as y if necessary.
derivative  Non-negative integer. The default is the ordinary function evaluation, otherwise
            the derivative with respect to lambda.
epsilon  Numeric and positive value. The tolerance given to values of lambda when
          comparing it to 0 or 2.
inverse  Logical. Return the inverse transformation?

Details

The Yeo-Johnson transformation can be thought of as an extension of the Box-Cox transformation. It handles both positive and negative values, whereas the Box-Cox transformation only handles positive values. Both can be used to transform the data so as to improve normality. They can be used to perform LMS quantile regression.

Value

A vector of transformed values.

Author(s)

Thomas Yee

Description

Uses nlminb to find the optimal Yeo-Johnson transformation parameters conditional on a parametric model specification.

Usage

yj_trans(form, data, trans.vars, round.digits = 3, ...)

Arguments

form  A formula with a dependent variable that will be optimally scaled
data  A data frame.
trans.vars  A character string identifying the variables that should be transformed
round.digits  Number of digits to round the transformation parameters.
...  Other arguments to be passed down to lm.
ziChange

**Value**

A linear model object that was estimated on the optimally transformed variables.

**Author(s)**

Dave Armstrong

---

**ziChange**

*Maximal First Differences for Zero-Inflated Models*

---

**Description**

Calculates the change in predicted counts or optionally the predicted probability of being in the zero-count group, for maximal discrete changes in all covariates holding all other variables constant at typical values.

**Usage**

```r
ziChange(obj, data, typical.dat = NULL, type = "count")
```

**Arguments**

- `obj` A model object of class `zeroinfl`.
- `data` Data frame used to fit object.
- `typical.dat` Data frame with a single row containing values at which to hold variables constant when calculating first differences. These values will be passed to `predict`, so factors must take on a single value, but have all possible levels as their levels attribute.
- `type` Character string of either ‘count’ (to obtain changes in predicted counts) or ‘zero’ (to obtain changes in the predicted probability of membership in the zero group).

**Details**

The function calculates the changes in predicted counts, or optionally the predicted probability of being in the zero group, for maximal discrete changes in the covariates. This function works with polynomials specified with the `poly` function. It also works with multiplicative interactions of the covariates by virtue of the fact that it holds all other variables at typical values. By default, typical values are the median for quantitative variables and the mode for factors. The way the function works with factors is a bit different. The function identifies the two most different levels of the factor and calculates the change in predictions for a change from the level with the smallest prediction to the level with the largest prediction.
Value

A list with the following elements:

diffs A matrix of calculated first differences
minmax A matrix of values that were used to calculate the predicted changes

Author(s)

Dave Armstrong
Index

*Topic datasets
  aclp, 4
  france, 23
  InteractionEx, 28

aclp, 4
alsosDV, 5
aveEffPlot, 6

balsos, 7
BGMtest, 8
binfit, 9
btscs, 10

cat2Table, 11
changeSig, 12
combTest, 13
crSpanTest, 13
crTest, 15
cv.lo2, 16

daIntfun, 17
daIntfun2, 18, 21
daIntfun3, 20
daMisp (daMisp-package), 3
daMisp-package, 3

effect, 22
effect_logistf, 22

france, 23
glmChange, 23
glmChange2, 25

inspect, 26
intEff, 27
InteractionEx, 28
intQualQuant, 28

loessDeriv, 31

logit.cc, 31
logit_cd/logit.cc, 31
logit_dd/logit.cc, 31

makeHypSurv, 32
mnlAveEffPlot, 33
mnlChange, 34
mnlChange2, 35
mnlfit, 36
mnlSig, 37

NKnots, 38
NKnotsTest, 40

oc2plot, 41
ordAveEffPlot, 42
ordChange, 41, 44
ordChange2, 45
ordfit, 46
outEff, 47
outXT, 48

p.adjust, 40
panel.2cat, 49
panel.ci, 50
panel.doublerug, 51
panel.transci, 52
pgumbel, 52
plot.balsos, 53
plot.loess, 54
plot.secdiff, 55
poisfit, 55
poisGOF, 56
powerTrans, 57
pre, 57
prepanel.ci, 59
print.diffci, 59
print.glmc2, 61
print.iqq, 61
print.mnlfit(mnlfit), 36
print.ordChange, 62
print.ordfit(ordfit), 46
print.pre, 62
probc1, 60, 63
probgroup, 64
probit_cc (logit_cc), 31
probit_cd (logit_cc), 31
probit_dd (logit_cc), 31
scaleDataFrame, 65
searchVarLabels, 66
secondDiff, 66
simPredpolr, 68
stan, 7
summary.balsos, 68
summary.secdiff, 69
test.balsos, 69
testGAMint, 70
testLoess, 71
testNL, 72
tscslag, 73
yeo.johnson, 73
yj_trans, 74
ziChange, 75