Package 'games'

February 23, 2015

Title Statistical Estimation of Game-Theoretic Models

Version 1.1.2

Date 2015-02-22

Description Provides estimation and analysis functions for strategic statistical models.

License GPL (>= 2)

Depends maxLik (>= 0.7-0), Formula (>= 0.2-0), MASS

Imports stringr

Suggests testthat

Author Curtis S. Signorino [aut], Brenton Kenkel [aut, cre]

Maintainer Brenton Kenkel <brenton.kenkel@gmail.com>

NeedsCompilation no

Repository CRAN

Date/Publication 2015-02-23 00:24:37

R topics documented:

`games-package` *A package for estimating strategic statistical models.*

Description

A package for estimating strategic statistical models.

Acknowledgements

We thank the Wallis Institute of Political Economy for financial support.

References

Brenton Kenkel and Curtis S. Signorino. 2014. "Estimating Extensive Form Games in R." *Journal of Statistical Software* 56(8):1–27.

Curtis S. Signorino. 2003. "Structure and Uncertainty in Discrete Choice Models." *Political Analysis* 11:316–344.

`data_122` *Simulated egame122 data*

Description

Simulated data for illustrating `egame122`.

Usage

```
data(data_122)
```

Details

The variables are:

`f1`, `f2` Factors with levels "a", "b", "c"

`x1–x5` Numeric variables entering Player 1's utilities

`z1–z3` Numeric variables entering Player 2's utilities

`a1` Indicator for Player 1's move (L or R)

`a2` Indicator for Player 2's move (L or R)

`y` Factor containing outcome

See Also

`egame122`

`data_123` *Simulated egame123 data*

Description

Simulated data for illustrating `egame123`.

Usage

```
data(data_123)
```

Details

The variables are:

`x1–x8` Regressors

`a1` Indicator for Player 1's move (L or R)

`a2` Indicator for Player 2's move (L or R)

`a3` Indicator for Player 3's move (L or R)

`y` Numeric variable containing outcome number: 1, 3, 5, or 6, corresponding to labels in the game tree in the `egame123` documentation.

See Also

`egame123`

data_ult *Simulated ultimatum data*

Description

Simulated data for illustrating `ultimatum`.

Usage

```
data(data_ult)
```

Details

The variables are:

`offer` The offer made by Player 1

`accept` Whether Player 2 accepted the offer (0 for rejection, 1 for acceptance)

`w1, w2` Variables entering both players' reservation values

`x1–x4` Variables entering Player 1's reservation value

`z1–z4` Variables entering Player 2's reservation value

The maximum offer size is 15.

See Also

`ultimatum`

egame12 *Strategic model with 2 players, 3 terminal nodes*

Description

Fits a strategic model with two players and three terminal nodes, as in the game illustrated below in "Details".

Usage

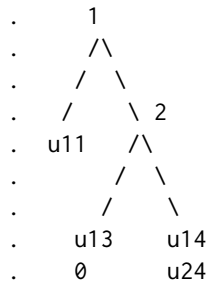
```
egame12(formulas, data, subset, na.action, link = c("probit", "logit"),
  type = c("agent", "private"), startvals = c("sbi", "unif", "zero"),
  fixedUtils = NULL, sdformula = NULL, sdByPlayer = FALSE, boot = 0,
  bootreport = TRUE, profile, method = "BFGS", ...)
```

Arguments

formulas	a list of four formulas, or a Formula object with four right-hand sides. See "Details" and the examples below.
data	a data frame containing the variables in the model.
subset	optional logical expression specifying which observations from data to use in fitting.
na.action	how to deal with NAs in data. Defaults to the na.action setting of options. See na.omit.
link	whether to use a probit (default) or logit link structure,
type	whether to use an agent-error ("agent", default) or private-information ("private") stochastic structure.
startvals	whether to calculate starting values for the optimization using statistical backwards induction ("sbi", default), draw them from a uniform distribution ("unif"), or to set them all to 0 ("zero")
fixedUtils	numeric vector of values to fix for u11, u13, u14, and u24 respectively. NULL (the default) indicates that these should be estimated with regressors rather than fixed.
sdformula	an optional list of formulas or a Formula containing a regression equation for the scale parameter. The formula(s) should have nothing on the left-hand side; the right-hand side should have one equation if sdByPlayer is FALSE and two equations if sdByPlayer is TRUE. See the examples below for how to specify sdformula.
sdByPlayer	logical: if scale parameters are being estimated (i.e., sdformula or fixedUtils is non-NULL), should a separate one be estimated for each player? This option is ignored unless fixedUtils or sdformula is specified.
boot	integer: number of bootstrap iterations to perform (if any).
bootreport	logical: whether to print status bar when performing bootstrap iterations.
profile	output from running profile.game on a previous fit of the model, used to generate starting values for refitting when an earlier fit converged to a non-global maximum.
method	character string specifying which optimization routine to use (see maxLik)
...	other arguments to pass to the fitting function (see maxLik).

Details

The model corresponds to the following extensive-form game, described in Signorino (2003):



If Player 1 chooses L, the game ends and Player 1 receives payoffs of u11. (Player 2's utilities in this case cannot be identified in a statistical model.) If Player 1 chooses L, then Player 2 can choose L, resulting in payoffs of u13 for Player 1 and 0 for Player 2, or R, with payoffs of u14 for 1 and u24 for 2.

The four equations specified in the function's `formulas` argument correspond to the regressors to be placed in u11, u13, u14, and u24 respectively. If there is any regressor (including the constant) placed in all of u11, u13, and u14, `egame12` will stop and issue an error message, because the model is then unidentified (see Lewis and Schultz 2003). There are two equivalent ways to express the formulas passed to this argument. One is to use a list of four formulas, where the first contains the response variable(s) (discussed below) on the left-hand side and the other three are one-sided. For instance, suppose:

- u11 is a function of `x1`, `x2`, and a constant
- u13 is set to 0
- u14 is a function of `x3` and a constant
- u24 is a function of `z` and a constant.

The list notation would be `formulas = list(y ~ x1 + x2, ~ 0, ~ x3, ~ z)`. The other method is to use the `Formula` syntax, with one left-hand side and four right-hand sides (separated by vertical bars). This notation would be `formulas = y ~ x1 + x2 | 0 | x3 | z`.

To fix a utility at 0, just use `0` as its equation, as in the example just given. To estimate only a constant for a particular utility, use `1` as its equation.

There are three equivalent ways to specify the outcome in `formulas`. One is to use a numeric vector with three unique values, with their values (from lowest to highest) corresponding with the terminal nodes of the game tree illustrated above (from left to right). The second is to use a factor, with the levels (in order as given by `levels(y)`) corresponding to the terminal nodes. The final way is to use two indicator variables, with the first standing for whether Player 1 moves L (0) or R (1), the second standing for Player 2's choice if Player 1 moves R. (The values of the second when Player 1 moves L should be set to 0 or 1, **not** `NA`, in order to ensure that observations are not dropped from the data when `na.action = na.omit`.) The way to specify `formulas` when using indicator variables is, for example, `y1 + y2 ~ x1 + x2 | 0 | x3 | z`.

If `fixedUtils` or `sdformula` is specified, the estimated parameters will include terms labeled `log(sigma)` (for probit links) or `log(lambda)`. These are the scale parameters of the stochastic components of the players' utility. If `sdByPlayer` is `FALSE`, then the variance of error terms (or the equation describing it, if `sdformula` contains non-constant regressors) is assumed to be common across all players. If `sdByPlayer` is `TRUE`, then two variances (or equations) are estimated: one for each player. For more on the interpretation of the scale parameters in these models and how it differs between the agent error and private information models, see Signorino (2003).

The model is fit using `maxLik`, using the BFGS optimization method by default (see `maxBFGS`). Use the `method` argument to specify an alternative from among those supplied by `maxLik`.

Value

An object of class `c("game", "egame12")`. A game object is a list containing:

`coefficients` estimated parameters of the model.

vcov estimated variance-covariance matrix. Cells referring to a fixed parameter (e.g., a utility when fixedUtils is specified) will contain NAs.

log.likelihood vector of individual log likelihoods (left unsummed for use with non-nested model tests).

call the call used to produce the model.

convergence a list containing the optimization method used (see argument method), the number of iterations to convergence, the convergence code and message returned by maxLik, and an indicator for whether the (analytic) gradient was used in fitting.

formulas the final Formula object passed to model.frame (including anything specified for the scale parameters).

link the specified link function.

type the specified stochastic structure (i.e., agent error or private information).

model the model frame containing all variables used in fitting.

xlevels a record of the levels of any factor regressors.

y the dependent variable, represented as a factor.

equations names of each separate equation (e.g., "u1(sq)", "u1(cap)", etc.).

fixed logical vector specifying which parameter values, if any, were fixed in the estimation procedure.

boot.matrix if boot was non-zero, a matrix of bootstrap parameter estimates (otherwise NULL).

localID an indicator for whether the Hessian matrix is negative definite, a sufficient condition for local identification of the model parameters.

The second class of the returned object, egame12, is for use in generation of predicted probabilities.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>) and Curtis S. Signorino

References

Jeffrey B. Lewis and Kenneth A Schultz. 2003. "Revealing Preferences: Empirical Estimation of a Crisis Bargaining Game with Incomplete Information." *Political Analysis* 11:345–367.

Curtis S. Signorino. 2003. "Structure and Uncertainty in Discrete Choice Models." *Political Analysis* 11:316–344.

See Also

summary.game and predProbs for postestimation analysis; makeFormulas for formula specification.

Examples

```
data("war1800")

## Model formula:
f1 <- esc + war ~ s_wt_re1 + revis1 | 0 | regime1 | balanc + regime2
##    ^^^^^^^^^   ^^^^^^^^^^^^^^^^^   ^   ^^^^^^^   ^^^^^^^^^^^^^^^^
##        y              u11         u13    u14           u24

m1 <- egame12(f1, data = war1800)
summary(m1)

m2 <- egame12(f1, data = war1800, link = "logit")
summary(m2)

m3 <- egame12(f1, data = war1800, subset = year >= 1850)
summary(m3)

m4 <- egame12(f1, data = war1800, boot = 10)
summary(m4)
summary(m4, useboot = FALSE)

## Estimating scale parameters under fixed utilities
utils <- c(-1, 0, -1.4, 0.1)
m5 <- egame12(esc + war ~ 1, data = war1800, fixedUtils = utils)
summary(m5)

m6 <- egame12(esc + war ~ 1, data = war1800, fixedUtils = utils, sdByPlayer = TRUE)
summary(m6)

## Estimating scale parameters with regressors
m7 <- egame12(f1, data = war1800, sdformula = ~ balanc - 1)
summary(m7)

## Using a factor outcome
y <- ifelse(war1800$esc == 1, ifelse(war1800$war == 1, "war", "cap"), "sq")
war1800$y <- factor(y, levels = c("sq", "cap", "war"))
f2 <- update(Formula(f1), y ~ .)

m8 <- egame12(f2, data = war1800)
summary(m8)
```

---

egame122 *Strategic model with 2 players, 4 terminal nodes*

Description

Fits a strategic model with two players and four terminal nodes, as in the game illustrated below in "Details".

Usage

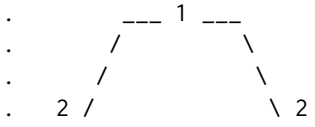
```
egame122(formulas, data, subset, na.action, link = c("probit", "logit"),
  type = c("agent", "private"), startvals = c("sbi", "unif", "zero"),
  fixedUtils = NULL, sdformula = NULL, sdByPlayer = FALSE, boot = 0,
  bootreport = TRUE, profile, method = "BFGS", ...)
```

Arguments

formulas	a list of six formulas, or a Formula object with six right-hand sides. See "Details" and "Examples".
data	a data frame.
subset	an optional logical vector specifying which observations from data to use in fitting.
na.action	how to deal with NAs in data. Defaults to the na.action setting of options. See na.omit
link	whether to use a probit (default) or logit link structure,
type	whether to use an agent-error ("agent", default) or private-information ("private") stochastic structure.
startvals	whether to calculate starting values for the optimization from statistical backwards induction ("sbi", default), draw them from a uniform distribution ("unif"), or to set them all to 0 ("zero")
fixedUtils	numeric vector of values to fix for u11, u12, u13, u14, u22, and u24. NULL (the default) indicates that these should be estimated with regressors, not fixed.
sdformula	an optional list of formulas or a Formula containing a regression equation for the scale parameter. See egame12 for details.
sdByPlayer	logical: if scale parameters are being estimated (i.e., sdformula or fixedUtils is non-NULL), should a separate one be estimated for each player? This option is ignored unless fixedUtils or sdformula is specified.
boot	integer: number of bootstrap iterations to perform (if any).
bootreport	logical: whether to print status bar during bootstrapping.
profile	output from running profile.game on a previous fit of the model, used to generate starting values for refitting when an earlier fit converged to a non-global maximum.
method	character string specifying which optimization routine to use (see maxLik)
...	other arguments to pass to the fitting function (see maxLik).

Details

The model corresponds to the following extensive-form game:



```
.    / \           / \
.   /   \         /   \
.  /     \       /     \
. u11    u12    u13    u14
. 0      u22    0      u24
```

For additional details on any of the function arguments or options, see egame12. The only difference is that the right-hand side of formulas must have six components (rather than four) in this case.

Ways to specify the dependent variable in egame122:

- Numeric vector y, numbered 1 through 4, corresponding to the outcomes as labeled in the game tree above.
- Factor y, where y has four levels, corresponding in order to the outcomes as labeled above.
- Indicator variables y1 + y2, where y1 indicates whether Player 1 moves left or right, and y2 indicates whether Player 2 moves left or right.
- Indicator variables y1 + y2 + y3, where y1 indicates whether Player 1 moves left or right, y2 indicates Player 2's move in case Player 1 moved left, and y3 indicates Player 2's move in case Player 1 moved right. Non-observed values of y2 and y3 should be set to 0, **not** NA, to ensure that observations are not dropped when na.action = na.omit.

Value

An object of class c("game", "egame122"). See egame12 for a description of the game class.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>) and Curtis S. Signorino

Examples

```
data("data_122")

## Model formula:
fr1 <- y ~ x1 + x2 | x3 + f1 | 0 | x4 + x5 | z1 + z2 | z3 + f2
##     ^   ^^^^^^^   ^^^^^^^   ^   ^^^^^^^   ^^^^^^^   ^^^^^^^
##     y     u11       u12    u13    u14       u22       u24

m1 <- egame122(fr1, data = data_122)
summary(m1)

## Dummy specification of the dependent variable
fr2 <- update(Formula(fr1), a1 + a2 ~ .)
m2 <- egame122(fr2, data = data_122)
summary(m2)
```

egame123 *Strategic model with 3 players, 4 terminal nodes*

Description

Fits a strategic model with three players and four terminal nodes, as in the game illustrated below in "Details".

Usage

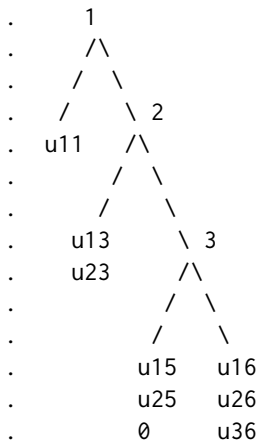
```
egame123(formulas, data, subset, na.action, link = c("probit", "logit"),
  type = c("agent", "private"), startvals = c("sbi", "unif", "zero"),
  fixedUtils = NULL, sdformula = NULL, sdByPlayer = FALSE, boot = 0,
  bootreport = TRUE, profile, method = "BFGS", ...)
```

Arguments

formulas	a list of eight formulas, or a Formula object with eight right-hand sides. See "Details" and "Examples".
data	a data frame.
subset	an optional logical vector specifying which observations from data to use in fitting.
na.action	how to deal with NAs in data. Defaults to the na.action setting of options. See na.omit
link	whether to use a probit (default) or logit link structure,
type	whether to use an agent-error ("agent", default) or private-information ("private") stochastic structure.
startvals	whether to calculate starting values for the optimization from statistical backwards induction ("sbi", default), draw them from a uniform distribution ("unif"), or to set them all to 0 ("zero")
fixedUtils	numeric vector of values to fix for u11, u13, u15, u16, u23, u25, u26, and u36. NULL (the default) indicates that these should be estimated with regressors, not fixed.
sdformula	an optional list of formulas or a Formula containing a regression equation for the scale parameter. See egame12 for details.
sdByPlayer	logical: if scale parameters are being estimated (i.e., sdformula or fixedUtils is non-NULL), should a separate one be estimated for each player? This option is ignored unless fixedUtils or sdformula is specified.
boot	integer: number of bootstrap iterations to perform (if any).
bootreport	logical: whether to print status bar during bootstrapping.
profile	output from running profile.game on a previous fit of the model, used to generate starting values for refitting when an earlier fit converged to a non-global maximum.
method	character string specifying which optimization routine to use (see maxLik)
...	other arguments to pass to the fitting function (see odemaxLik).

Details

The model corresponds to the following extensive-form game:



For additional details on any of the function arguments or options, see egame12. The only difference is that the right-hand side of formulas must have eight components (rather than four) in this case.

Ways to specify the dependent variable in egame123:

- Numeric vector y containing 4 unique values, corresponding to the outcomes (in order from left to right) as labeled in the game tree above.
- Factor y, where y has four levels, corresponding in order to the outcomes as labeled above.
- Indicator variables y1 + y2 + y3, where y1 indicates whether Player 1 moves left or right, y2 indicates Player 2's move, and y3 indicates Player 3's move. Non-observed values of y2 and y3 (where the game ended before the move could be made) should be set to 0, **not** NA, to ensure that observations are not dropped when na.action = na.omit.

Value

An object of class c("game", "egame123"). See egame12 for a description of the game class.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

Examples

```
data("data_123")

## Model formula:
f1 <- y ~ x1 + x2 | 0 | x3 | x4 + x5 | 0 | x6 | x7 | x8
##    ^   ^^^^^^^   ^   ^^   ^^^^^^^   ^   ^^   ^^   ^^
##    y     u11    u13  u15    u16    u23  u25  u26  u36

m1 <- egame123(f1, data = data_123, link = "probit", type = "private")
summary(m1)
```

```
## Dummy specification of the dependent variable
f2 <- update(Formula(f1), a1 + a2 + a3 ~ .)
m2 <- egame123(f2, data = data_123, link = "probit", type = "private")
summary(m2)
```

`latexTable` *LaTeX table for strategic models*

Description

Makes a LaTeX table of strategic model results.

Usage

```
latexTable(x, digits = max(3, getOption("digits") - 2), scientific = NA,
  blankfill = "", math.style.negative = TRUE, file = "",
  floatplace = "htbp", caption = NULL, label = NULL, rowsep = 2,
  useboot = TRUE)
```

Arguments

`x`	a fitted model of class `game`.
`digits`	number of digits to print.
`scientific`	logical or integer value to control use of scientific notation. See `format`.
`blankfill`	text to fill empty cells (those where a certain variable did not enter the given equation).
`math.style.negative`	whether negative signs should be "math style" or plain hyphens. Defaults to `TRUE`.
`file`	file to save the output in. Defaults to `""`, which prints the table to the R console.
`floatplace`	where to place the table float; e.g., for `\begin{table}[htp]`, use `floatplace = "htp"`.
`caption`	caption to use (none if `NULL`)
`label`	reference label to use (none if `NULL`)
`rowsep`	amount of space (in points) to put between rows.
`useboot`	logical: use bootstrap estimates (if available) to calculate standard errors?

Details

`latexTable` prints LaTeX code for the presentation of results from a strategic model. Each row contains one regressor, and each column contains one of the utility (or variance term) equations in the model. For example, in a model fit with `egame12`, the four columns will be u11, u13, u14, and u24 respectively. Each cell contains the estimated parameter, atop its standard error in parentheses. Cells that are empty because a regressor is not included in a particular equation are filled with the

string specified in the option `blankfill`. Signorino and Tarar (2006, p. 593) contains a table of this variety.

The table generated depends on the **multirow** package for LaTeX, so make sure to include `\usepackage{multirow}` in the preamble of your document.

The `digits` option does not yet work seamlessly; you may have to resort to trial and error.

Value

`x`, invisibly.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

References

Curtis S. Signorino and Ahmer Tarar. 2006. "A Unified Theory and Test of Extended Immediate Deterrence." *American Journal of Political Science* 50(3):586–605.

Examples

```
data("war1800")
f1 <- esc + war ~ s_wt_re1 + revis1 | 0 | balanc + revis1 | balanc
m1 <- egame12(f1, data = war1800)

latexTable(m1)
latexTable(m1, digits = 8)
latexTable(m1, blankfill = "--")  ## Dashes in blank cells

## Not run:
    latexTable(m1, file = "my_table.tex")  ## Write to file

## End(Not run)
```

---

leblang2003 *Currency attacks*

Description

Data on speculative currency attacks and devaluation decisions for 90 countries from 1985 to 1998.

Usage

```
data(leblang2003)
```

Details

The dataset is taken from Leblang (2003). The unit of observation is the country-month, and the variables are:

`outcome` Whether the country faced no speculative attack, defended its currency against an attack, or devalued in response to an attack in the given month

`preelec` Indicator for being in the three months prior to an election

`postelec` Indicator for being in the three months following an election

`rightgov` Indicator for a right-wing party being in power

`unifgov` Indicator for unified government: in presidential systems, the same party controlling the presidency and the lower house of the legislature; in parliamentary systems, one party/coalition having a majority of seats

`lreserves` Logged ratio of currency reserves to base money in the previous month

`realinterest` Domestic real interest rate in the previous month

`lexports` Logged ratio of exports to GDP in the previous month

`capcont` Indicator for capital controls in the previous year

`overval` Overvaluation of the real exchange rate

`creditgrow` Domestic credit growth in the previous month

`service` External debt service (as percentage of exports) paid in previous month

`USinterest` U.S. domestic interest rates in the previous month

`contagion` Number of other countries experiencing speculative attacks in the same month

`prioratt` Number of prior speculative attacks experienced by the country

`nation` Country name

`month` Month of observation

`year` Year of observation

All of the non-binary variables other than `nation`, `month`, and `year` are standardized to have mean 0 and unit variance.

We are grateful to David Leblang for allowing us to redistribute his data. The original replication file is available in Stata format at https://sites.google.com/site/davidaleblang/data-1 (as of 2015-02-22).

References

David Leblang. 2003. "To Defend or Devalue: The Political Economy of Exchange Rate Policy." *International Studies Quarterly* 47: 533–559.

See Also

egame12

Examples

```
## Replicate analysis in Leblang (2003)
data("leblang2003")

## NOTE: Convergence tolerance is set to 1e-4 to reduce testing runtime on
## CRAN; do not reduce tolerance in real applications!
m1 <- egame12(outcome ~
              capcont + lreserves + overval + creditgrow + USinterest + service
              + contagion + prioratt - 1 |
              1 |
              1 |
              unifgov + lexports + preelec + postelec + rightgov + realinterest
              + capcont + lreserves,
              data = leblang2003,
              link = "probit",
              type = "private",
              reltol = 1e-4)

summary(m1)
```

LW *Lambert's W*

Description

Solves for W in the equation $We^W = x$.

Usage

```
LW(x)
```

Arguments

x	vector of values to solve for.

Details

The function is based on the code given in Barry et al. (1995). It is used to calculate fitted values for the `ultimatum` model.

If negative values of `x` are supplied, `NaN`s will likely be returned.

Value

Solutions to Lambert's W for each value in `x`.

Author(s)

Curt Signorino (<curt.signorino@rochester.edu>)

References

D.A. Barry, P.J. Culligan-Hensley, and S.J. Barry. 1995. "Real Values of the W-Function." *ACM Transactions on Mathematical Software* 21(2):161–171.

Examples

```
x <- rexp(10)
w <- LW(x)
all.equal(x, w * exp(w))
```

`makeFormulas` *Model formula construction*

Description

Interactive prompt for constructing model formulas.

Usage

```
makeFormulas(model, outcomes)
```

Arguments

`model`	name of the model (must be from the **games** package) for which to make a formula.
`outcomes`	character vector with descriptions of the possible outcomes of the game (see "Details" for a more precise explanation)

Details

All of the staistical models in the **games** package require the specification of multiple model formulas, as each player's utility is a function of potentially different regressors. The number of equations to specify ranges from two in `ultimatum` to eight in `egame123`. `makeFormulas` is an interactive tool to simplify the specification process.

To use `makeFormulas`, specify the model you want to fit (`model`) and descriptions of the outcomes of the game (`outcomes`). The order of the descriptions in `outcomes` should match the numbering in the game tree in the help page for `model`. For example, with `egame122`, the order is:

1. Player 1 moves Left, Player 2 moves Left
2. Player 1 moves Left, Player 2 moves Right
3. Player 1 moves Right, Player 2 moves Left
4. Player 1 moves Right, Player 2 moves Right

If the dependent variable in the dataset (`dat`) is a factor (`y`) whose levels contain the descriptions, then either `outcomes = dat$y` or `outcomes = levels(dat$y)` will work.

As an example, consider the following use of `egame122`. Player 1 is the legislature, which can propose budget cuts (left on the game tree) or increases (right). Player 2 is the president, who can sign or veto the legislature's budget proposal (left and right respectively). The variables of interest are the president's party (`presparty`), the legislature's party (`legparty`), and the year's percentage GDP growth (`growth`). To construct the formulas for this case, run `makeFormulas(egame122, outcomes = c("budget cuts pas`
The first set of options that appears is

```
Equation for player 1's utility from budget cuts passed:

1: fix to 0
2: intercept only
3: regressors, no intercept
4: regressors with intercept
```

To specify this utility as a function of a constant, the legislature's party, and GDP growth, select option 4 and enter `legparty growth` at the prompt. `makeFormulas` will then move on to ask about Player 1's utility for the other three outcomes, followed by Player 2's utility for the outcomes for which her utility is not fixed to 0 (see `egame122`). See "Examples" below for a full example of the input and constructed formula in this case.

It is **not** necessary to use `makeFormulas` to specify model formulas. See the help file of each model for examples of "manually" making the formula.

Value

An object of class `"Formula"`, typically to be used as the `formulas` argument in a statistical model from the **games** package.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

See Also

`Formula` (and the **Formula** package generally) for the details of how **games** deals with multiple model formulas.

Examples

```
## Not run:
R> f1 <- makeFormulas(egame122, outcomes = c("budget cuts passed",
"budget cuts vetoed", "budget increase passed", "budget increase vetoed"))

---
Equation for player 1's utility from budget cuts passed:

1: fix to 0
2: intercept only
3: regressors, no intercept
```

```
4: regressors with intercept

Selection: 4

Enter variable names (separated by spaces):
legparty growth

---
Equation for player 1's utility from budget cuts vetoed:

1: fix to 0
2: intercept only
3: regressors, no intercept
4: regressors with intercept

Selection: 2

---
Equation for player 1's utility from budget increase passed:

1: fix to 0
2: intercept only
3: regressors, no intercept
4: regressors with intercept

Selection: 4

Enter variable names (separated by spaces):
legparty growth

---
Equation for player 1's utility from budget increase vetoed:

1: fix to 0
2: regressors, no intercept

Selection: 1

---
Equation for player 2's utility from budget cuts vetoed:

1: fix to 0
2: intercept only
3: regressors, no intercept
4: regressors with intercept

Selection: 4

Enter variable names (separated by spaces):
presparty growth

---
Equation for player 2's utility from budget increase vetoed:
```

```
1: fix to 0
2: intercept only
3: regressors, no intercept
4: regressors with intercept

Selection: 4

Enter variable names (separated by spaces):
presparty growth

---
What is the name of the dependent variable in the dataset? (If stored as
action indicators/dummies, separate their names with spaces.)
budgincrease veto
R> f1
budgincrease + veto ~ legparty + growth | 1 | legparty + growth |
    0 | presparty + growth | presparty + growth

## End(Not run)
```

Mode *Mode of a vector*

Description

Finds the modal value of a vector of any class.

Usage

```
Mode(x, na.rm = FALSE)
```

Arguments

x	a vector (lists and arrays will be flattened).
na.rm	logical: strip NA values?

Details

Based on the Stack Overflow answer http://stackoverflow.com/a/8189441/143383

Value

The value of x that occurs most often. If there is a tie, the one that appears first (among those tied) is chosen.

Author(s)

Ken Williams (on Stack Overflow)

Examples

```
x <- c(1, 3, 3, 4)
Mode(x)  # 3
x.char <- letters[x]
Mode(x.char)  # "c"
x.factor <- as.factor(x.char)
Mode(x.factor)  # "c", with levels "a", "c", "d"
x.logical <- x > 3
Mode(x.logical)  # FALSE

## Behavior with ties
y <- c(3, 3, 1, 1, 2)
Mode(y)  # 3
z <- c(2, 1, 1, 3, 3)
Mode(z)  # 1
```

---

plot.predProbs *Plot predicted probabilities*

Description

Plots predicted probabilities and associated confidence bands, using the data returned from a call to predProbs.

Usage

```
## S3 method for class 'predProbs'
plot(x, which = NULL, ask = FALSE, ...)
```

Arguments

x	an object of class predProbs (i.e., a data frame returned by predProbs).
which	optional integer specifying which plot (as numbered in the menu displayed when ask == TRUE) to make. If none is given, all available plots are printed in succession.
ask	logical: display interactive menu with options for which plot to make?
...	further arguments to pass to the plotting function. See plot.default (when the variable on the x-axis is continuous) or bxp (when it is discrete).

Details

Most predProbs objects will be associated with multiple plots: one for each outcome in the estimated model. These are the three or four terminal nodes for a egame12 or egame122 model respectively; for an ultimatum model, these are the expected offer and the probability of acceptance. By default, plot.predProbs produces plots for all of them, so only the last will be visible unless the graphics device is set to have multiple figures (e.g., by setting par(mfrow = ...)). The argument ask displays a menu to select among the possible plots for a given object, and which allows for this to be done non-interactively.

Value

an object of class `preplot.predProbs`, invisibly. This contains the raw information used by lower-level plotting functions.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

Examples

```
data("war1800")
f1 <- esc + war ~ s_wt_re1 + revis1 | 0 | regime1 | balanc + regime2
m1 <- egame12(f1, data = war1800, boot = 10)
pp1 <- predProbs(m1, x = "balanc", n = 5)
pp2 <- predProbs(m1, x = "regime1")

## if "ask" is FALSE and "which" isn't specified, all plots are printed
op <- par(mfrow = c(2, 2))
plot(pp1)
par(op)

## Not run:
    plot(pp1, ask = TRUE)

    ## Make a plot selection (or 0 to exit):
    ##   1: plot: Pr(~esc)
    ##   2: plot: Pr(esc,~war)
    ##   3: plot: Pr(esc,war)
    ##   4: plot all terms

## End(Not run)

## To change line type for confidence bounds, use argument `lty.ci`
plot(pp1, which = 3, lty.ci = 3)

## All the standard plotting options work too
plot(pp1, which = 3, xlab = "Capabilities", ylab = "Probability", main = "Title")

## Discrete `x` variables are plotted via R's boxplot functionality
plot(pp2, which = 3)
```

`plot.profile.game` *Plot profiles of strategic model log-likelihoods*

Description

Plot output of `profile.game`.

Usage

```
## S3 method for class 'profile.game'
plot(x, show.pts = FALSE, ...)
```

Arguments

x	an object of class profile.game, typically created by running profile.game on a fitted game model
show.pts	logical: plot a point for the log-likelihood of each profiled model?
...	other arguments, currently ignored

Details

This method provides plots for a quick assessment of whether game models have failed to converge to a global maximum. For each parameter profiled (see profile.game for details of the profiling process), a spline interpolation of the log-likelihood profile is provided, with an "x" marking the value at the original parameter estimate.

Sometimes the plot will seem to indicate that the original fit did not reach the global maximum, even though profile.game did not issue the non-covergence warning. This is an artifact of the interpolation, which can be confirmed by re-running plot.profile.game with the argument show.pts = TRUE.

Value

x, invisibly

Author(s)

Brenton Kenkel

See Also

profile.game

---

predict.game *Predicted probabilities for strategic models*

Description

Makes predicted probabilities from a strategic model.

Usage

```
## S3 method for class 'game'
predict(object, ...)

## S3 method for class 'egame12'
predict(object, newdata, type=c("outcome", "action"), na.action = na.pass, ...)
## S3 method for class 'egame122'
predict(object, newdata, type=c("outcome", "action"), na.action = na.pass, ...)
## S3 method for class 'egame123'
predict(object, newdata, type=c("outcome", "action"), na.action = na.pass, ...)
## S3 method for class 'ultimatum'
predict(object, newdata, na.action = na.pass, n.sim = 1000, ...)
```

Arguments

object	a fitted model of class game.
...	other arguments, currently ignored.
newdata	data frame of values to make the predicted probabilities for. If this is left empty, the original dataset is used.
type	whether to provide probabilities for outcomes (e.g., L, RL, or RR in egame12) or for actions (e.g., whether 2 moves L or R given that 1 moved R).
na.action	how to deal with NAs in newdata
n.sim	number of simulation draws to use per observation for ultimatum models (see Details).

Details

This method uses a fitted strategic model to make predictions for a new set of data. This is useful for cross-validating or for graphical analysis. For many uses, such as analyzing the marginal effect of a particular independent variable, the function predProbs will be more convenient.

In the ultimatum model, there is not an analytic expression for the expected value of Player 1's offer. Therefore, predicted values are instead generating via simulation by drawing errors from a logistic distribution. The number of draws per observation can be controlled via the n.sim argument. For replicability, we recommend seeding the random number generator via set.seed before using predict.ultimatum.

Value

A data frame of predicted probabilities.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

See Also

predProbs provides a more full-featured and user-friendly wrapper, including plots and confidence bands.

predProbs *User-friendly predicted probability analysis*

Description

Easy generation and plotting of predicted probabilities from a fitted strategic model.

Usage

```
predProbs(model, x, xlim = c(min(x), max(x)), n = 100, ci = 0.95,
  type = c("outcome", "action"), makePlots = FALSE, report = TRUE, ...)
```

Arguments

model	a fitted model of class game.
x	character string giving the name of the variable to place "on the x-axis" while all others are held constant. Partial matches are accepted.
xlim	numeric, length 2: the range that x should be varied over (if x is continuous). Defaults to the observed range of x.
n	integer: the number of observations to generate (if x is continuous).
ci	numeric: width of the confidence interval to estimate around each predicted probability. Set to 0 to estimate no confidence intervals.
type	whether to generate predicted values for outcomes (the default) or actions
makePlots	logical: whether to automatically make the default plot for the returned object. See plot.predProbs.
report	logical: whether to print a status bar while obtaining the confidence intervals for the predicted probabilities.
...	used to set values for variables other than x in the profile of observations. See "Details" and "Examples".

Details

predProbs provides an easy way to analyze the estimated marginal effect of an independent variable on the probability of particular outcomes, using the estimates returned by a strategic model. The procedure is designed so that, for a preliminary analysis, the user can simply specify the fitted model and the independent variable of interest, and quickly obtain plots of predicted probabilities. However, it is flexible enough to allow for finely tuned analysis as well.

The procedure works by varying x, the variable of interest, across its observed range (or one specified by the user in xlim) while holding all other independent variables in the model fixed. The profile created by default is as follows (the same defaults as in the sim function in the **Zelig** package):

- numeric, non-binary variables are fixed at their means
- ordered variables are fixed at their medians

- all others are fixed at their modes (see Mode)

However, it is possible to override these defaults for any or all variables. For example, to set a variable named polity to its lower quartile, call predProbs with the argument polity = quantile(polity, 0.25). To set a factor variable to a particular level, provide the name of the level as a character string (in quotes). (Also see the examples below.)

Confidence intervals for each predicted point are generated by bootstrap. If model has a non-null boot.matrix element (i.e., a bootstrap was performed with the model fitting), then these results are used to make the confidence intervals. Otherwise, a parametric bootstrap sample is generated by sampling from a multivariate normal distribution around the parameter estimates. In this case, a warning is issued.

For information on plotting the predicted probabilities, see plot.predProbs. The plots are made with base graphics. If you prefer to use an alternative graphics package, all the information necessary to make the plots is included in the data frame returned.

Value

An object of class predProbs. This is a data frame containing each hypothetical observation's predicted probability, the upper and lower bounds of the confidence interval, and the value of each regressor.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>). Code for escaping special regex characters was taken from the Hmisc package's function escapeRegex, written by Charles Dupont.

See Also

predict.game for somewhat more flexible (but fussier) generation of predicted probabilities.

Examples

```
data("war1800")
f1 <- esc + war ~ s_wt_re1 + revis1 | 0 | regime1 | balanc + regime2
m1 <- egame12(f1, data = war1800, boot = 10)

pp1 <- predProbs(m1, x = "s_wt_re1", n = 5)
print(pp1)  ## Hypothetical observations and their predicted probs
plot(pp1, which = 2)  ## See ?plot.predProbs for more plot examples

## Changing the profile used
pp2 <- predProbs(m1, x = "s_wt_re1", n = 5, revis1 = 1, balanc = 0.7)
pp3 <- predProbs(m1, x = "s_wt_re1", n = 5, regime1 = "dem")
pp4 <- predProbs(m1, x = "s_wt_re1", n = 5, balanc = median(balanc))

## Variable names (other than `x`) must match exactly!
## Not run:
    pp5 <- predProbs(m1, x = "s_wt_re1", bal = 0.7)  ## Error will result

## End(Not run)
```

```
## `x` can be a factor too
pp6 <- predProbs(m1, x = "regime1")

## Action probabilities
pp7 <- predProbs(m1, x = "regime1", type = "action")
```

`print.game` *Print a strategic model object*

Description

The default method for printing a game object.

Usage

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

Arguments

x	a fitted model of class game
...	other arguments, currently ignored

Details

Prints the call and coefficients of a fitted strategic model.

Value

x, invisibly

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

print.summary.game *Print strategic model summary*

Description

Print output from summary.game

Usage

```
## S3 method for class 'summary.game'
print(x, ...)
```

Arguments

x	an object of class summary.game, typically produced by running summary on a fitted model of class game
...	other arguments, currently ignored

Details

Prints the standard regression results table from a fitted strategic model, along with the log-likelihood, AIC, and number of observations.

Value

x, invisibly.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

See Also

summary.game

profile.game *Likelihood profiles for fitted strategic models*

Description

Calculate profile likelihood to assess convergence of a model.

Usage

```
## S3 method for class 'game'
profile(fitted, which = 1:p, steps = 5, dist = 3,
  use.se = TRUE, report = TRUE, ...)
```

Arguments

`fitted`	a fitted model of class `game`.
`which`	integer vector giving the indices of the parameters to be profiled. The default is to use all parameters. Parameters that were held fixed in the original fitting are ignored if selected.
`steps`	number of steps to take (in each direction) from the original value for each parameter to be profiled.
`dist`	distance the last step should be from the original parameter value (in terms of standard errors if `use.se` is `TRUE`; absolute value otherwise). Should be a numeric vector of length equal to 1 or `length(coef(fitted))`.
`use.se`	logical: whether `dist` refers to standard errors
`report`	logical: whether to print status bar (for complex models or those with many parameters, profiling can be lengthy)
`...`	other arguments to be passed to the fitting function (see odemaxLik).

Details

Likelihood profiling can help determine if a model fit failed to reach a global maximum, which can be an issue (especially for the `ultimatum` model). The process of profiling is as follows: a parameter selected to be profiled is fixed at certain values spaced around its originally estimated value, while the log-likelihood is maximized with respect to the other parameters in the model. For models with large numbers of observations or parameters, profiling may take a long time, as $p \times (2s + 1)$ models will be fit (p: number of parameters; s: number of steps).

The function will issue a warning if a model fit in profiling has a log-likelihood exceeding that of the original model. This means the original fit failed to reach a global maximum, and any inferences based on the fitted model are invalid. If this occurs, refit the model, passing the `profile.game` output to the fitting function's `profile` argument (as in the example below). The new fit will use the coefficients from the profile fit with the highest log-likelihood as starting values.

The function is based loosely on `profile.glm` in the **MASS** package. However, that function focuses on the calculation of exact confidence intervals for regression coefficients, whereas this one is for diagnosing non-convergence. Future versions of the **games** package may incorporate the confidence interval functionality as well.

Value

A list of data frames, each containing the estimated coefficients across the profiled values for a particular parameter. The first column of each data frame is the log-likelihood for the given fits. The returned object is of class `c("profile.game", "profile")`.

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

See Also

`plot.profile.game` for plotting profiled likelihoods

Examples

```
data("student_offers")

## A model that does not converge to global max
f1 <- offer + accept ~ gender1 | gender2
m1 <- ultimatum(f1, maxOffer = 100, data = student_offers, s2 = 1)

p1 <- profile(m1)  ## Issues warning
plot(p1)

## Refit model with better starting values
m2 <- ultimatum(f1, maxOffer = 100, data = student_offers, s2 = 1, profile = p1)
p2 <- profile(m2)
plot(p2)

logLik(m1)
logLik(m2)  ## Improved
```

---

student_offers *Data from students playing the ultimatum game*

Description

Data from a trial of the ultimatum game with graduate students.

Usage

```
data(student_offers)
```

Details

The variables are:

`offer` The offer made by Player 1

`accept` Whether Player 2 accepted the offer (0 for rejection, 1 for acceptance)

`gender1` Whether Player 1 is female

`gender2` Whether Player 2 is female

The maximum offer size is 100.

See Also

`ultimatum`

summary.game *Summarize a strategic model object*

Description

The default method for summarizing a game object.

Usage

```
## S3 method for class 'game'
summary(object, useboot = TRUE, ...)
```

Arguments

object	a fitted model of class game
useboot	logical: use bootstrap estimates (if present) to construct standard error estimates?
...	other arguments, currently ignored

Details

Forms the standard regression results table from a fitted strategic model. Normally used interactively, in conjunction with print.summary.game.

Value

an object of class summary.game, containing the coefficient matrix and other information needed for printing

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

See Also

print.summary.game

`ultimatum` *Statistical ultimatum game*

Description

Estimates the statistical ultimatum game described in Ramsay and Signorino (2009), illustrated below in "Details".

Usage

```
ultimatum(formulas, data, subset, na.action, minOffer = 0, maxOffer,
  offertol = sqrt(.Machine$double.eps), s1 = NULL, s2 = NULL,
  outcome = c("both", "offer"), boot = 0, bootreport = TRUE, profile,
  method = "BFGS", ..., reltol = 1e-12)
```

Arguments

`formulas`	a list of two formulas, or a `Formula` object with two right-hand sides. See "Details" and the examples below.
`data`	data frame containing the variables in the model.
`subset`	optional logical expression specifying which observations from `data` to use in fitting.
`na.action`	how to deal with `NA`s in `data`. Defaults to the `na.action` setting of `options`. See `na.omit`.
`minOffer`	numeric: the lowest offer Player 1 could feasibly make (default 0).
`maxOffer`	numeric: the highest offer Player 1 could feasibly make.
`offertol`	numeric: offers within `offertol` of `minOffer`/`maxOffer` will be considered to be at the minimum/maximum. (This is used to prevent floating-point problems and need not be changed in most applications.)
`s1`	numeric: scale parameter for Player 1. If `NULL` (the default), the parameter will be estimated.
`s2`	numeric: scale parameter for Player 2. If `NULL` (the default), the parameter will be estimated.
`outcome`	the outcome of interest: just Player 1's offer ("offer") or both the offer and its acceptance ("both"). See "Details".
`boot`	integer: number of bootstrap iterations to perform (if any).
`bootreport`	logical: whether to print status bar when performing bootstrap iterations.
`profile`	output from running `profile.game` on a previous fit of the model, used to generate starting values for refitting when an earlier fit converged to a non-global maximum.
`method`	character string specifying which optimization routine to use (see `maxLik`)
`...`	other arguments to pass to the fitting function (see `maxLik`).
`reltol`	numeric: relative convergence tolerance level (see `optim`). Use of values higher than the default is discouraged.

Details

The model corresponds to the following extensive-form game, described in Ramsay and Signorino (2009):

```
.       1
.      / \
.     /   \
.    /     \ y in [0, Q]
.   /       \
.   ---------
.       /\  2
.      /  \
.     /    \
.    /      \
. Q - y     R1
. y         R2
```

Q refers to the maximum feasible offer (the argument `maxOffer`).

The two equations on the right-hand side of `formulas` refer to Player 1's and Player 2's reservation values respectively. The left-hand side should take the form `offer + acceptance`, where `outcome` contains the numeric value of the offer made and `acceptance` is an indicator for whether it was accepted. (If `outcome` is set to "offer", the acceptance indicator can be omitted. See below for more.)

The `outcome` argument refers to whether the outcome of interest is just the level of the offer made, or both the level of the offer and whether it was accepted. If acceptance was unobserved, then `outcome` should be set to "offer". If so, the estimates for Player 2's reservation value should be interpreted as Player 1's expectations about these parameters. It may also be useful to set `outcome` to "offer" even if acceptance data are available, for the purpose of comparing the strategic model to other models of offer levels (as in Ramsay and Signorino 2009). If an acceptance variable is specified but `outcome` is set to "offer", the acceptance data will be used for starting values but not in the actual fitting.

Numerical instability is not uncommon in the statistical ultimatum game, especially when the scale parameters are being estimated.

Value

An object of class `c("game", "ultimatum")`. For details on the `game` class, see `egame12`. The `ultimatum` class is just for use in the generation of predicted values (see `predProbs`) and profiling (see `profile.game`).

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>) and Curtis S. Signorino

References

Kristopher W. Ramsay and Curtis S. Signorino. 2009. "A Statistical Model of the Ultimatum Game." Available online at http://www.rochester.edu/college/psc/signorino/research/RamsaySignorino_Ultimatum.pdf.

Examples

```
data(data_ult)

## Model formula:
f1 <- offer + accept ~ x1 + x2 + x3 + x4 + w1 + w2 | z1 + z2 + z3 + z4 + w1 + w2
##                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^   ^^^^^^^^^^^^^^^^^^^^^^^^^^^
##                                 R1                            R2

m1 <- ultimatum(f1, data = data_ult, maxOffer = 15)
summary(m1)

## Estimating offer size only
f2 <- update(Formula(f1), offer ~ .)
m2 <- ultimatum(f2, data = data_ult, maxOffer = 15, outcome = "offer")
summary(m2)

## Fixing scale terms
m3 <- ultimatum(f1, data = data_ult, maxOffer = 15, s1 = 5, s2 = 1)
summary(m3)
```

---

vuong *Non-nested model tests*

Description

Perform Vuong's (1989) or Clarke's (2007) test for non-nested model selection.

Usage

```
vuong(model1, model2, outcome1=NULL, outcome2=NULL, level=0.05,
digits=2)

clarke(model1, model2, outcome1=NULL, outcome2=NULL, level=0.05, digits=2)
```

Arguments

model1	A fitted statistical model of class "game", "lm", or "glm"
model2	A fitted statistical model of class "game", "lm", or "glm" whose dependent variable is the same as that of model1
outcome1	Optional: if model1 is of class "game", specify an integer to restrict attention to a particular binary outcome (the corresponding column of predict(model1)). For ultimatum models, "offer" or "accept" may also be used. See "Details" below for more information on when to specify an outcome. If model1 is not of class "game" and outcome1 is non-NULL, it will be ignored and a warning will be issued.
outcome2	Optional: same as outcome1, but corresponding to model2.
level	Numeric: significance level for the test.
digits	Integer: number of digits to print

Details

These tests are for comparing two statistical models that have the same dependent variable, where neither model can be expressed as a special case of the other (i.e., they are non-nested). The null hypothesis is that the estimated models are the same Kullback-Leibler distance from the true model. To adjust for potential differences in the dimensionality of the models, the test statistic for both `vuong` and `clarke` is corrected using the Bayesian information criterion (see Clarke 2007 for details).

It is crucial that the dependent variable be exactly the same between the two models being tested, including the order the observations are placed in. The `vuong` and `clarke` functions check for such discrepancies, and stop with an error if any is found. Models with non-null `weights` are not yet supported.

When comparing a strategic model to a (generalized) linear model, you must take care to ensure that the dependent variable is truly the same between models. This is where the `outcome` arguments come into play. For example, in an `ultimatum` model where acceptance is observed, the dependent variable for each observation is the vector consisting of the offer size and an indicator for whether it was accepted. This is not the same as the dependent variable in a least-squares regression of offer size, which is a scalar for each observation. Therefore, for a proper comparison of `model1` of class "ultimatum" and `model2` of class `"lm"`, it is necessary to specify `outcome1 = "offer"`. Similarly, consider an `egame12` model on the `war1800` data, where player 1 chooses whether to escalate the crisis and player 2 chooses whether to go to war. The dependent variable for each observation in this model is the vector of each player's choice. By contrast, in a logistic regression where the dependent variable is whether war occurs, the dependent variable for each observation is a scalar. To compare these models, it is necessary to specify `outcome1 = 3`.

Value

Typical use will be to run the function interactively and examine the printed output. The functions return an object of class `"nonnest.test"`, which is a list containing:

`stat` The test statistic

`test` The type of test (`"vuong"` or `"clarke"`)

`level` Significance level for the test

`digits` Number of digits to print

`loglik1` Vector of observationwise log-likelihoods for `model1`

`loglik2` Vector of observationwise log-likelihoods for `model2`

`nparams` Integer vector containing the number of parameters fitted in `model1` and `model2` respectively

`nobs` Number of observations of the dependent variable being modeled

Author(s)

Brenton Kenkel (<brenton.kenkel@gmail.com>)

References

Quang H. Vuong. 1989. "Likelihood Ratio Tests for Model Selection and Non-Nested Hypotheses." *Econometrica* 57(2): 307–333.

Kevin Clarke. 2007. "A Simple Distribution-Free Test for Nonnested Hypotheses." *Political Analysis* 15(3): 347–363.

Examples

```
data("war1800")

## Balance of power model
f1 <- esc + war ~ balanc + s_wt_re1 | 0 | balanc | balanc + s_wt_re1
m1 <- egame12(f1, data = war1800, subset = !is.na(regime1) & !is.na(regime2))

## Regime type model
f2 <- esc + war ~ regime1 | 0 | regime1 + regime2 | regime1 + regime2
m2 <- egame12(f2, data = war1800)

## Comparing two strategic models
vuong(model1 = m1, model2 = m2)
clarke(model1 = m1, model2 = m2)

## Comparing strategic model to logit - must specify `outcome1` appropriately
logit1 <- glm(war ~ balanc + s_wt_re1, data = m1$model, family=binomial)
vuong(model1 = m1, outcome1 = 3, model2 = logit1)
clarke(model1 = m1, outcome1 = 3, model2 = logit1)

logit2 <- glm(sq ~ regime1 + regime2, data = war1800, family=binomial)
vuong(model1 = m2, outcome1 = 1, model2 = logit2)
clarke(model1 = m2, outcome1 = 1, model2 = logit2)

## Ultimatum model
data(data_ult)
f3 <- offer + accept ~ w1 + w2 + x1 + x2 | w1 + w2 + z1 + z2
m3 <- ultimatum(f3, maxOffer = 15, data = data_ult)
ols1 <- lm(offer ~ w1 + w2 + x1 + x2 + z1 + z2, data = data_ult)
vuong(model1 = m3, outcome1 = "offer", model2 = ols1)
clarke(model1 = m3, outcome1 = "offer", model2 = ols1)
```

---

war1800 *19th-century international disputes*

Description

Dataset of militarized international disputes between 1816 and 1899.

Usage

```
data(war1800)
```

Details

The dataset is taken from the Correlates of War project. The unit of observation is the dyad-year, and the variables are:

`ccode1` Initiator's COW country code

`ccode2` Respondent's COW country code

`year` Year of dispute

`cap_1` Initiator's military capabilities (as percent of total system capabilities)

`cap_2` Respondent's military capabilities (as percent of total system capabilities)

`balanc` Balance of dyadic capabilities possessed by the initiator (i.e., `cap_1 / (cap_1 + cap_2)`)

`s_wt_re1` Dyadic S-score (see Signorino and Ritter 1998), weighted by initiator's region

`s_wt_re2` Dyadic S-score, weighted by respondent's region

`dem1` Initiator's Polity score

`dem2` Respondent's Polity score

`distance` Distance (in miles) between initiator and respondent

`peaceyrs` Years since last dispute in this dyad

`midnum` Dispute's number in the MID data set

`revis1` Whether the initiator had "revisionist" aims

`revis2` Whether the respondent had "revisionist" aims

`sq` Indicator for status quo outcome

`capit` Indicator for capitulation outcome

`war` Indicator for war outcome

`esc` Indicator for escalation (i.e., either capitulation or war occurs)

`regime1` Initiator's regime type (calculated from `dem1`)

`regime2` Respondent's regime type (calculated from `dem2`)

References

Daniel M. Jones, Stuart A. Bremer and J. David Singer. 1996. "Militarized Interstate Disputes, 1816-1992: Rationale, Coding Rules, and Empirical Patterns." *Conflict Management and Peace Science* 15(2): 163–213.

See Also

`egame12`

Index