Package 'pscl'

March 7, 2020

Version 1.5.5

Date 2020-02-25

Title Political Science Computational Laboratory

Author Simon Jackman, with contributions from
Alex Tahk, Achim Zeileis, Christina Maimone, Jim Fearon and Zoe Meers

Maintainer Simon Jackman <simon.jackman@sydney.edu.au>

Imports MASS, datasets, grDevices, graphics, stats, utils

Suggests lattice, MCMCpack, car, lmtest, sandwich, zoo, coda, vcd, mvtnorm, mgcv

Description Bayesian analysis of item-response theory (IRT) models, roll call analysis; computing highest density regions; maximum likelihood estimation of zero-inflated and hurdle models for count data; goodness-of-fit measures for GLMs; data sets used in writing and teaching at the Political Science Computational Laboratory; seats-votes curves.

LazyData true

License GPL-2

URL http://github.com/atahk/pscl

NeedsCompilation yes

RoxygenNote 6.0.1

Repository CRAN

Date/Publication 2020-03-07 12:00:02 UTC

R topics documented:

`absentee` *Absentee and Machine Ballots in Pennsylvania State Senate Races*

Description

Absentee ballot outcomes contrasted with machine ballots, cast in Pennsylvania State Senate elections, selected districts, 1982-1993.

Usage

```
data(absentee)
```

Format

A data frame with 22 observations on the following 8 variables.

`year` a numeric vector, year of election, 19xx

`district` a numeric vector, Pennsylvania State Senate district

`absdem` a numeric vector, absentee ballots cast for the Democratic candidate

`absrep` a numeric vector, absentee ballots cast for the Republican candidate

`machdem` a numeric vector, votes cast on voting machines for the Democratic candidate

`machrep` a numeric vector, votes cast on voting machines for the Republican candidate

`dabs` a numeric vector, Democratic margin among absentee ballots

`dmach` a numeric vector, Democratic margin among ballots case on voting machines

Details

In November 1993, the state of Pennsylvania conducted elections for its state legislature. The result in the Senate election in the 2nd district (based in Philadelphia) was challenged in court, and ultimately overturned. The Democratic candidate won 19,127 of the votes cast by voting machine, while the Republican won 19,691 votes cast by voting machine, giving the Republican a lead of 564 votes. However, the Democrat won 1,396 absentee ballots, while the Republican won just 371 absentee ballots, more than offsetting the Republican lead based on the votes recorded by machines on election day. The Republican candidate sued, claiming that many of the absentee ballots were fraudulent. The judge in the case solicited expert analysis from Orley Ashenfelter, an economist at Princeton University. Ashenfelter examined the relationship between absentee vote margins and machine vote margins in 21 previous Pennsylvania Senate elections in seven districts in the Philadelphia area over the preceding decade.

Source

Ashenfelter, Orley. 1994. Report on Expected Absentee Ballots. Typescript. Department of Economics, Princeton University.

References

Ashenfelter, Orley, Phillip Levine and David Zimmerman. 2003. *Statistics and Econometrics: Methods and Applications*. New York: John Wiley and Sons.

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Hoboken, New Jersey. Examples 2.13, 2.14, 2.15.

Examples

```
data(absentee)
summary(absentee)

denom <- absentee$absdem + absentee$absrep
y <- (absentee$absdem - absentee$absrep)/denom * 100
denom <- absentee$machdem + absentee$machrep
x <- (absentee$machdem - absentee$machrep)/denom *100

ols <- lm(y ~ x,
          subset=c(rep(TRUE,21),FALSE)  ## drop data point 22
          )

## predictions for disputed absentee point
yhat22 <- predict(ols,
                  newdata=list(x=x[22]),
                  se.fit=TRUE,
                  interval="prediction")
tstat <- (y[22]-yhat22$fit[,"fit"])/yhat22$se.fit
cat("tstat on actual outcome for obs 22:",tstat,"\n")
cat(paste("Pr(t>",round(tstat,2),") i.e., one-sided:\n",sep=""))
cat(1-pt(tstat,df=yhat22$df),"\n")

## make a picture
xseq <- seq(min(x)-.1*diff(range(x)),
            max(x)+.1*diff(range(x)),
            length=100)
yhat <- predict(ols,interval="prediction",
                newdata=list(x=xseq))
plot(y~x,
     type="n",
     axes=FALSE,
     ylim=range(yhat,y),
     xlim=range(xseq),xaxs="i",
     xlab="Democratic Margin, Machine Ballots (Percentage Points)",
     ylab="Democratic Margin, Absentee Ballots (Percentage Points)")
polygon(x=c(xseq,rev(xseq)),  ## overlay 95% prediction CI
        y=c(yhat[,"lwr"],rev(yhat[,"upr"])),
        border=FALSE,
```

```
        col=gray(.85))
abline(ols,lwd=2)           ## overlay ols
points(x[-22],y[-22],pch=1) ## data
points(x[22],y[22],pch=16)  ## disputed data point

text(x[22],y[22],
     "Disputed\nElection",
     cex=.75,
     adj=1.25)
axis(1)
axis(2)
```

---

admit *Applications to a Political Science PhD Program*

Description

Ordinal ratings (faculty evaluations) of applicants to a Political Science PhD Program.

Usage

```
data(admit)
```

Format

A data frame with 106 observations on the following 6 variables.

`score` an ordered factor with levels 1 < 2 < 3 < 4 < 5

`gre.quant` applicant's score on the quantitative section of the GRE; the maximum score is 800

`gre.verbal` applicant's score on the verbal section of the GRE; the maximum score is 800

`ap` 1 if the applicant indicated an interest in American politics; 0 otherwise

`pt` 1 if the applicant indicated an interest in Political Theory; 0 otherwise

`female` 1 for female applicants; 0 otherwise

References

Jackman, Simon. 2004. "What Do We Learn From Graduate Admissions Committees?: A Multiple-Rater, Latent Variable Model, with Incomplete Discrete and Continuous Indicators." *Political Analysis*. 12(4):400-424.

Examples

```
data(admit)
summary(admit)
## ordered probit model
op1 <- MASS::polr(score ~ gre.quant + gre.verbal + ap + pt + female,
            Hess=TRUE,
            data=admit,
            method="probit")
summary(op1)
hitmiss(op1)
logLik(op1)
pR2(op1)
```

AustralianElectionPolling

Political opinion polls in Australia, 2004-07

Description

The results of 239 published opinion polls measuring vote intentions (1st preference vote intention in a House of Representatives election) between the 2004 and 2007 Australian Federal elections, from 4 survey houses.

Usage

```
data(AustralianElectionPolling)
```

Format

A data frame with 239 observations on the following 14 variables.

`ALP` a numeric vector, percentage of respondents reported as intending to vote for the Australian Labor Party

`Lib` a numeric vector, percentage of respondents reported as intending to vote for the Liberal Party

`Nat` a numeric vector, percentage of respondents reported as intending to vote for the National Party

`Green` a numeric vector, percentage of respondents reported as intending to vote for the Greens

`FamilyFirst` a numeric vector, percentage of respondents reported as intending to vote for the Family First party

`Dems` a numeric vector, percentage of respondents reported as intending to vote for the Australian Democrats

`OneNation` a numeric vector, percentage of respondents reported as intending to vote for One Nation

`DK` a numeric vector, percentage of respondents reported as expressing no preference or a "don't know" response

`sampleSize` a numeric vector, reported sample size of the poll

`org` a factor with levels `Galaxy`, `Morgan,F2F`, `Newspoll`, `Nielsen` and `Morgan,Phone`, indicating the survey house and/or mode of the poll

`startDate` a Date, reported start of the field period

`endDate` a Date, reported end of the field period

`source` a character vector, source of the poll report

`remark` a character vector, remarks noted by author and/or research assistant coders

Details

Morgan uses two modes: phone and face-to-face.

The 2004 Australian election was on October 9; the ALP won 37.6% of the 1st preferences cast in elections for the House of Representatives. The ALP won the 2007 election (November 24) with 43.4% of 1st preferences.

The ALP changed leaders twice in the 2004-07 inter-election period spanned by these data: (1) Mark Latham resigned the ALP leadership on January 18 2005 and was replaced by Kim Beazley; (2) Beazley lost the ALP leadership to Kevin Rudd on December 4, 2006.

The then Prime Minister, John Howard, announced the November 2007 election on October 14, 2007.

Source

See the `source` variable. Andrea Abel assisted with the data collection.

References

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Hoboken, New Jersey. Example 9.3.

Examples

```
data(AustralianElectionPolling)
if(require(lattice)) {
    lattice::xyplot(ALP ~ startDate | org,
       data=AustralianElectionPolling,
       layout=c(1,5),
       type="b",
       xlab="Start Date",
       ylab="ALP")
}

## test for house effects
y <- AustralianElectionPolling$ALP/100
v <- y*(1-y)/AustralianElectionPolling$sampleSize
w <- 1/v
m1 <- mgcv::gam(y ~ s(as.numeric(startDate)),
          weight=w,
          data=AustralianElectionPolling)
m2 <- update(m1, ~ . + org)
anova(m1,m2)
```

`AustralianElections` *elections to Australian House of Representatives, 1949-2016*

Description

Aggregate data on the 24 elections to Australia's House of Representatives, 1949 to 2016.

Usage

```
data(AustralianElections)
```

Format

A data frame with the following variables:

`date` date of election, stored using the `Date` class

`Seats` numeric, number of seats in the House of Representatives

`Uncontested` numeric, number of uncontested seats

`ALPSeats` numeric, number of seats won by the Australian Labor Party

`LPSeats` numeric, number of seats won by the Liberal Party

`NPSeats` numeric, number of seats won by the National Party (previously known as the Country Party)

`OtherSeats` numeric, number of seats won by other parties and/or independent candidates

`ALP` numeric, percentage of first preference votes cast for Australian Labor Party candidates

`ALP2PP` numeric, percentage of the two-party preferred vote won by Australian Labor Party candidates

`LP` numeric, percent of first preference votes cast for Liberal Party candidates

`NP` numeric, percent of first preference votes cast for National Party (Country Party) candidates

`DLP` numeric, percent of first preference votes cast for Democratic Labor Party candidates

`Dem` numeric, percent of first preference votes cast for Australian Democrat candidates

`Green` numeric, percent of first preference votes cast for Green Party candidates

`Hanson` numeric, percent of first preference votes cast for candidates from Pauline Hanson's One Nation party

`Com` numeric, percent of first preference votes cast for Communist Party candidates

`AP` numeric, percent of first preference votes cast for Australia Party candidates

`Informal` numeric, percent of ballots cast that are spoiled, blank, or otherwise uncountable (usually because of errors in enumerating preferences)

`Turnout` numeric, percent of enrolled voters recorded as having turned out to vote (Australia has compulsory voting)

Note

The Liberal National Party of Queensland formed in 2008 after a merger of the Liberal Party and the National Party. In all elections following 2008, they have been categorised under LP.

Source

Australian Electoral Commission. http://www.aec.gov.au.

References

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Hoboken, New Jersey. Example 3.5.

Examples

```
data(AustralianElections)
attach(AustralianElections)
alpSeatShare <- ALPSeats/Seats
alpVoteShare <- ALP2PP/100

## log-odds transforms
x <- log(alpVoteShare/(1-alpVoteShare))
y <- log(alpSeatShare/(1-alpSeatShare))

ols <- lm(y~x)   ## Tufte-style seats-votes regression

xseq <- seq(-4.5,4.5,length=500)
yhat <- coef(ols)[1] + coef(ols)[2]*xseq
yhat <- exp(yhat)/(1+exp(yhat))
xseq <- exp(xseq)/(1+exp(xseq))

## seats vote curve
plot(x=alpVoteShare,
     y=alpSeatShare,
     xlab="ALP Vote Share",
     ylab="ALP Seat Share")
lines(xseq,yhat,lwd=2)
abline(h=.5,lty=2)
abline(v=.5,lty=2)
```

---

betaHPD *compute and optionally plot beta HDRs*

Description

Compute and optionally plot highest density regions for the Beta distribution.

Usage

```
betaHPD(alpha,beta,p=.95,plot=FALSE,xlim=NULL,debug=FALSE)
```

Arguments

alpha	scalar, first shape parameter of the Beta density. Must be greater than 1, see details
beta	scalar, second shape parameter of the Beta density. Must be greater than 1, see details
p	scalar, content of HPD, must lie between 0 and 1
plot	logical flag, if TRUE then plot the density and show the HDR
xlim	numeric vector of length 2, the limits of the density's support to show when plotting; the default is NULL, in which case the function will confine plotting to where the density is non-negligible
debug	logical flag, if TRUE produce messages to the console

Details

The Beta density arises frequently in Bayesian models of binary events, rates, and proportions, which take on values in the open unit interval. For instance, the Beta density is a conjugate prior for the unknown success probability in binomial trials. With shape parameters $\alpha > 1$ and $\beta > 1$, the Beta density is unimodal.

In general, suppose $\theta \in \Theta \subseteq R^k$ is a random variable with density $f(\theta)$. A highest density region (HDR) of $f(\theta)$ with content $p \in (0, 1]$ is a set $\mathcal{Q} \subseteq \Theta$ with the following properties:

$$\int_{\mathcal{Q}} f(\theta) d\theta = p$$

and

$$f(\theta) > f(\theta^*) \, \forall \, \theta \in \mathcal{Q}, \theta^* \notin \mathcal{Q}.$$

For a unimodal Beta density (the class of Beta densities handled by this function), a HDR of content $0 < p < 1$ is simply an interval $\mathcal{Q} \in (0, 1)$.

This function uses numerical methods to solve for the end points of a HDR for a Beta density with user-specified shape parameters, via repeated calls to the functions dbeta, pbeta and qbeta. The function optimize is used to find points v and w such that

$$f(v) = f(w)$$

subject to the constraint

$$\int_v^w f(\theta; \alpha, \beta) d\theta = p,$$

where $f(\theta; \alpha, \beta)$ is a Beta density with shape parameters α and β.

In the special case of $\alpha = \beta > 1$, the end points of a HDR with content p are given by the $(1 \pm p)/2$ quantiles of the Beta density, and are computed with the qbeta function.

Again note that the function will only compute a HDR for a unimodal Beta density, and exit with an error if alpha<=1 | beta <=1. Note that the uniform density results with $\alpha = \beta = 1$, which does not have a unique HDR with content $0 < p < 1$. With shape parameters $\alpha < 1$ and $\beta > 1$ (or vice-versa, respectively), the Beta density is infinite at 0 (or 1, respectively), but still integrates to one, and so a HDR is still well-defined (but not implemented here, at least not yet). Similarly, with $0 < \alpha, \beta < 1$ the Beta density is infinite at both 0 and 1, but integrates to one, and again a HDR of content $p < 1$ is well-defined in this case, but will be a set of two disjoint intervals (again, at present, this function does not cover this case).

Value

If the numerical optimization is successful an vector of length 2, containing v and w, defined above. If the optimization fails for whatever reason, a vector of NAs is returned.

The function will also produce a plot of the density with area under the density supported by the HDR shaded, if the user calls the function with plot=TRUE; the plot will appear on the current graphics device.

Debugging messages are printed to the console if the debug logical flag is set to TRUE.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>. Thanks to John Bullock who discovered a bug in an earlier version.

See Also

pbeta, qbeta, dbeta, uniroot

Examples

```
betaHPD(4,5)
betaHPD(2,120)
betaHPD(120,45,p=.75,xlim=c(0,1))
```

---

bioChemists *article production by graduate students in biochemistry Ph.D. programs*

Description

A sample of 915 biochemistry graduate students.

Usage

```
data(bioChemists)
```

Format

art count of articles produced during last 3 years of Ph.D.

fem factor indicating gender of student, with levels Men and Women

mar factor indicating marital status of student, with levels Single and Married

kid5 number of children aged 5 or younger

phd prestige of Ph.D. department

ment count of articles produced by Ph.D. mentor during last 3 years

References

Long, J. Scott. 1990. The origins of sex differences in science. *Social Forces*. 68(3):1297-1316.

Long, J. Scott. 1997. *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks, California: Sage.

`ca2006` *California Congressional Districts in 2006*

Description

Election returns and identifying information, California's 53 congressional districts in the 2006 Congressional elections.

Usage

```
data(ca2006)
```

Format

A data frame with 53 observations on the following 11 variables.

`district` numeric, number of Congressional district

`D` numeric, number of votes for the Democratic candidate

`R` numeric, votes for the Republican candidate

`Other` numeric, votes for other candidates

`IncParty` character, party of the incumbent (or retiring member), `D` or `R`

`IncName` character, last name of the incumbent, character `NA` if no incumbent running

`open` logical, `TRUE` if no incumbent running

`contested` logical, `TRUE` if both major parties ran candidates

`Bush2004` numeric, votes for George W. Bush (R) in the district in the 2004 presidential election

`Kerry2004` numeric, votes for John Kerry (D) in 2004

`Other2004` numeric votes for other candidates in 2004

`Bush2000` numeric, votes for George W. Bush in 2000

`Gore2000` numeric, votes for Al Gore (D) in 2000

Source

2006 data from the California Secretary of State's web site, http://www.sos.ca.gov/elections/prior-elections/statewide-election-results/general-election-november-7-2006/statement-vote/.

2004 and 2000 presidential vote in congressional districts from the 2006 *Almanac of American Politics*.

Thanks to Arthur Aguirre for the updated links, above.

References

Michael Baraon and Richard E. Cohen. 2006. *The Almanac of American Politics, 2006.* National Journal Group: Washington, D.C.

Examples

```
data(ca2006)

## 2006 CA congressional vote against 2004 pvote
y <- ca2006$D/(ca2006$D+ca2006$R)
x <- ca2006$Kerry2004/(ca2006$Kerry2004+ca2006$Bush2004)

pch <- rep(19,length(y))
pch[ca2006$open] <- 1
col <- rep("black",length(y))
col[11] <- "red"    ## Pembo (R) loses to McNerney (D)
plot(y~x,pch=pch,
     col=col,
     xlim=range(x,y,na.rm=TRUE),
     ylim=range(x,y,na.rm=TRUE),
     xlab="Kerry Two-Party Vote, 2004",
     ylab="Democratic Two-Party Vote Share, 2006")
abline(0,1)
abline(h=.5,lty=2)
abline(v=.5,lty=2)
legend(x="topleft",
       bty="n",
       col=c("red","black","black"),
       pch=c(19,19,1),
       legend=c("Seat Changing Hands",
         "Seat Retained by Incumbent Party",
         "Open Seat (no incumbent running)")
       )
```

`computeMargins` *add information about voting outcomes to a rollcall object*

Description

Add summaries of each roll call vote to a `rollcall` object.

Usage

```
computeMargins(object, dropList = NULL)
```

Arguments

`object`	an object of class `rollcall`
`dropList`	a `list` (or `alist`) listing voting decisions, legislators and/or votes to be dropped from the analysis; see `dropRollCall` for details.

Details

The subsetting implied by the dropList is first applied to the rollcall object, via dropRollCall. Then, for each remaining roll call vote, the number of legislators voting "Yea", "Nay", and not voting are computed, using the encoding information in the codes component of the rollcall object via the convertCodes function. The matrix of vote counts are added to the rollcall object as a component voteMargins.

Value

An object of class rollcall, with a component voteMargins that is a matrix with four columns:

Yea	number of legislators voting "Yea"
Nay	number of legislators voting "Nay"
NA	number of legislators not voting "Nay"
Min	the number of legislators voting on the losing side of the roll call

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

dropRollCall on specifying a dropList. The vote-specific marginals produced by this function are used by as dropRollCall, summary.ideal and predict.ideal.

Examples

```
data(s109)
tmp <- computeMargins(s109)
dim(tmp$voteMargins)   ## 645 by 4

tmp <- computeMargins(s109,
                      dropList=list(codes="notInLegis",lop=0))
dim(tmp$voteMargins)   ## 544 by 4
```

---

constrain.items *constrain item parameters in analysis of roll call data*

Description

Sets constraints on specified item parameters in Bayesian analysis of roll call data by generating appropriate priors and start values for Markov chain Monte Carlo iterations.

Usage

```
constrain.items(obj, dropList = list(codes = "notInLegis", lop = 0),
                x, d = 1)
```

Arguments

obj	an object of class rollcall.
dropList	a list (or alist) indicating which voting decisions, legislators and/or roll calls are to be excluded from the subsequent analysis; see dropRollCall for details.
x	a list containing elements with names matching votes found in dimnames(object$votes)[[2]] (but after any subsetting specified by dropList). Each component of the list must be a vector containing d elements, specifying the value to which the item discrimination parameters should be constrained, in each of the d dimensions. The intercept or item difficultly parameter will not be constrained.
d	numeric, positive integer, the number of dimensions for which to set up the priors and start values.

Details

constrain.items and its cousin, constrain.legis are usefully thought of as "pre-processor" functions, generating priors *and* start values for both the item parameters and the ideal points. For the items specified in x, the prior mean for each dimension is set to the value given in x, and the prior precision for each dimension is set to 1e12 (i.e., a near-degenerate "spike" prior). For the other items, the priors are set to a mean of 0 and precision 0.01. All of the ideal points are given normal priors with mean 0, precision 1.

Start values are also generated for both ideal points and item parameters. The start values for the items specified in x are set to the values specified in x. The list resulting from constrain.items can then be given as the value for the parameters priors and startvals when ideal is run. The user is responsible for ensuring that a sufficient number of items are constrained such that when ideal is run, the model parameters are identified.

dropRollCall is first called to generate the desired roll call matrix. The entries of the roll call matrix are mapped to c(0,1,NA) using the codes component of the rollcall object. See the discussion in the documentation of ideal for details on the generation of start values.

Value

a list with elements:

xp	prior means for ideal points. A matrix of dimensions number of legislators in obj by d.
xpv	prior meansprecisions for ideal points. A matrix of dimensions number of legislators in obj by d.
bp	prior means for item parameters. A matrix of dimensions number of items or votes in obj by d+1.
bpv	prior meansprecisions for item parameters. A matrix of dimensions number of items or votes in obj by d+1.
xstart	start values for ideal points. A matrix of dimensions number of legislators in obj by d.
bstart	start values for ideal points. A matrix of dimensions number of items or votes in obj by d+1.

See Also

rollcall, ideal, constrain.legis

Examples

```
## Not run:
data(s109)
f <- system.file("extdata","id1.rda",package="pscl")
load(f)
id1sum <- summary(id1,include.beta=TRUE)
suspect1 <- id1sum$bSig[[1]]=="95
close60 <- id1sum$bResults[[1]][,"Yea"] < 60
close40 <- id1sum$bResults[[1]][,"Yea"] > 40
suspect <- suspect1 & close60 & close40
id1sum$bResults[[1]][suspect,]
suspectVotes <- dimnames(id1sum$bResults[[1]][suspect,])[[1]]


## constraints on 2d model,
## close rollcall poorly fit by 1d model
## serves as reference item for 2nd dimension

cl <- constrain.items(s109,
                      x=list("2-150"=c(0,7),
                        "2-169"=c(7,0)),
                      d=2)

id1Constrained <- ideal(s109,
                        d=2,
                        meanzero=TRUE,
                        priors=cl,
                        startvals=cl,
                        maxiter=1e5,
                        burnin=1e3,
                        thin=1e2)
summary(id1Constrained,include.beta=TRUE)

## End(Not run)
```

`constrain.legis` *constrain legislators' ideal points in analysis of roll call data*

Description

Sets constraints on specified legislators for ideal point estimation by generating appropriate priors and start values.

Usage

```
constrain.legis(obj, dropList = list(codes = "notInLegis", lop = 0),
                x, d = 1)
```

Arguments

obj	an object of class rollcall.
dropList	a list (or alist) indicating which voting decisions, legislators and/or roll calls are to be excluded from the subsequent analysis; see dropRollCall for details.
x	a list containing elements with names partially matching legislators found in dimnames(object$votes)[[1]] (but after any sub-setting specified by dropList). Each element must be a vector containing d elements, specifying the value to which the ideal point should be constrained in each of d dimensions. x must have at least d+1 components; i.e., supplying a necessary (but not sufficient) set of constraints for global identification of the parameters of a d-dimensional item-response model, see Details.
d	the number of dimensions for which to set up the priors and start values.

Details

constrain.items and its cousin, constrain.legis are usefully thought of as "pre-processor" functions, implementing identification constraints for the ideal point model by generating priors *and* start values for both the item parameters and the ideal points.

For the legislators specified in x, the prior mean for each dimension is set to the specified value and the prior precision for each dimension is set to 1e12 (i.e., a near-degenerate "spike" prior, and, for all practical purposes, constraining that parameter to a fixed value). For the other legislators, the priors on their ideal points are set to a mean of 0 and a small precision of .01, corresponding to a prior variance of 100, or a prior 95 percent confidence interval of -20 to 20. All of the item parameter priors are set to mean 0, precision 0.01.

Start values are also generated for both ideal points and item parameters. The start values for the legislators named in x are set to the values specified in x. The list resulting from constrain.legis can then be given as the value for the parameters priors and startvals when ideal is run. constrain.legis requires that d+1 constraints be specified; if the constrained ideal points points are linearly independent, then the parameters of the item-response model are (at least locally) identified. For instance, when fitting a 1 dimensional model, constraining the ideal points of two legislators is sufficient to globally identify the model parameters.

dropRollCall is first called to generate the desired roll call matrix. The entries of the roll call matrix are mapped to c(0,1,NA) using the codes component of the rollcall object. See the discussion in the documentation of ideal for details on the generation of start values.

Value

a list with elements:

xp	prior means for ideal points. A matrix of dimensions number of legislators in rc by d.

xpv	prior meansprecisions for ideal points. A matrix of dimensions number of legislators in rc by d.
bp	prior means for item parameters. A matrix of dimensions number of items or votes in rc by d+1.
bpv	prior meansprecisions for item parameters. A matrix of dimensions number of items or votes in rc by d+1.
x	start values for ideal points. A matrix of dimensions number of legislators in rc by d.
b	start values for ideal points. A matrix of dimensions number of items or votes in rc by d+1.

See Also

rollcall, ideal, constrain.items. See pmatch on how supplied names are matched against the names in the rollcall object.

Examples

```
data(s109)
cl <- constrain.legis(s109,
                      x=list("KENNEDY"=-1,
                        "ENZI"=1),
                      d=1)

## Not run:
## too long for examples
id1Constrained <- ideal(s109,
                        d=1,
                        priors=cl,      ## use cl
                        startvals=cl,   ## use cl
                        maxiter=5000,
                        burnin=500,
                        thin=25)
summary(id1Constrained)

cl2 <- constrain.legis(s109,
                       x=list("KENNEDY"=c(-1,0),
                         "ENZI"=c(1,0),
                         "CHAFEE"=c(0,-.5)),
                       d=2)


id2Constrained <- ideal(s109,
                        d=2,
                        priors=cl2,      ## priors (w constraints)
                        startvals=cl2,   ## start value (w constraints)
                        store.item=TRUE,
                        maxiter=5000,
                        burnin=500,
                        thin=25)
```

```
summary(id2Constrained,include.items=TRUE)

## End(Not run)
```

---

convertCodes *convert entries in a rollcall matrix to binary form*

Description

Convert roll call matrix to binary form using encoding information.

Usage

```
convertCodes(object, codes = object$codes)
```

Arguments

object	rollcall object
codes	list, mapping entries in the votes component of rollcall object to 0 ('Nay'), 1 ('Yea') and NA (missing, abstentions, etc). Defaults to the codes component of the rollcall object.

Details

See rollcall for details on the form of the codes list.

Value

a matrix with dimensions equal to the dimensions of the votes component of the rollcall object.

Note

Any entries in the votes matrix that can not be mapped into c(0,1,NA) using the information in codes are mapped to NA, with an informative message sent to the console.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

rollcall

Examples

```
data(s109)
mat <- convertCodes(s109)
table(mat,exclude=NULL)
```

dropRollCall *drop user-specified elements from a rollcall object*

Description

Drop user-specified elements of rollcall object, returning a roll call object.

Usage

```
dropRollCall(object, dropList,debug=FALSE)
```

Arguments

object	an object of class rollcall
dropList	a list (or alist) with some (or all) of the following components:
	codes character or numeric, possibly a vector. If character, it should match the names of object$codes, indicating the set of entries in object$votes to be set to NA. If numeric, then codes indicates the entries in object$votes that will be set to NA.
	lop numeric, non-negative integer, less than number of legislators represented in object. Roll calls with lop or fewer legislators voting in the minority are dropped.
	legisMin numeric, non-negative integer, less than number of roll calls represented in object. Legislators with legisMin or fewer votes are dropped.
	dropLegis an expression that evaluates to mode logical, vector of length equal to the number of legislators represented in object. The expression is evaluated in the legis.data component of the rollcall object. Legislators for whom the expression evaluates to TRUE are dropped.
	dropVotes an expression that evaluates to mode logical, vector of length equal to the number of rollcalls represented in object. The expression is evaluated in the vote.data component of the rollcall object. Rollcalls for which the expression evaluates to TRUE are dropped.
debug	logical, set to TRUE to see messages printed to the console as inspection and subsetting of the rollcall object takes place

Details

It is often desirable to restrict the analysis of roll call data in various ways. For one thing, unanimous votes provide no information discriminating among legislators: hence, summary and analysis should almost always use dropList=list(lop=0). See the examples for other possibilities, limited only by the information supplied in legis.data and votes.data.

Value

An object of class rollcall with components modified/added by the subsetting indicated in the dropList.

Note

With the exception of codes, each component of dropList generates a vector of mode logical, either with respect to legislators or votes. These logical vectors are then combined element-wise, such that if any one of the subsetting restrictions is TRUE for a particular legislator or vote, then that legislator or vote is dropped. Some summaries are reported to the console along the way if debug=TRUE.

dropRollCall adds a component named dropInfo to the rollcall object it returns. This component is itself a list containing named components

legislators a vector of mode logical, with each element TRUE if the legislator is retained in the returned rollcall object.

votes a vector of mode logical, with each element TRUE if the corresponding is retained in the returned rollcall object.

dropList the dropList supplied as input to dropRollCall.

If the input rollcall object is itself the product of a call to dropRollCall, the dropInfo component on output is a list with named components

previous the dropInfo component of the input rollcall object.

new the dropInfo list created by the current call to dropRollCall.

Functions like summary.rollcall try to handle this information sensibly.

When dropList uses the dropLegis or dropVotes components then dropList should be constructed via the alist command; this ensures that the dropLegis and dropVotes components of dropList are objects of mode expression, and evaluated to mode logical in the legis.data and vote.data environments by the function, if possible (rather than being evaluated immediately in the environment calling dropRollCall or constructing dropList). See the examples. This is not entirely satisfactory, and behavior more like the subset argument in function lm would be preferable.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

dropUnanimous, summary.rollcall, ideal, alist.

Examples

```
data(s109)
s109.working <- dropRollCall(s109,
                             dropList=list(lop=0))
summary(s109.working)

s109.working <- dropRollCall(s109,
                             dropList=list(lop=0,
                               code="notInLegis"))
summary(s109.working)
```

```
s109.working <- dropRollCall(s109,
                             dropList=list(lop=3,
                               code="notInLegis"))
summary(s109.working)

## note use of alist, since dropLegis is an expression
dropList <- alist(lop=3,
                  dropLegis=party!="D",
                  code="notInLegis")
s109.working <- dropRollCall(s109,dropList=dropList,debug=TRUE)
summary(s109.working)

s109.working <- dropRollCall(s109.working,dropList=list(legisMin=25))
summary(s109.working)


## Not run:
## read 102nd House from Poole web site
h102 <- readKH("ftp://voteview.ucsd.edu/dtaord/hou102kh.ord")

## drop President from roll call matrix
h102 <- dropRollCall(h102,
                     dropList=alist(dropLegis=state=="USA"))
summary(h102)

## End(Not run)
```

`dropUnanimous` *drop unanimous votes from rollcall objects and matrices*

Description

Drop unanimous votes from rollcall objects and rollcall matrices.

Usage

```
dropUnanimous(obj, lop = 0)
```

Arguments

`obj`	object, either of class `rollcall` or `matrix`
`lop`	numeric, non-negative integer, less than number of legislators represented in `obj`. Roll calls with `lop` or fewer legislators voting in the minority are dropped. Default is 0, meaning that unanimous votes are dropped.

Details

Unanimous votes are the equivalent of test items that all subjects score "correct" (or all subjects scores "incorrect"); since there is no variation among the legislators/subjects, these votes/items provide no information as to latent traits (ideology, preferences, ability). A reasonably large number of rollcalls in any contemporary U.S. Congress are unanimous.

Specific methods are provided for objects of class `rollcall` or `matrix`.

Value

A `rollcall` object or a `matrix` depending on the class of `object`.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

`dropRollCall`, `rollcall`, `summary.rollcall`, `ideal`

Examples

```
data(s109)
s109.working <- dropUnanimous(s109)
summary(s109.working)
```

EfronMorris *Batting Averages for 18 major league baseball players, 1970*

Description

Batting averages for 18 major league baseball players, first 45 at bats of the 1970 season.

Usage

```
data(EfronMorris)
```

Format

`name` character, name of player

`team` character, team of player, abbreviated

`league` character, National League or American League

`r` numeric, hits in 1st 45 at bats

`y` numeric, r/45, batting average over 1st 45 at bats

`n` numeric, number of at bats, remainder of 1970 season

`p` numeric, batting average over remainder of 1970 season

Source

Efron, Bradley and Carl Morris. 1975. Data Analysis Using Stein's Estimator and Its Generalizations. *Journal of the American Statistical Association*. 70:311-319.

Examples

```
data(EfronMorris)
attach(EfronMorris)
plot(p~y,
     xlim=range(p,y),
     ylim=range(p,y),
     xlab="Batting Average, 1st 45 at bats",
     ylab="Batting Average, Remainder of Season")
abline(0,1)
```

---

extractRollCallObject *return the roll call object used in fitting an ideal model*

Description

Given a fitted model of class ideal, return the rollcall object that was used in the model fitting (i.e., apply all subsetting and recoding implied by the dropList passed to ideal).

Usage

```
extractRollCallObject(object)
```

Arguments

object	an object of class ideal

Details

This function is used by many post-estimation commands that operate on objects of class ideal. The function inspects the call attribute of the ideal object, extracting the name of the rollcall object and the dropList, then hands them over to dropRollCall.

Value

An object of class rollcall

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

rollcall; see dropRollCall for details on the form of a dropList.

Examples

```
data(s109)
f = system.file("extdata","id1.rda",package="pscl")
load(f)
tmp <- extractRollCallObject(id1)
summary(tmp)
v <- convertCodes(tmp)         ## roll call matrix per se
```

`hitmiss` *Table of Actual Outcomes against Predicted Outcomes for discrete data models*

Description

Cross-tabulations of actual outcomes against predicted outcomes for discrete data models, with summary statistics such as percent correctly predicted (PCP) under fitted and null models. For models with binary responses (generalized linear models with `family=binomial`), the user can specific a classification threshold for the predicted probabilities.

Usage

```
hitmiss(obj, digits = max(3, getOption("digits") - 3), ...)

## S3 method for class 'glm'
hitmiss(obj,digits=max(3,getOption("digits")-3),
            ...,
            k=.5)
```

Arguments

`obj`	a fitted model object, such as a `glm` with `family=binomial`, a `polr` model for ordinal responses, or a `multinom` model for unordered/multinomial outcomes
`digits`	number of digits to display in on-screen output
`...`	additional arguments passed to or from other functions
`k`	classification threshold for binary models

Details

For models with binary responses, the user can specify a parameter $0 < k < 1$; if the predicted probabilities exceed this threshold then the model is deemed to have predicted y=1, and otherwise to have predicted y=0. Measures like percent correctly predicted are crude summaries of model fit; the cross-tabulation of actual against predicted is somewhat more informative, providing a little more insight as to where the model fits less well.

Value

For `hitmiss.glm`, a vector of length 3:

`pcp`	Percent Correctly Predicted
`pcp0`	Percent Correctly Predicted among y=0
`pcp1`	Percent Correctly Predicted among y=1

Note

To-do: The `glm` method should also handle binomial data presented as two-vector success/failures counts; and count data with `family=poisson`, the `glm.nb` models and `zeroinfl` and `hurdle` etc. We should also make the output a class with prettier print methods, i.e., save the cross-tabulation in the returned object etc.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

`pR2` for pseudo r-squared; `predict`; `extractAIC`. See also the **ROCR** package and the `lroc` function in the **epicalc** package for ROC computations for assessing binary classifications.

Examples

```
data(admit)
## ordered probit model
op1 <- MASS::polr(score ~ gre.quant + gre.verbal + ap + pt + female,
            Hess=TRUE,
            data=admit,
            method="probit")
hitmiss(op1)
```

`hurdle` *Hurdle Models for Count Data Regression*

Description

Fit hurdle regression models for count data via maximum likelihood.

Usage

```
hurdle(formula, data, subset, na.action, weights, offset,
  dist = c("poisson", "negbin", "geometric"),
  zero.dist = c("binomial", "poisson", "negbin", "geometric"),
  link = c("logit", "probit", "cloglog", "cauchit", "log"),
  control = hurdle.control(...),
  model = TRUE, y = TRUE, x = FALSE, ...)
```

Arguments

formula	symbolic description of the model, see details.
data, subset, na.action	arguments controlling formula processing via model.frame.
weights	optional numeric vector of weights.
offset	optional numeric vector with an a priori known component to be included in the linear predictor of the count model. See below for more information on offsets.
dist	character specification of count model family.
zero.dist	character specification of the zero hurdle model family.
link	character specification of link function in the binomial zero hurdle (only used if zero.dist = "binomial".
control	a list of control arguments specified via hurdle.control.
model, y, x	logicals. If TRUE the corresponding components of the fit (model frame, response, model matrix) are returned.
...	arguments passed to hurdle.control in the default setup.

Details

Hurdle count models are two-component models with a truncated count component for positive counts and a hurdle component that models the zero counts. Thus, unlike zero-inflation models, there are *not* two sources of zeros: the count model is only employed if the hurdle for modeling the occurrence of zeros is exceeded. The count model is typically a truncated Poisson or negative binomial regression (with log link). The geometric distribution is a special case of the negative binomial with size parameter equal to 1. For modeling the hurdle, either a binomial model can be employed or a censored count distribution. The outcome of the hurdle component of the model is the occurrence of a non-zero (positive) count. Thus, for most models, positive coefficients in the hurdle component indicate that an increase in the regressor increases the probability of a non-zero count. Binomial logit and censored geometric models as the hurdle part both lead to the same likelihood function and thus to the same coefficient estimates. A censored negative binomial model for the zero hurdle is only identified if there is at least one non-constant regressor with (true) coefficient different from zero (and if all coefficients are close to zero the model can be poorly conditioned).

The formula can be used to specify both components of the model: If a formula of type y ~ x1 + x2 is supplied, then the same regressors are employed in both components. This is equivalent to y ~ x1 + x2 | x1 + x2. Of course, a different set of regressors could be specified for the zero hurdle component, e.g., y ~ x1 + x2 | z1 + z2 + z3 giving the count data model y ~ x1 + x2 conditional on (|) the zero hurdle model y ~ z1 + z2 + z3.

Offsets can be specified in both parts of the model pertaining to count and zero hurdle model: y ~ x1 + offset(x2) | z1 + z2 + offset(z3), where x2 is used as an offset (i.e., with coefficient fixed to 1) in the count part and z3 analogously in the zero hurdle part. By the rule stated above y ~ x1 + offset(x2) is expanded to y ~ x1 + offset(x2) | x1 + offset(x2). Instead of using the offset() wrapper within the formula, the offset argument can also be employed which sets an offset only for the count model. Thus, formula = y ~ x1 and offset = x2 is equivalent to formula = y ~ x1 + offset(x2) | x1.

All parameters are estimated by maximum likelihood using optim, with control options set in hurdle.control. Starting values can be supplied, otherwise they are estimated by glm.fit (the

default). By default, the two components of the model are estimated separately using two optim calls. Standard errors are derived numerically using the Hessian matrix returned by optim. See hurdle.control for details.

The returned fitted model object is of class "hurdle" and is similar to fitted "glm" objects. For elements such as "coefficients" or "terms" a list is returned with elements for the zero and count components, respectively. For details see below.

A set of standard extractor functions for fitted model objects is available for objects of class "hurdle", including methods to the generic functions print, summary, coef, vcov, logLik, residuals, predict, fitted, terms, model.matrix. See predict.hurdle for more details on all methods.

Value

An object of class "hurdle", i.e., a list with components including

coefficients	a list with elements "count" and "zero" containing the coefficients from the respective models,
residuals	a vector of raw residuals (observed - fitted),
fitted.values	a vector of fitted means,
optim	a list (of lists) with the output(s) from the optim call(s) for minimizing the negative log-likelihood(s),
control	the control arguments passed to the optim call,
start	the starting values for the parameters passed to the optim call(s),
weights	the case weights used,
offset	a list with elements "count" and "zero" containing the offset vectors (if any) from the respective models,
n	number of observations (with weights > 0),
df.null	residual degrees of freedom for the null model (= n - 2),
df.residual	residual degrees of freedom for fitted model,
terms	a list with elements "count", "zero" and "full" containing the terms objects for the respective models,
theta	estimate of the additional θ parameter of the negative binomial model(s) (if negative binomial component is used),
SE.logtheta	standard error(s) for $\log(\theta)$,
loglik	log-likelihood of the fitted model,
vcov	covariance matrix of all coefficients in the model (derived from the Hessian of the optim output(s)),
dist	a list with elements "count" and "zero" with character strings describing the respective distributions used,
link	character string describing the link if a binomial zero hurdle model is used,
linkinv	the inverse link function corresponding to link,
converged	logical indicating successful convergence of optim,
call	the original function call,

formula	the original formula,
levels	levels of the categorical regressors,
contrasts	a list with elements "count" and "zero" containing the contrasts corresponding to levels from the respective models,
model	the full model frame (if model = TRUE),
y	the response count vector (if y = TRUE),
x	a list with elements "count" and "zero" containing the model matrices from the respective models (if x = TRUE).

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

References

Cameron, A. Colin and Pravin K. Trivedi. 1998. *Regression Analysis of Count Data*. New York: Cambridge University Press.

Cameron, A. Colin and Pravin K. Trivedi 2005. *Microeconometrics: Methods and Applications*. Cambridge: Cambridge University Press.

Mullahy, J. 1986. Specification and Testing of Some Modified Count Data Models. *Journal of Econometrics*. **33**:341–365.

Zeileis, Achim, Christian Kleiber and Simon Jackman 2008. "Regression Models for Count Data in R." *Journal of Statistical Software*, **27**(8). URL http://www.jstatsoft.org/v27/i08/.

See Also

hurdle.control, glm, glm.fit, glm.nb, zeroinfl

Examples

```
## data
data("bioChemists", package = "pscl")

## logit-poisson
## "art ~ ." is the same as "art ~ . | .", i.e.
## "art ~ fem + mar + kid5 + phd + ment | fem + mar + kid5 + phd + ment"
fm_hp1 <- hurdle(art ~ ., data = bioChemists)
summary(fm_hp1)

## geometric-poisson
fm_hp2 <- hurdle(art ~ ., data = bioChemists, zero = "geometric")
summary(fm_hp2)

## logit and geometric model are equivalent
coef(fm_hp1, model = "zero") - coef(fm_hp2, model = "zero")

## logit-negbin
fm_hnb1 <- hurdle(art ~ ., data = bioChemists, dist = "negbin")
```

```
summary(fm_hnb1)

## negbin-negbin
## (poorly conditioned zero hurdle, note the standard errors)
fm_hnb2 <- hurdle(art ~ ., data = bioChemists, dist = "negbin", zero = "negbin")
summary(fm_hnb2)
```

`hurdle.control` *Control Parameters for Hurdle Count Data Regression*

Description

Various parameters that control fitting of hurdle regression models using `hurdle`.

Usage

```
hurdle.control(method = "BFGS", maxit = 10000, trace = FALSE,
  separate = TRUE, start = NULL, ...)
```

Arguments

`method`	characters string specifying the `method` argument passed to `optim`.
`maxit`	integer specifying the `maxit` argument (maximal number of iterations) passed to `optim`.
`trace`	logical or integer controlling whether tracing information on the progress of the optimization should be produced (passed to `optim`).
`separate`	logical. Should the estimation of the parameters in the truncated count component and hurdle zero component be carried out separately? See details.
`start`	an optional list with elements `"count"` and `"zero"` (and potentially `"theta"`) containing the coefficients for the corresponding component.
`...`	arguments passed to `optim`.

Details

All parameters in `hurdle` are estimated by maximum likelihood using `optim` with control options set in `hurdle.control`. Most arguments are passed on directly to `optim`, only `trace` is also used within `hurdle` and `separate`/`start` control how `optim` is called.

Starting values can be supplied via `start` or estimated by `glm.fit` (default). If `separate = TRUE` (default) the likelihoods of the truncated count component and the hurdle zero component will be maximized separately, otherwise the joint likelihood is set up and maximized. Standard errors are derived numerically using the Hessian matrix returned by `optim`. To supply starting values, `start` should be a list with elements `"count"` and `"zero"` and potentially `"theta"` (a named vector, for models with negative binomial components only) containing the starting values for the coefficients of the corresponding component of the model.

Value

A list with the arguments specified.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

See Also

hurdle

Examples

```
data("bioChemists", package = "pscl")

## default start values
fm1 <- hurdle(art ~ fem + ment, data = bioChemists,
              dist = "negbin", zero = "negbin")

## user-supplied start values and other options
fm2 <- hurdle(art ~ fem + ment, data = bioChemists,
              dist = "negbin",
              zero = "negbin",
              trace=TRUE,
              separate=FALSE,
              start = list(count = c(0.3, -0.2, 0),
                           zero = c(4, -2, 0.8),
                           theta = c(count = 2, zero = 0.1)))
```

---

hurdletest *Testing for the Presence of a Zero Hurdle*

Description

Wald test of the null hypothesis that no zero hurdle is required in hurdle regression models for count data.

Usage

```
hurdletest(object, ...)
```

Arguments

object	A fitted model object of class "hurdle" as returned by hurdle, see details for more information.
...	arguments passed to linearHypothesis.

Details

If the same count distribution and the same set of regressors is used in the hurdle model for both, the count component and the zero hurdle component, then a test of pairwise equality between all coefficients from the two components assesses the null hypothesis that no hurdle is needed in the model.

The function `hurdletest` is a simple convenience interface to the function `linearHypothesis` from the **car** packages that can be employed to carry out a Wald test for this hypothesis.

Value

An object of class `"anova"` as returned by `linearHypothesis`.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

References

Cameron, A. Colin and Pravin K. Trivedi. 1998. *Regression Analysis of Count Data*. New York: Cambridge University Press.

Cameron, A. Colin and Pravin K. Trivedi 2005. *Microeconometrics: Methods and Applications*. Cambridge: Cambridge University Press.

See Also

`hurdle`, `linearHypothesis`

Examples

```
data("bioChemists", package = "pscl")
fm <- hurdle(art ~ ., data = bioChemists, dist = "negbin", zero = "negbin")
hurdletest(fm)
```

---

ideal *analysis of educational testing data and roll call data with IRT models, via Markov chain Monte Carlo methods*

Description

Analysis of `rollcall` data via the spatial voting model; equivalent to a 2 parameter item-response model to educational testing data. Model fitting via Markov chain Monte Carlo (MCMC).

Usage

```
ideal(object, codes = object$codes,
      dropList = list(codes = "notInLegis", lop = 0),
      d = 1, maxiter = 10000, thin = 100, burnin = 5000,
      impute = FALSE,
      normalize = FALSE,
      meanzero = normalize,
      priors = NULL, startvals = "eigen",
      store.item = FALSE, file = NULL,
      verbose=FALSE, use.voter=NULL)
```

Arguments

object	an object of class rollcall
codes	a list describing the types of voting decisions in the roll call matrix (the votes component of the rollcall object); defaults to object$codes, the codes in the rollcall object.
dropList	a list (or alist) listing voting decisions, legislators and/or votes to be dropped from the analysis; see dropRollCall for details.
d	numeric, (small) positive integer (default = 1), dimensionality of the ability space (or "policy space" in the rollcall context).
maxiter	numeric, positive integer, multiple of thin, number of MCMC iterations
thin	numeric, positive integer, thinning interval used for recording MCMC iterations.
burnin	number of MCMC iterations to run before recording. The iteration numbered burnin will be recorded. Must be a multiple of thin.
impute	logical, whether to treat missing entries of the rollcall matrix as missing at random, sampling from the predictive density of the missing entries at each MCMC iteration.
normalize	logical, impose identification with the constraint that the ideal points have mean zero and standard deviation one, in each dimension. For one dimensional models this option is sufficient to locally identify the model parameters. See Details.
meanzero	to be deprecated/ignored; use normalize instead.
priors	a list of parameters (means and variances) specifying normal priors for the legislators' ideal points. The default is NULL, in which case the normal priors used have mean zero and precision 1 for the ideal points (ability parameters) and mean zero and precision .04 (variance 25) for the bill parameters (item discrimination and difficulty parameters). If not NULL, priors must be a list with as many as four named components xp,xpv,bp,bpv: xp a n by d matrix of prior *means* for the legislators' ideal points; or alternatively, a scalar, which will be replicated to fill a n by d matrix. xpv a n by d matrix of prior *precisions* (inverse variances); or alternatively, a scalar, which will be replicated to fill a n by d matrix.

bp a m by d+1 matrix of prior means for the item parameters (with the item difficulty parameter coming last); or alternatively, a scalar, which will be replicated to fill a m by d+1 matrix.

bpv a m by d+1 matrix of prior precisions for the item parameters; or alternatively, a scalar, which will be replicated to fill a m by d+1 matrix.

None of the components should contain NA. If any of the four possible components are not provided, then the corresponding component of priors is assigned using the default values described above.

startvals either a string naming a method for generating start values, valid options are "eigen" (the default), "random" or a list containing start values for legislators' ideal points and item parameters. See Details.

store.item logical, whether item discrimination parameters should be stored. Storing item discrimination parameters can consume a large amount of memory. These need to be stored for prediction; see predict.ideal.

file string, file to write MCMC output. Default is NULL, in which case MCMC output is stored in memory. Note that post-estimation commands like plot will not work unless MCMC output is stored in memory.

verbose logical, default is FALSE, which generates relatively little output to the R console during execution.

use.voter A vector of logicals of length n controlling which legislators' vote data informs item parameter estimates. Legislators corresponding to FALSE entries will not have their voting data included in updates of the item parameters. The default value of NULL will run the standard ideal-point model, which uses all legislators in updating item parameters. See Jessee (2016).

Details

The function fits a d+1 parameter item-response model to the roll call data object, so in one dimension the model reduces to the two-parameter item-response model popular in educational testing. See References.

Identification: The model parameters are **not identified** without the user supplying some restrictions on the model parameters; i.e., translations, rotations and re-scalings of the ideal points are observationally equivalent, via offsetting transformations of the item parameters. It is the user's responsibility to impose these identifying restrictions if desired. The following brief discussion provides some guidance.

For one-dimensional models (i.e., d=1), a simple route to identification is the normalize option, by imposing the restriction that the means of the posterior densities of the ideal points (ability parameters) have mean zero and standard deviation one, across legislators (test-takers). This normalization supplies *local* identification (that is, identification up to a 180 degree rotation of the recovered dimension).

Near-degenerate "spike" priors (priors with arbitrarily large precisions) or the constrain.legis option on any two legislators' ideal points ensures *global* identification in one dimension.

Identification in higher dimensions can be obtained by supplying fixed values for d+1 legislators' ideal points, provided the supplied fixed points span a d-dimensional space (e.g., three supplied ideal points form a triangle in d=2 dimensions), via the constrain.legis option. In this case the

function defaults to vague normal priors on the unconstrained ideal points, but at each iteration the sampled ideal points are transformed back into the space of identified parameters, applying the linear transformation that maps the d+1 fixed ideal points from their sampled values to their fixed values. Alternatively, one can impose restrictions on the item parameters via constrain.items. See the examples in the documentation for the constrain.legis and constrain.items.

Another route to identification is via *post-processing*. That is, the user can run ideal without any identification constraints. This does not pose any formal/technical problem in a Bayesian analysis. The fact that the posterior density may have multiple modes doesn't imply that the posterior is improper or that it can't be explored via MCMC methods. – but then use the function postProcess to map the MCMC output from the space of unidentified parameters into the subspace of identified parameters. See the example in the documentation for the postProcess function.

When the normalize option is set to TRUE, an unidentified model is run, and the ideal object is post-processed with the normalize option, and then returned to the user (but again, note that the normalize option is only implemented for unidimensional models).

Start values. Start values can be supplied by the user, or generated by the function itself.

The default method, corresponding to startvals="eigen", first forms a n-by-n correlation matrix from the double-centered roll call matrix (subtracting row means, and column means, adding in the grand mean), and then extracts the first d principal components (eigenvectors), scaling the eigenvectors by the square root of their corresponding eigenvector. If the user is imposing constraints on ideal points (via constrain.legis), these constraints are applied to the corresponding elements of the start values generated from the eigen-decomposition. Then, to generate start values for the rollcall/item parameters, a series of binomial glms are estimated (with a probit link), one for each rollcall/item, $j = 1, \ldots, m$. The votes on the j-th rollcall/item are binary responses (presumed to be conditionally independent given each legislator's latent preference), and the (constrained or unconstrained) start values for legislators are used as predictors. The estimated coefficients from these probit models are used as start values for the item discrimination and difficulty parameters (with the intercepts from the probit GLMs multiplied by -1 so as to make those coefficients difficulty parameters).

The default eigen method generates extremely good start values for low-dimensional models fit to recent U.S. congresses, where high rates of party line voting result in excellent fits from low dimensional models. The eigen method may be computationally expensive or lead to memory errors for rollcall objects with large numbers of legislators.

The random method generates start values via iid sampling from a N(0,1) density, via rnorm, imposing any constraints that may have been supplied via constrain.legis, and then uses the probit method described above to get start values for the rollcall/item parameters.

If startvals is a list, it must contain the named components x and/or b, or named components that (uniquely) begin with the letters x and/or b. The component x must be a vector or a matrix of dimensions equal to the number of individuals (legislators) by d. If supplied, startvals$b must be a matrix with dimension number of items (votes) by d+1. The x and b components cannot contain NA. If x is not supplied when startvals is a list, then start values are generated using the default eiegn method described above, and start values for the rollcall/item parameters are regenerated using the probit method, ignoring any user-supplied values in startvals$b. That is, user-supplied values in startvals$b are only used when accompanied by a valid set of start values for the ideal points in startvals$x.

Implementation via Data Augmentation. The MCMC algorithm for this problem consists of a Gibbs sampler for the ideal points (latent traits) and item parameters, conditional on latent data y^*,

generated via a data augmentation (DA) step. That is, following Albert (1992) and Albert and Chib (1993), if $y_{ij} = 1$ we sample from the truncated normal density

$$y^*_{ij} \sim N(x'_i\beta_j - \alpha_j, 1)\mathcal{I}(y^*_{ij} \geq 0)$$

and for $y_{ij} = 0$ we sample

$$y^*_{ij} \sim N(x'_i\beta_j - \alpha_j, 1)\mathcal{I}(y^*_{ij} < 0)$$

where $\mathcal{I}$ is an indicator function evaluating to one if its argument is true and zero otherwise. Given the latent y^*, the conditional distributions for x and (β, α) are extremely simple to sample from; see the references for details.

This data-augmented Gibbs sampling strategy is easily implemented, but can sometimes require many thousands of samples in order to generate tolerable explorations of the posterior densities of the latent traits, particularly for legislators with short and/or extreme voting histories (the equivalent in the educational testing setting is a test-taker who gets almost every item right or wrong).

Value

a `list` of class `ideal` with named components

`n`	`numeric`, integer, number of legislators in the analysis, after any subsetting via processing the `dropList`.
`m`	`numeric`, integer, number of rollcalls in roll call matrix, after any subsetting via processing the `dropList`.
`d`	`numeric`, integer, number of dimensions fitted.
`x`	a three-dimensional `array` containing the MCMC output with respect to the the ideal point of each legislator in each dimension. The three-dimensional array is in iteration-legislator-dimension order. The iterations run from `burnin` to `maxiter`, at an interval of `thin`.
`beta`	a three-dimensional `array` containing the MCMC output for the item parameters. The three-dimensional array is in iteration-rollcall-parameter order. The iterations run from `burnin` to `maxiter`, at an interval of `thin`. Each rollcall has `d+1` parameters, with the item-discrimination parameters stored first, in the first `d` components of the 3rd dimension of the `beta` array; the item-difficulty parameter follows in the final `d+1` component of the 3rd dimension of the `beta` array.
`xbar`	a `n` by `d` `matrix` containing the means of the MCMC samples for the ideal point of each legislator in each dimension, using iterations `burnin` to `maxiter`, at an interval of `thin`.
`betabar`	a `m` by `d+1` `matrix` containing the means of the MCMC samples for the item-specific parameters, using iterations `burnin` to `maxiter`, at an interval of `thin`.
`args`	calling arguments, evaluated in the frame calling `ideal`.
`call`	an object of class `call`, containing the arguments passed to `ideal` as unevaluated expressions or values (for functions arguments that evaluate to scalar integer or logical such as `maxiter`, `burnin`, etc).

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>, with help from Christina Maimone and Alex Tahk.

References

Albert, James. 1992. Bayesian Estimation of normal ogive item response curves using Gibbs sampling. *Journal of Educational Statistics*. 17:251-269.

Albert, James H. and Siddhartha Chib. 1993. Bayesian Analysis of Binary and Polychotomous Response Data. *Journal of the American Statistical Association*. 88:669-679.

Clinton, Joshua, Simon Jackman and Douglas Rivers. 2004. The Statistical Analysis of Roll Call Data. *American Political Science Review*. 98:335-370.

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Hoboken, New Jersey.

Jessee, Stephen. 2016. (How) Can We Estimate the Ideology of Citizens and Political Elites on the Same Scale? *American Journal of Political Science*.

Patz, Richard J. and Brian W. Junker. 1999. A Straightforward Approach to Markov Chain Monte Carlo Methods for Item Response Models. *Journal of Education and Behavioral Statistics*. 24:146-178.

Rivers, Douglas. 2003. "Identification of Multidimensional Item-Response Models." Typescript. Department of Political Science, Stanford University.

van Dyk, David A and Xiao-Li Meng. 2001. The art of data augmentation (with discussion). *Journal of Computational and Graphical Statistics*. 10(1):1-111.

See Also

`rollcall`, `summary.ideal`, `plot.ideal`, `predict.ideal`. `tracex` for graphical display of MCMC iterative history.

`idealToMCMC` converts the MCMC iterates in an `ideal` object to a form that can be used by the coda library.

`constrain.items` and `constrain.legis` for implementing identifying restrictions.

`postProcess` for imposing identifying restrictions *ex post*.

`MCMCirt1d` and `MCMCirtKd` in the **MCMCpack** package provide similar functionality to `ideal`.

Examples

```
## Not run:
## long run, many iterations
data(s109)
n <- dim(s109$legis.data)[1]
x0 <- rep(0,n)
x0[s109$legis.data$party=="D"] <- -1
x0[s109$legis.data$party=="R"] <- 1

id1 <- ideal(s109,
             d=1,
             startvals=list(x=x0),
```

```
              normalize=TRUE,
              store.item=TRUE,
              maxiter=260E3,
              burnin=10E3,
              thin=100)

## End(Not run)
```

---

idealToMCMC *convert an object of class ideal to a coda MCMC object*

Description

Converts the x element of an ideal object to an MCMC object, as used in the **coda** package.

Usage

```
idealToMCMC(object, burnin=NULL)
```

Arguments

object	an object of class ideal.
burnin	of the recorded MCMC samples, how many to discard as burnin? Default is NULL, in which case the value of burnin in the ideal object is used.

Value

A mcmc object as used by the **coda** package, starting at iteration start, drawn from the x component of the ideal object.

Note

When specifying a value of burnin different from that used in fitting the ideal object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the n-th iteration stored in an ideal object will not be iteration n if the user specified thin>1 in the call to ideal. Here, iterations are tagged with their iteration number. Thus, if the user called ideal with thin=10 and burnin=100 then the stored iterations are numbered 100,110,120,.... Any future subsetting via a burnin refers to this iteration number.

See Also

ideal, mcmc

Examples

```
data(s109)
f = system.file("extdata",package="pscl","id1.rda")
load(f)
id1coda <- idealToMCMC(id1)
summary(id1coda)
```

igamma *inverse-Gamma distribution*

Description

Density, distribution function, quantile function, and highest density region calculation for the inverse-Gamma distribution with parameters `alpha` and `beta`.

Usage

```
densigamma(x,alpha,beta)
   pigamma(q,alpha,beta)
qigamma(p,alpha,beta)
rigamma(n,alpha,beta)
igammaHDR(alpha,beta,content=.95,debug=FALSE)
```

Arguments

`x,q`	vector of quantiles
`p`	vector of probabilities
`n`	number of random samples in `rigamma`
`alpha,beta`	rate and shape parameters of the inverse-Gamma density, both positive
`content`	scalar, 0 < `content` < 1, volume of highest density region
`debug`	logical; if TRUE, debugging information from the search for the HDR is printed

Details

The inverse-Gamma density arises frequently in Bayesian analysis of normal data, as the (marginal) conjugate prior for the unknown variance parameter. The inverse-Gamma density for $x > 0$ with parameters $\alpha > 0$ and $\beta > 0$ is

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{-\alpha-1} \exp(-\beta/x)$$

where $\Gamma(x)$ is the gamma function

$$\Gamma(a) = \int_0^\infty t^{a-1} \exp(-t) dt$$

and so ensures $f(x)$ integrates to one. The inverse-Gamma density has a mean at $\beta/(\alpha - 1)$ for $\alpha > 1$ and has variance $\beta^2/((\alpha-1)^2(\alpha-2))$ for $\alpha > 2$. The inverse-Gamma density has a unique mode at $\beta/(\alpha + 1)$.

The evaluation of the density, cumulative distribution function and quantiles is done by calls to the `dgamma`, `pgamma` and `igamma` functions, with the arguments appropriately transformed. That is, note that if $x \sim IG(\alpha, \beta)$ then $1/x \sim G(\alpha, \beta)$.

Highest Density Regions. In general, suppose x has a density $f(x)$, where $x \in \Theta$. Then a highest density region (HDR) for x with content $p \in (0, 1]$ is a region (or set of regions) $\mathcal{Q} \subseteq \Theta$ such that:

$$\int_{\mathcal{Q}} f(x)dx = p$$

and

$$f(x) > f(x^*) \, \forall \, x \in \mathcal{Q}, x^* \notin \mathcal{Q}.$$

For a continuous, unimodal density defined with respect to a single parameter (like the inverse-Gamma case considered here with parameters $0 < \alpha < \infty$, $0 < \beta < \infty$), a HDR region Q of content p (with $0 < p < 1$) is a unique, closed interval on the real half-line.

This function uses numerical methods to solve for the boundaries of a HDR with `content` p for the inverse-Gamma density, via repeated calls the functions `densigamma`, `pigamma` and `qigamma`. In particular, the function `uniroot` is used to find points v and w such that

$$f(v) = f(w)$$

subject to the constraint

$$\int_v^w f(x; \alpha, \beta)d\theta = p.$$

Value

`densigamma` gives the density, `pigamma` the distribution function, `qigamma` the quantile function, `rigamma` generates random samples, and `igammaHDR` gives the lower and upper limits of the HDR, as defined above (`NA`s if the optimization is not successful).

Note

The `densigamma` is named so as not to conflict with the `digamma` function in the R `base` package (the derivative of the `gamma` function).

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

`gamma`, `dgamma`, `pgamma`, `qgamma`, `uniroot`

Examples

```
alpha <- 4
beta <- 30
summary(rigamma(n=1000,alpha,beta))

xseq <- seq(.1,30,by=.1)
fx <- densigamma(xseq,alpha,beta)
plot(xseq,fx,type="n",
     xlab="x",
     ylab="f(x)",
```

```
     ylim=c(0,1.01*max(fx)),
     yaxs="i",
     axes=FALSE)
axis(1)
title(substitute(list(alpha==a,beta==b),list(a=alpha,b=beta)))
q <- igammaHDR(alpha,beta,debug=TRUE)
xlo <- which.min(abs(q[1]-xseq))
xup <- which.min(abs(q[2]-xseq))
plotZero <- par()$usr[3]
polygon(x=xseq[c(xlo,xlo:xup,xup:xlo)],
        y=c(plotZero,
          fx[xlo:xup],
          rep(plotZero,length(xlo:xup))),
        border=FALSE,
        col=gray(.45))
lines(xseq,fx,lwd=1.25)


## Not run:
alpha <- beta <- .1
xseq <- exp(seq(-7,30,length=1001))
fx <- densigamma(xseq,alpha,beta)
plot(xseq,fx,
     log="xy",
     type="l",
     ylim=c(min(fx),1.01*max(fx)),
     yaxs="i",
     xlab="x, log scale",
     ylab="f(x), log scale",
     axes=FALSE)
axis(1)

title(substitute(list(alpha==a,beta==b),list(a=alpha,b=beta)))
q <- igammaHDR(alpha,beta,debug=TRUE)
xlo <- which.min(abs(q[1]-xseq))
xup <- which.min(abs(q[2]-xseq))
plotZero <- min(fx)
polygon(x=xseq[c(xlo,xlo:xup,xup:xlo)],
        y=c(plotZero,
          fx[xlo:xup],
          rep(plotZero,length(xlo:xup))),
        border=FALSE,
        col=gray(.45))
lines(xseq,fx,lwd=1.25)

## End(Not run)
```

---

iraqVote *U.S. Senate vote on the use of force against Iraq, 2002.*

Description

On October 11, 2002, the United States Senate voted 77-23 to authorize the use of military force against Iraq. This data set lists the "Ayes" and "Nays" for each Senator and some covariates.

Usage

```
data(iraqVote)
```

Format

A data frame with 100 observations on the following 6 variables.

`y` a numeric vector, the recorded vote (1 if Aye, 0 if Nay)

`state.abb` two letter abbreviation for each state

`name` senator name, party and state, e.g., `AKAKA (D HI)`

`rep` logical, `TRUE` for Republican senators

`state.name` name of state

`gorevote` numeric, the vote share recorded by Al Gore in the corresponding state in the 2000 Presidential election

Details

The only Republican to vote against the resolution was Lincoln Chafee (Rhode Island); Democrats split 29-22 in favor of the resolution.

Source

Keith Poole, 107th Senate Roll Call Data. https://voteview.com/static/data/out/votes/S107_votes.ord The Iraq vote is vote number 617.

David Leip's Atlas of U.S. Presidential Elections. http://uselectionatlas.org

References

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Chichester. Example 8.3.

Examples

```
data(iraqVote)
## probit model
glm1 <- glm(y ~ gorevote + rep,
            data=iraqVote,
            family=binomial(link=probit))
```

nj07 *rollcall object, National Journal key votes of 2007*

Description

A rollcall object containing 99 rollcalls from the 2nd session of the 110th U.S. Senate, designated by *National Journal* as the "key votes" of 2007. These data were used to by *National Journal* to rate (then Senator) Barack Obama was the "most liberal senator" in 2007.

Usage

```
data(nj07)
```

Format

A rollcall object containing the recorded votes, plus information identifying the legislators and the rollcalls.

Details

Note the coding scheme used by Poole and Rosenthal; Yea (1,2,3), Nay (4,5,6) etc.

Source

Keith Poole's web site: http://legacy.voteview.com/senate110.htm

Originally scraped from the Senate's web site by Jeff Lewis.

Josh Clinton compiled the list of *National Journal* key votes.

References

Clinton, Joshua and Simon Jackman. 2009. To Simulate or NOMINATE? *Legislative Studies Quarterly*. V34(4):593-621.

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Hoboken, New Jersey. Example 9.2.

Examples

```
require(pscl)
data(nj07)
is(nj07,"rollcall")    ## TRUE
nj07                   ## print method for class rollcall
names(nj07)
names(nj07$vote.data)
table(nj07$vote.data$policyArea)
summary(nj07)          ## summary method
summary(nj07,verbose=TRUE)
```

ntable *nicely formatted tables*

Description

Nicely formatted tables, with row or column marginals etc.

Usage

```
ntable(x,y=NULL,
       percent=1,digits=2,
       row=FALSE,col=FALSE)
```

Arguments

x	vector or factor
y	vector of factor
percent	integer, 1 for row percentages (default), 2 for column percentages
digits	integer, digits to print after decimal place (default is 2)
row	logical, if TRUE, print row marginals
col	logical, if TRUE, print column marginals

Details

A wrapper function to prop.table that produces prettier looking results.

Value

nothing returned; the function prints the table and exits silently.

Author(s)

Jim Fearon <jfearon@stanford.edu>

See Also

prop.table, table

Examples

```
data(bioChemists)
attach(bioChemists)
ntable(fem)
ntable(fem,mar,row=TRUE)
ntable(fem,mar,per=2,col=TRUE)
ntable(fem,mar,per=2,row=TRUE,col=TRUE)
```

odTest *likelihood ratio test for over-dispersion in count data*

Description

Compares the log-likelihoods of a negative binomial regression model and a Poisson regression model.

Usage

```
odTest(glmobj, alpha=.05, digits = max(3, getOption("digits") - 3))
```

Arguments

glmobj	an object of class negbin produced by glm.nb
alpha	significance level of over-dispersion test
digits	number of digits in printed output

Details

The negative binomial model relaxes the assumption in the Poisson model that the (conditional) variance equals the (conditional) mean, by estimating one extra parameter. A likelihood ratio (LR) test can be used to test the null hypothesis that the restriction implicit in the Poisson model is true. The LR test-statistic has a non-standard distribution, even asymptotically, since the negative binomial over-dispersion parameter (called theta in glm.nb) is restricted to be positive. The asymptotic distribution of the LR (likelihood ratio) test-statistic has probability mass of one half at zero, and a half χ^2_1 distribution above zero. This means that if testing at the α = .05 level, one should not reject the null unless the LR test statistic exceeds the critical value associated with the 2α = .10 level; this LR test involves just one parameter restriction, so the critical value of the test statistic at the p = .05 level is 2.7, instead of the usual 3.8 (i.e., the .90 quantile of the χ^2_1 distribution, versus the .95 quantile).

A Poisson model is run using glm with family set to link{poisson}, using the formula in the negbin model object passed as input. The logLik functions are used to extract the log-likelihood for each model.

Value

None; prints results and returns silently

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>. John Fox noted an error in an earlier version.

References

A. Colin Cameron and Pravin K. Trivedi (1998) *Regression analysis of count data*. New York: Cambridge University Press.

Lawless, J. F. (1987) Negative Binomial and Mixed Poisson Regressions. *The Canadian Journal of Statistics*. 15:209-225.

See Also

glm.nb, logLik

Examples

```
data(bioChemists)
modelnb <- MASS::glm.nb(art ~ .,
                 data=bioChemists,
                 trace=TRUE)
odTest(modelnb)
```

---

partycodes *political parties appearing in the U.S. Congress*

Description

Numeric codes and names of 85 political parties appearing in Poole and Rosenthal's collection of U.S. Congressional roll calls.

Usage

```
data(partycodes)
```

Format

- codeinteger, numeric code for legislator appearing in Poole and Rosenthal rollcall data files
- partycharacter, name of party

Details

The function readKH converts the integer codes into strings, via a table lookup in this data frame.

Source

Keith Poole's website: http://legacy.voteview.com/PARTY3.HTM

See Also

readKH

plot.ideal *plots an ideal object*

Description

Plot of the results of an ideal point estimation contained in an object of class `ideal`.

Usage

```
## S3 method for class 'ideal'
plot(x, conf.int=0.95, burnin=NULL, ...)

plot1d(x, d=1, conf.int=0.95, burnin=NULL,
       showAllNames = FALSE, ...)

plot2d(x, d1=1, d2=2, burnin=NULL,
       overlayCuttingPlanes=FALSE, ...)
```

Arguments

`x`	an object of class `ideal`
`conf.int`	for "ideal" objects with 1 dimension estimated, the level of the confidence interval to plot around the posterior mean for each legislator. If 2 or more dimensions were estimated, `conf.int` is ignored.
`d`	integer, which dimension to display in a 1d plot, if the object is a multidimensional ideal object.
`burnin`	of the recorded MCMC samples, how many to discard as burnin? Default is `NULL`, in which case the value of `burnin` in the `ideal` object is used.
`showAllNames`	`logical`, if `TRUE`, the vertical axis will the names of all legislators. Default is `FALSE` to reduce clutter on typical-sized graph.
`d1`	integer, the number of the first dimension to plot when plotting multi-dimensional `ideal` objects. This dimension will appear on the horizontal (x) axis.
`d2`	integer, the number of the second dimension to plot when plotting multi-dimensional `ideal` objects. This dimension will appear on the vertical (y) axis.
`overlayCuttingPlanes`	logical, if `TRUE`, overlay the estimated bill-specific cutting planes
`...`	other parameters to be passed through to plotting functions.

Details

If the `ideal` object comes from fitting a `d=1` dimensional model, then `plot.ideal` plots the mean of the posterior density over each legislator's ideal point, accompanied by a `conf.int` confidence interval. In this case, `plot.ideal` is simply a wrapper function to `plot1d`.

If the `ideal` object has `d=2` dimensions, then `plot2d` is called, which plots the (estimated) mean of the posterior density of each legislator's ideal point (i.e., the ideal point/latent trait is a point in

2-dimensional Euclidean space, and the posterior density for each ideal point is a bivariate density). Single dimension summaries of the estimated ideal points (latent traits) can be obtained for multi-dimensional ideal objects by passing the ideal object directly to plot1d with d set appropriately.

If the ideal object has d>2 dimensions, a scatterplot matrix is produced via pairs, with the posterior means of the ideal points (latent traits) plotted against one another, dimension by dimension.

For unidimensional and two-dimensional models, if party information is available in the rollcall object contained in the ideal object, legislators from different parties are plotted in different colors. If the ideal object has more than 2 dimensions, plot.ideal() produces a matrix of plots of the mean ideal points of each dimension against the posterior mean ideal points of the other dimensions.

Note

When specifying a value of burnin different from that used in fitting the ideal object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the n-th iteration stored in an ideal object will not be iteration n if the user specified thin>1 in the call to ideal. Here, iterations are tagged with their iteration number. Thus, if the user called ideal with thin=10 and burnin=100 then the stored iterations are numbered 100,110,120,.... Any future subsetting via a burnin refers to this iteration number.

See Also

ideal; tracex for trace plots, a graphical aid useful in diagnosing convergence of the MCMC algorithms.

Examples

```
## Not run:
data(s109)
id1 <- ideal(s109,
             d=1,
             normalize=TRUE,
             store.item=TRUE,
             maxiter=500,   ## short run for examples
             burnin=100,
             thin=10)

plot(id1)


id2 <- ideal(s109,
             d=2,
             store.item=TRUE,
             maxiter=11e2,
             burnin=1e2,
             verbose=TRUE,
             thin=25)

plot(id2,overlayCuttingPlanes=TRUE)

id2pp <- postProcess(id2,
                     constraints=list(BOXER=c(-1,0),
```

```
                        INHOFE=c(1,0),
                        CHAFEE=c(0,.25)))

plot(id2pp,overlayCuttingPlanes=TRUE)

## End(Not run)
```

`plot.predict.ideal` *plot methods for predictions from ideal objects*

Description

Plot classification success rates by legislators, or by roll calls, using predictions from ideal.

Usage

```
## S3 method for class 'predict.ideal'
plot(x, type = c("legis", "votes"),...)
```

Arguments

`x`	an object of class `predict.ideal`.
`type`	string; one of `legis` or `votes`.
`...`	further arguments passed to or from other methods.

Details

`type="legis"` produces a plot of the "percent correctly predicted" for each legislator/subject (using the classification threshold set in `predict.ideal`) against the estimated ideal point of each legislator/subject (the estimated mean of the posterior density of the ideal point), dimension at a time. If the legislators' party affiliations are available in the `rollcall` object that was passed to `ideal`, then legislators from the same party are plotted with a unique color.

`type="votes"` produces a plot of classification rates for each roll call, by the percentage of legislators voting for the losing side. The `x`-ordinate is jittered for clarity.

Value

After drawing plots on the current device, exits silently returning `invisible(NULL)`.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

`predict.ideal` `ideal`

Examples

```
data(s109)
f = system.file("extdata","id1.rda",package="pscl")
load(f)
phat <- predict(id1)
plot(phat,type="legis")
plot(phat,type="votes")
```

`plot.seatsVotes` *plot seats-votes curves*

Description

Plot seats-votes curves produced by `seatsVotes`

Usage

```
## S3 method for class 'seatsVotes'
plot(x, type = c("seatsVotes", "density"),
                legend = "bottomright", transform=FALSE, ...)
```

Arguments

`x`	an object of class `seatsVotes`
`type`	character, partially matching the options above; see details
`legend`	where to put the legend when plotting with `type="seatsVotes"`
`transform`	logical, whether to transform the vote shares for `type="density"`; see Details
`...`	arguments passed to or from other functions (e.g., options for the `density` function, when `type="density"`)

Details

A seats-votes curve (with various annotations) is produced with option `type="seatsVotes"`.

A density plot of the vote shares is produced with `type="density"`. A bimodal density corresponds to an electoral system with a proliferation of safe seats for both parties, and a seats-votes curve that is relatively flat (or "unresponsive") in the neighborhood of average district-level vote shares of 50 percent. The density is fitted using the defaults in the `density` function, but with the density constrained to fall to zero at the extremes of the data, via the `from` and `to` options to density. A `rug` is added to the density plot.

If `transform=TRUE`, the vote shares are transformed prior to plotting, so as to reduce the extent to which extreme vote shares close to zero or 1 determine the shape of the density (i.e., this option is available only for plots of `type="density"`). The transformation is a sinusoidal function, a scaled "log-odds/inverse-log-odds" function mapping from (0,1) to (0,1): i.e., $f(x) = g(k \cdot h(x))$ where $h(x)$ is the log-odds transformation $h(x) = \log(x/(1-x))$, k is a scaling parameter set to $\sqrt{3}$, and $g(x)$ is the inverse-log-odds transformation $g(x) = \exp(x)/(1+\exp(x))$. Note that this transformation is cosmetic, with the effect of assigning more of the graphing region to be devoted to marginal seats.

Value

The function performs the requested plots and exits silently with `invisible{NULL}`.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

density, rug

Examples

```
data(ca2006)
x <- ca2006$D/(ca2006$D+ca2006$R)
sv <- seatsVotes(x,
                 desc="Democratic Vote Shares, California 2006 congressional elections")

plot(sv)
plot(sv,type="density")
plot(sv,type="density",transform=TRUE)
```

---

politicalInformation *Interviewer ratings of respondent levels of political information*

Description

Interviewers administering the 2000 American National Election Studies assigned an ordinal rating to each respondent's "general level of information" about politics and public affairs.

Usage

```
data(politicalInformation)
```

Format

A data frame with 1807 observations on the following 8 variables.

`y` interviewer rating, a factor with levels `Very Low` `Fairly Low` `Average` `Fairly High` `Very High`

`collegeDegree` a factor with levels `No` `Yes`

`female` a factor with levels `No` `Yes`

`age` a numeric vector, respondent age in years

`homeOwn` a factor with levels `No` `Yes`

`govt` a factor with levels `No` `Yes`

`length` a numeric vector, length of ANES pre-election interview in minutes

`id` a factor, unique identifier for each interviewer

Details

Seven respondents have missing data on the ordinal interviewer rating. The covariates age and length also have some missing data.

Source

The National Election Studies (www.electionstudies.org). THE 2000 NATIONAL ELECTION STUDY [dataset]. Ann Arbor, MI: University of Michigan, Center for Political Studies [producer and distributor].

References

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Hoboken, New Jersey.

Examples

```
data(politicalInformation)

table(politicalInformation$y,exclude=NULL)

op <- MASS::polr(y ~ collegeDegree + female + log(age) + homeOwn + govt + log(length),
           data=politicalInformation,
           Hess=TRUE,
           method="probit")
```

---

postProcess *remap MCMC output via affine transformations*

Description

Remap the MCMC iterates in an ideal object via an affine transformation, imposing identifying restrictions ex post (aka post-processing).

Usage

```
postProcess(object, constraints="normalize", debug = FALSE)
```

Arguments

object	an object of class ideal
constraints	list of length d+1, each component providing a set of d restrictions, where d is the dimension of the fitted ideal model; or the character string normalize (default). If a list, the name of each component should uniquely match a legislator/subject's name. See Details.
debug	logical flag for verbose output, used for debugging

Details

Item-response models are unidentified without restrictions on the underlying parameters. Consider the d=1 dimensional case. The model is

$$P(y_{ij} = 1) = F(x_i\beta_j - \alpha_j)$$

Any linear transformation of the latent traits, say,

$$x^* = mx + c$$

can be exactly offset by applying the appropriate linear transformations to the item/bill parameters, meaning that there is no unique set of values for the model parameters that will maximize the likelihood function. In higher dimensions, the latent traits can also be transformed via any arbitrary rotation, dilation and translation, with offsetting transformations applied to the item/bill parameters.

One strategy in MCMC is to ignore the lack of identification at run time, but apply identifying restrictions ex post, "post-processing" the MCMC output, iteration-by-iteration. In a d-dimensional IRT model, a sufficient condition for global identification is to fix d+1 latent traits, provided the constrained latent traits span the d dimensional latent space. This function implements this strategy. The user supplies a set of constrained ideal points in the constraints list. The function then processes the MCMC output in the ideal object, finding the transformation that maps the current iteration's sampled values for x (latent traits/ideal points) into the sub-space of identified parameters defined by the fixed points in constraints; i.e., what is the affine transformation that maps the unconstrained ideal points into the constraints? Aside from minuscule numerical inaccuracies resulting from matrix inversion etc, this transformation is exact: after post-processing, the d+1 constrained points do not vary over the MCMC iterations. The remaining n-d-1 ideal points are subject to (posterior) uncertainty; the "random tour" of the joint parameter space of these parameters produced by the MCMC algorithm has been mapped into a subspace in which the parameters are globally identified.

If the ideal object was produced with store.item set to TRUE, then the item parameters are also post-processed, applying the inverse transformation. Specifically, recall that the IRT model is

$$P(y_{ij} = 1) = F(x_i'\beta_j)$$

where in this formulation x_i is a vector of length d+1, including a -1 to put a constant term into the model (i.e., the intercept or difficulty parameter is part of β_j). Let A denote the non-singular, d+1-by-d+1 matrix that maps the x into the space of identified parameters. Recall that this transformation is computed iteration by iteration. Then each x_i is transformed to $x_i^* = Ax_i$ and β_j is transformed to $\beta_j^* = A^{-1}\beta_j$, $i = 1, \ldots, n; j = 1, \ldots, m$.

Local identification can be obtained for a one-dimensional model by simply imposing a normalizing restriction on the ideal points: this normalization (mean zero, standard deviation one) is the default behavior, but (a) is only sufficient for local identification when the rollcall object was fit with d=1; (b) is not sufficient for even local identification when d>1, with further restrictions required so as to rule out other forms of invariance (e.g., translation, or "dimension-switching", a phenomenon akin to label-switching in mixture modeling).

The default is to impose dimension-by-dimension normalization with respect to the means of the marginal posterior densities of the ideal points, such that the these means (the usual Bayes estimates of the ideal points) have mean zero and standard deviation one across legislators. An offsetting transformation is applied to the items parameters as well, if they are saved in the ideal object.

Specifically, in one-dimension, the two-parameter IRT model is

$$P(y_{ij} = 1) = F(x_i\beta_j - \alpha_j).$$

If we normalize the x_i to $x*_i = (x_i - c)/m$ then the offsetting transformations for the item/bill parameters are $\beta_j^* = \beta_j m$ and $\alpha_j^* = \alpha_j - c\beta_j$.

Value

An object of class `ideal`, with components suitably transformed and recomputed (i.e., `x` is transformed and `xbar` recomputed, and if the `ideal` object was fit with `store.item=TRUE`, `beta` is transformed and `betabar` is recomputed).

Note

Applying transformations to obtain identification can sometimes lead to surprising results. Each data point makes the same likelihood contributions with either the identified or unidentified parameters. But, in general, predictions generated with the parameters set to their posterior means will differ depending on whether one uses the identified subset of parameters or the unidentified parameters. For this reason, caution should be used when using a function such as `predict` after post-processing output from `ideal`. A better strategy is to compute the estimand of interest at each iteration and then take averages over iterations.

When specifying a value of `burnin` different from that used in fitting the `ideal` object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the `n`-th iteration stored in an `ideal` object will not be iteration `n` if the user specified `thin>1` in the call to `ideal`. Here, iterations are tagged with their iteration number. Thus, if the user called `ideal` with `thin=10` and `burnin=100` then the stored iterations are numbered `100,110,120,...`. Any future subsetting via a `burnin` refers to this iteration number.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

References

Hoff, Peter, Adrian E. Raftery and Mark S. Handcock. 2002. Latent Space Approaches to Social Network Analysis. *Journal of the American Statistical Association* 97:1090–1098.

Edwards, Yancy D. and Greg M. Allenby. 2003. Multivariate Analysis of Multiple Response Data. *Journal of Marketing Research* 40:321–334.

Rivers, Douglas. 2003. "Identification of Multidimensional Item-Response Models." Typescript. Department of Political Science, Stanford University.

Examples

```
data(s109)
f = system.file("extdata",package="pscl","id1.rda")
load(f)

id1Local <- postProcess(id1)    ## default is to normalize
summary(id1Local)
```

```
id1pp <- postProcess(id1,
                     constraints=list(BOXER=-1,INHOFE=1))
summary(id1pp)

## two-dimensional fit
f = system.file("extdata",package="pscl","id2.rda")
load(f)

id2pp <- postProcess(id2,
                     constraints=list(BOXER=c(-1,0),
                       INHOFE=c(1,0),
                       CHAFEE=c(0,.25)))

tracex(id2pp,d=1:2,
       legis=c("BOXER","INHOFE","COLLINS","FEINGOLD","COLEMAN",
         "CHAFEE","MCCAIN","KYL"))
```

---

pR2 *compute various pseudo-R2 measures*

Description

compute various pseudo-R2 measures for various GLMs

Usage

```
pR2(object, ...)
```

Arguments

`object`	a fitted model object for which `logLik`, `update`, and `model.frame` methods exist (e.g., an object of class `glm`, `polr`, or `mulitnom`)
`...`	additional arguments to be passed to or from functions

Details

Numerous pseudo r-squared measures have been proposed for generalized linear models, involving a comparison of the log-likelihood for the fitted model against the log-likelihood of a null/restricted model with no predictors, normalized to run from zero to one as the fitted model provides a better fit to the data (providing a rough analogue to the computation of r-squared in a linear regression).

Value

A vector of length 6 containing

`llh`	The log-likelihood from the fitted model
`llhNull`	The log-likelihood from the intercept-only restricted model

G2	Minus two times the difference in the log-likelihoods
McFadden	McFadden's pseudo r-squared
r2ML	Maximum likelihood pseudo r-squared
r2CU	Cragg and Uhler's pseudo r-squared

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

References

Long, J. Scott. 1997. *Regression Models for Categorical and Limited Dependent Variables*. Sage. pp104-106.

See Also

extractAIC, logLik

Examples

```
data(admit)
## ordered probit model
op1 <- MASS::polr(score ~ gre.quant + gre.verbal + ap + pt + female,
            Hess=TRUE,
            data=admit,
            method="probit")
pR2(op1)
```

---

predict.hurdle *Methods for hurdle Objects*

Description

Methods for extracting information from fitted hurdle regression model objects of class "hurdle".

Usage

```
## S3 method for class 'hurdle'
predict(object, newdata,
  type = c("response", "prob", "count", "zero"), na.action = na.pass,
  at = NULL, ...)
## S3 method for class 'hurdle'
residuals(object, type = c("pearson", "response"), ...)

## S3 method for class 'hurdle'
coef(object, model = c("full", "count", "zero"), ...)
## S3 method for class 'hurdle'
```

```
vcov(object, model = c("full", "count", "zero"), ...)

## S3 method for class 'hurdle'
terms(x, model = c("count", "zero"), ...)
## S3 method for class 'hurdle'
model.matrix(object, model = c("count", "zero"), ...)
```

Arguments

object, x	an object of class "hurdle" as returned by hurdle.
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
type	character specifying the type of predictions or residuals, respectively. For details see below.
na.action	function determining what should be done with missing values in newdata. The default is to predict NA.
at	optionally, if type = "prob", a numeric vector at which the probabilities are evaluated. By default 0:max(y) is used where y is the original observed response.
model	character specifying for which component of the model the terms or model matrix should be extracted.
...	currently not used.

Details

A set of standard extractor functions for fitted model objects is available for objects of class "hurdle", including methods to the generic functions print and summary which print the estimated coefficients along with some further information. The summary in particular supplies partial Wald tests based on the coefficients and the covariance matrix (estimated from the Hessian in the numerical optimization of the log-likelihood). As usual, the summary method returns an object of class "summary.hurdle" containing the relevant summary statistics which can subsequently be printed using the associated print method.

The methods for coef and vcov by default return a single vector of coefficients and their associated covariance matrix, respectively, i.e., all coefficients are concatenated. By setting the model argument, the estimates for the corresponding model component can be extracted.

Both the fitted and predict methods can compute fitted responses. The latter additionally provides the predicted density (i.e., probabilities for the observed counts), the predicted mean from the count component (without zero hurdle) and the predicted ratio of probabilities for observing a non-zero count. The latter is the ratio of probabilities for a non-zero implied by the zero hurdle component and a non-zero count in the non-truncated count distribution. See also Appendix C in Zeileis et al. (2008).

The residuals method can compute raw residuals (observed - fitted) and Pearson residuals (raw residuals scaled by square root of variance function).

The terms and model.matrix extractors can be used to extract the relevant information for either component of the model.

A logLik method is provided, hence AIC can be called to compute information criteria.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

References

Zeileis, Achim, Christian Kleiber and Simon Jackman 2008. "Regression Models for Count Data in R." *Journal of Statistical Software*, **27**(8). URL http://www.jstatsoft.org/v27/i08/.

See Also

hurdle

Examples

```
data("bioChemists", package = "pscl")
fm <- hurdle(art ~ ., data = bioChemists)

plot(residuals(fm) ~ fitted(fm))

coef(fm)
coef(fm, model = "zero")

summary(fm)
logLik(fm)
```

`predict.ideal` *predicted probabilities from an ideal object*

Description

Compute predicted probabilities from an ideal object. This predict method uses the posterior mean values of x and β to make predictions.

Usage

```
## S3 method for class 'ideal'
predict(object,
                        cutoff=.5,
                        burnin=NULL,
                        ...)

## S3 method for class 'predict.ideal'
print(x,digits=2,...)
```

Arguments

object	an object of class ideal (produced by ideal) with item parameters (beta) stored; i.e., store.item=TRUE was set when the ideal object was fitted
cutoff	numeric, a value between 0 and 1, the threshold to be used for classifying predicted probabilities of a Yea votes as predicted Yea and Nay votes.
burnin	of the recorded MCMC samples, how many to discard as burnin? Default is NULL, in which case the value of burnin in the ideal object is used.
x	object of class predict.ideal
digits	number of digits in printed object
...	further arguments passed to or from other methods.

Details

Predicted probabilities are computed using the mean of the posterior density of of x (ideal points, or latent ability) and β (bill or item parameters). The percentage correctly predicted are determined by counting the percentages of votes with predicted probabilities of a Yea vote greater than or equal to the cutoff as the threshold.

Value

An object of class predict.ideal, containing:

pred.probs	the calculated predicted probability for each legislator for each vote.
prediction	the calculated prediction (0 or 1) for each legislator for each vote.
correct	for each legislator for each vote, whether the prediction was correct.
legis.percent	for each legislator, the percent of votes correctly predicted.
vote.percent	for each vote, the percent correctly predicted.
yea.percent	the percent of yea votes correctly predicted.
nay.percent	the percent of nay votes correctly predicted.
party.percent	the average value of the percent correctly predicted by legislator, separated by party, if party information exists in the rollcall object used for ideal. If no party information is available, party.percent = NULL.
overall.percent	the total percent of votes correctly predicted.
ideal	the name of the ideal object, which can be later evaluated
desc	string, the descriptive text from the rollcall object passed to ideal

Note

When specifying a value of burnin different from that used in fitting the ideal object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the n-th iteration stored in an ideal object will not be iteration n if the user specified thin>1 in the call to ideal. Here, iterations are tagged with their iteration number. Thus, if the user called ideal with thin=10 and burnin=100 then the stored iterations are numbered 100,110,120,.... Any future subsetting via a burnin refers to this iteration number.

See Also

ideal, summary.ideal, plot.predict.ideal

Examples

```
data(s109)

f <- system.file("extdata","id1.rda",package="pscl")
load(f)
phat <- predict(id1)
phat         ## print method
```

---

predict.zeroinfl *Methods for zeroinfl Objects*

Description

Methods for extracting information from fitted zero-inflated regression model objects of class "zeroinfl".

Usage

```
## S3 method for class 'zeroinfl'
predict(object, newdata,
  type = c("response", "prob", "count", "zero"), na.action = na.pass,
  at = NULL, ...)
## S3 method for class 'zeroinfl'
residuals(object, type = c("pearson", "response"), ...)

## S3 method for class 'zeroinfl'
coef(object, model = c("full", "count", "zero"), ...)
## S3 method for class 'zeroinfl'
vcov(object, model = c("full", "count", "zero"), ...)

## S3 method for class 'zeroinfl'
terms(x, model = c("count", "zero"), ...)
## S3 method for class 'zeroinfl'
model.matrix(object, model = c("count", "zero"), ...)
```

Arguments

object, x	an object of class "zeroinfl" as returned by zeroinfl.
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
type	character specifying the type of predictions or residuals, respectively. For details see below.
na.action	function determining what should be done with missing values in newdata. The default is to predict NA.

at	optionally, if type = "prob", a numeric vector at which the probabilities are evaluated. By default 0:max(y) is used where y is the original observed response.
model	character specifying for which component of the model the terms or model matrix should be extracted.
...	currently not used.

Details

A set of standard extractor functions for fitted model objects is available for objects of class "zeroinfl", including methods to the generic functions print and summary which print the estimated coefficients along with some further information. The summary in particular supplies partial Wald tests based on the coefficients and the covariance matrix (estimated from the Hessian in the numerical optimization of the log-likelihood). As usual, the summary method returns an object of class "summary.zeroinfl" containing the relevant summary statistics which can subsequently be printed using the associated print method.

The methods for coef and vcov by default return a single vector of coefficients and their associated covariance matrix, respectively, i.e., all coefficients are concatenated. By setting the model argument, the estimates for the corresponding model components can be extracted.

Both the fitted and predict methods can compute fitted responses. The latter additionally provides the predicted density (i.e., probabilities for the observed counts), the predicted mean from the count component (without zero inflation) and the predicted probability for the zero component. The residuals method can compute raw residuals (observed - fitted) and Pearson residuals (raw residuals scaled by square root of variance function).

The terms and model.matrix extractors can be used to extract the relevant information for either component of the model.

A logLik method is provided, hence AIC can be called to compute information criteria.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

See Also

zeroinfl

Examples

```
data("bioChemists", package = "pscl")

fm_zip <- zeroinfl(art ~ ., data = bioChemists)
plot(residuals(fm_zip) ~ fitted(fm_zip))

coef(fm_zip)
coef(fm_zip, model = "count")

summary(fm_zip)
logLik(fm_zip)
```

predprob *compute predicted probabilities from fitted models*

Description

Compute predicted probabilities from fitted models, optionally at new covariate values.

Usage

```
predprob(obj, ...)
```

Arguments

obj	fitted model object
...	other arguments

Details

See documentation for specific methods.

Value

A matrix of predicted probabilities, each row a vector of predicted probabilities over the range of responses seen in the data (i.e., `min(y):max(y)`), conditional on the values of covariates.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

predprob.glm, predprob.zeroinfl

Examples

```
## Not run:
data("bioChemists")
zip <- zeroinfl(art ~ . | ., data = bioChemists, EM = TRUE)
phat <- predprob(zip)

newdata <- expand.grid(list(fem="Men",mar="Married",
                            kid5=1,phd=3.103,
                            ment=0:77))
phat <- predprob(zip, newdata = newdata)

## End(Not run)
```

predprob.glm *Predicted Probabilities for GLM Fits*

Description

Obtains predicted probabilities from a fitted generalized linear model object.

Usage

```
## S3 method for class 'glm'
predprob(obj, newdata = NULL, at = NULL, ...)
```

Arguments

obj	a fitted object of class inheriting from "glm"
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
at	an optional numeric vector at which the probabilities are evaluated. By default 0:max(y) where y is the original observed response.
...	arguments passed to or from other methods

Details

This method is only defined for glm objects with family=binomial or family=poisson, or negative binomial count models fit with the glm.nb function in library(MASS).

Value

A matrix of predicted probabilities. Each row in the matrix is a vector of probabilities, assigning predicted probabilities over the range of responses actually observed in the data. For instance, for models with family=binomial, the matrix has two columns for the "zero" (or failure) and "one" (success) outcomes, respectively, and trivially, each row in the matrix sums to 1.0. For counts fit with family=poisson or via glm.nb, the matrix has length(0:max(y)) columns. Each observation used in fitting the model generates a row to the returned matrix; alternatively, if newdata is supplied, the returned matrix will have as many rows as in newdata.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

predict.glm

Examples

```
data(bioChemists)
glm1 <- glm(art ~ .,
            data=bioChemists,
            family=poisson,
            trace=TRUE)  ## poisson GLM
phat <- predprob(glm1)
apply(phat,1,sum)                   ## almost all 1.0
```

`predprob.ideal` *predicted probabilities from fitting ideal to rollcall data*

Description

Computes predicted probabilities of a "Yea" vote conditional on the posterior means of the legislators' ideal points and vote-specific parameters.

Usage

```
## S3 method for class 'ideal'
predprob(obj, ...)
```

Arguments

`obj`	An object of class `ideal`
`...`	Arguments to be passed to other functions

Details

This is a wrapper function to `predict.ideal`, extracting just the predicted probabilities component of the object returned by that function. Predicted probabilities can and are generated for each voting decision, irrespective of whether the legislator actually voted on any particular roll call.

Value

A `matrix` of dimension `n` (number of legislators) by `m` (number of roll call votes).

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

`ideal`, `predprob`, `predict.ideal`

Examples

```
f <- system.file("extdata","id1.rda",package="pscl")
load(f)
phat <- predprob(id1)
dim(phat)
```

`presidentialElections` *elections for U.S. President, 1932-2016, by state*

Description

Democratic share of the presidential vote, 1932-2016, in each state and the District of Columbia.

Usage

```
data(presidentialElections)
```

Format

- `state`character, name of state
- demVotenumeric, percent of the vote for president won by the Democratic candidate
- yearnumeric, integer
- southlogical, `TRUE` if state is one of the 11 states of the former Confederacy

Note

1,047 observations, unbalanced panel data in long format. Hawaii and Alaska contribute data from 1960 onwards the District of Columbia contributes data from 1964 onward; Alabama has missing data for 1948 and 1964.

Source

David Leip's Atlas of U.S. Presidential Elections https://uselectionatlas.org

Examples

```
data(presidentialElections)
if(require(lattice)) {
    lattice::xyplot(demVote ~ year | state,
       panel=lattice::panel.lines,
       ylab="Democratic Vote for President (percent)",
       xlab="Year",
       data=presidentialElections,
       scales=list(y=list(cex=.6),x=list(cex=.35)),
       strip=strip.custom(par.strip.text=list(cex=.6)))
}

## Obama vs Kerry, except DC
```

```
y08 <- presidentialElections$year==2008
y04 <- presidentialElections$year==2004
tmpData <- merge(y=presidentialElections[y08,],
         x=presidentialElections[y04,],
                by="state")
tmpData <- tmpData[tmpData$state!="DC",]
xlim <- range(tmpData$demVote.x,tmpData$demVote.y)
col <- rep("black",dim(tmpData)[1])
col[tmpData$south.x] <- "red"

plot(demVote.y ~ demVote.x,
     xlab="Kerry Vote Share, 2004 (percent)",
     ylab="Obama Vote Share, 2008 (percent)",
     xlim=xlim,
     ylim=xlim,
     type="n",
     las=1,
     data=tmpData)
abline(0,1,lwd=2,col=gray(.65))
ols <- lm(demVote.y ~ demVote.x,
          data=tmpData)
abline(ols,lwd=2)

text(tmpData$demVote.x,
     tmpData$demVote.y,
     tmpData$state,
     col=col,
     cex=.65)
legend(x="topleft",
       bty="n",
       lwd=c(2,2),
       col=c(gray(.65),"black"),
       legend=c("No Change from 2004","Regression"))
legend(x="bottomright",
       bty="n",
       text.col=c("red","black"),
       legend=c("South","Non-South"))
```

`prussian` *Prussian army horse kick data*

Description

Deaths by year, by corp, from horse kicks.

Usage

```
data(prussian)
```

Format

A data frame with 280 observations on the following 3 variables.

`y` a numeric vector, count of deaths

`year` a numeric vector, 18XX, year of observation

`corp` a `factor`, corp of Prussian Army generating observation

Source

von Bortkiewicz, L. 1898. *Das Gesetz der Kleinen Zahlen.* Leipzig: Teubner.

Examples

```
data(prussian)
corpP <- glm(y ~ corp, family=poisson,data=prussian)
summary(corpP)
```

`readKH` *read roll call data in Poole-Rosenthal KH format*

Description

Creates a `rollcall` object from the flat file format for roll call data used by Keith Poole and Howard Rosenthal.

Usage

```
readKH(file,
       dtl=NULL,
       yea=c(1,2,3),
       nay=c(4,5,6),
       missing=c(7,8,9),
       notInLegis=0,
       desc=NULL,
       debug=FALSE)
```

Arguments

`file`	string, name of a file or URL holding KH data
`dtl`	string, name of a file or URL holding KH `dtl` file (information about votes); default is `NULL`, indicating no `dtl` file
`yea`	numeric, possibly a vector, code(s) for a Yea vote in the rollcall context (or a correct answer in the educational testing context). Default is `c(1,2,3)`, which corresponds to Yea, Paired Yea, and Announced Yea in Poole/Rosenthal data files.

nay	numeric, possibly a vector, code(s) for a Nay vote in the rollcall context (or an incorrect answer in the educational testing context). Default is c(4,5,6), which corresponds to Announced Nay, Paired Nay, and Nay in Poole/Rosenthal data files.
missing	numeric and/or NA, possible a vector, code(s) for missing data. Default is c(0,7,8,9,NA); the first four codes correspond to Not Yet a Member, Present (some Congresses), Present (some Congresses), and Not Voting.
notInLegis	numeric or NA, possibly a vector, code(s) for the legislator not being in the legislature when a particular roll call was recorded (e.g., deceased, retired, yet to be elected). Default is 0 for Poole/Rosenthal data files.
desc	string, describing the data, e.g., 82nd U.S. House of Representatives; default is NULL
debug	logical, print debugging information for net connection

Details

Keith Poole and Howard Rosenthal have gathered an impressive collection of roll call data, spanning every roll call cast in the United States Congress. This effort continues now as a real-time exercise, via a collaboration with Jeff Lewis (109th Congress onwards). Nolan McCarty collaborated on the compilation of roll call data for the 102nd through 108th Congress.

This function relies on some hard-coded features of Poole-Rosenthal flat files, and assumes that the file being supplied has the following structure (variable, start-end columns):

ICPSR legislator unique ID 4-8

ICPSR state ID 9-10

Congressional District 11-12

state name 13-20

party code 21-23

legislator name 26-36

roll-call voting record 37 to end-of-record

This function reads data files in that format, and creates a rollcall, for which there are useful methods such as summary.rollcall. The legis.data component of the rollcall object is a data.frame which contains:

state a 2-character string abbreviation of each legislator' state

icpsrState a 2-digit numeric code for each legislator's state, as used by the Inter-university Consortium for Political and Social Research (ICPSR)

cd numeric, the number of each legislator's congressional district within each state; this is always 0 for members of the Senate

icpsrLegis a unique numeric identifier for each legislator assigned by the ICPSR, as corrected by Poole and Rosenthal.

partyName character string, the name of each legislator's political party

party numeric, code for each legislator's political party; see http://legacy.voteview.com/PARTY3.HTM

The rownames attribute of this data frame is a concatenation of the legislators' names, party abbreviations (for Democrats and Republicans) and state, and (where appropriate), a district number; e.g., Bonner (R AL-1). This tag is also provided in the legis.name component of the returned rollcall object.

Poole and Rosenthal also make dtl files available for Congresses 1 through 106. These files contain information about the votes themselves, in a multiple-line per vote ascii format, and reside in the dtl director of Poole's web site, e.g., https://legacy.voteview.com/k7ftp/dtl/102s.dtl is the dtl file for the 102nd Senate. The default is to presume that no such file exists. When a dtl file is available, and is read, the votes.data attribute of the resulting rollcall object is a data.frame with one record per vote, with the following variables:

date vector of class Date, date of the rollcall, if available; otherwise NULL

description vector of mode character, descriptive text

The dtl files are presumed to have the date of the rollcall in the first line of text for each roll call, and lines 3 onwards contain descriptive text.

Finally, note also that the Poole/Rosenthal data sets often include the U.S. President as a pseudo-legislator, adding the announced positions of a president or the administration to the roll call matrix. This adds an extra "legislator" to the data set and can sometimes produce surprising results (e.g., a U.S. Senate of 101 senators), and a "legislator" with a surprisingly low party loyalty score (since the President/administration only announces positions on a relatively small fraction of all Congressional roll calls).

Value

an object of class rollcall, with components created using the identifying information in the Poole/Rosenthal files. If the function can not read the file (e.g., the user specified a URL and the machine is not connected to the Internet), the function fails with an error message (set debug=TRUE to help resolve these issues).

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

References

Poole, Keith and Howard Rosenthal. 1997. *Congress: A Political-Economic History of Roll Call Voting*. New York: Oxford University Press.

Poole, Keith. http://legacy.voteview.com

Rosenthal, Howard L. and Keith T. Poole. *United States Congressional Roll Call Voting Records, 1789-1990: Reformatted Data [computer file].* 2nd ICPSR release. Pittsburgh, PA: Howard L. Rosenthal and Keith T. Poole, Carnegie Mellon University, Graduate School of Industrial Administration [producers], 1991. Ann Arbor, MI: Inter-university Consortium for Political and Social Research [distributor], 2000. http://www.icpsr.umich.edu/icpsrweb/ICPSR/studies/09822

See Also

rollcall

Examples

```
## Not run:
h107 <- readKH("https://voteview.com/static/data/out/votes/H107_votes.ord",
                desc="107th U.S. House of Representatives")

s107 <- readKH("https://voteview.com/static/data/out/votes/S107_votes.ord",
                desc="107th U.S. Senate")

## End(Not run)
```

RockTheVote *Voter turnout experiment, using Rock The Vote ads*

Description

Voter turnout data spanning 85 cable TV systems, randomly allocated to a voter mobilization experiment targeting 18-19 year olds with "Rock the Vote" television advertisements

Usage

```
data(RockTheVote)
```

Format

A data frame with 85 observations on the following 6 variables.

`strata` numeric, experimental strata

`treated` numeric, 1 if a treated cable system, 0 otherwise

`r` numeric, number of 18 and 19 year olds turning out

`n` numeric, number of 19 and 19 year olds registered

`p` numeric, proportion of 18 and 19 year olds turning out

`treatedIndex` numeric, a counter indexing the 42 treated units

Details

Green and Vavreck (2008) implemented a cluster-randomized experimental design in assessing the effects of a voter mobilization treatment in the 2004 U.S. Presidential election. The clusters in this design are geographic areas served by a single cable television system. So as to facilitate analysis, the researchers restricted their attention to small cable systems whose reach is limited to a single zip code. Further, since the experiment was fielded during the last week of the presidential election, the researchers restricted their search to cable systems that were not in the 16 hotly-contested "battleground" states (as designated by the *Los Angeles Times*).

Eighty-five cable systems were available for randomization and were assigned to treatment after stratification on previous turnout levels in presidential elections (as determined from analysis of the corresponding states' voter registration files). Each cable system was matched with one or sometimes two other cable systems in the same state, yielding 40 strata. Then within each strata,

cable systems were randomly assigned to treatment and control conditions. Strata 3, 8 and 25 have two control cable systems and 1 treated system each, while strata 6 and 20 have two treated cable systems and one control system. The remaining 35 strata have 1 treated cable system and 1 control system. In this way there are 38 + 4 = 42 treated systems, spanning 40 experiment strata.

The treatment involved researchers purchasing prime-time advertising spots on four channels in the respective cable system in which the researchers aired voter mobilization ads. The ads were produced by *Rock the Vote*, targeted at younger voters, and aired four times per night, per channel, over the last eight days of the election campaign. After the election, public records were consulted to assemble data on turnout levels in the treated and control cable systems. In the analysis reported in Green and Vavreck (2008), the researchers focused on turnout among registered voters aged 18 and 19 years old.

References

Green, Donald P. and Lynn Vavreck. 2008. Analysis of Cluster-Randomized Experiments: A Comparison of Alternative Estimation Approaches. *Political Analysis* 16:138-152.

Jackman, Simon, 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Hoboken, New Jersey. Example 7.9.

Examples

```
data(RockTheVote)
## estimate MLEs of treatment effects
deltaFunction <- function(data){
  model <- glm(cbind(r,n-r)~treated,
               data=data,
               family=binomial)
  c(coef(model)[2],
    confint(model)[2,])
}


tmp <- by(RockTheVote,
          as.factor(RockTheVote$strata),
          deltaFunction)

tmp <- matrix(unlist(tmp),ncol=3,byrow=TRUE)

indx <- order(tmp[,1])

plot(y=1:40,
     x=tmp[indx,1],
     pch=16,cex=1.25,
     xlim=range(tmp),
     ylab="",
     axes=FALSE,
     xlab="Estimated Treatment Effect (MLEs, Logit Scale)")
text(y=1:40,
     x=par()$usr[1],
     pos=4,
     as.character((1:40)[indx]),
```

```
     cex=.5)
segments(x0=tmp[indx,2],
         x1=tmp[indx,3],
         y0=1:40,
         y1=1:40)
axis(1)
axis(3)
abline(v=0)
```

`rollcall` *create an object of class rollcall*

Description

Create a `rollcall` object, used for the analysis of legislative voting or, equivalently, item-response modeling of binary data produced by standardized tests, etc.

Usage

```
rollcall(data,
         yea=1, nay=0, missing=NA, notInLegis=9,
         legis.names=NULL, vote.names=NULL,
         legis.data=NULL, vote.data=NULL,
         desc=NULL, source=NULL)
```

Arguments

`data`	voting decisions (for roll calls), or test results (for IRT). Can be in one of two forms. First, `data` may be a `matrix`, with rows corresponding to legislators (subjects) and columns to roll calls (test items). `data` can also be a `list` with an element named `votes` containing the matrix described above.
`yea`	numeric, possibly a vector, code(s) for a Yea vote in the rollcall context, or a correct answer in the educational testing context. Default is 1.
`nay`	numeric, possibly a vector, code(s) for a Nay vote in the rollcall context, or an incorrect answer in the educational testing context. Default is 0.
`missing`	numeric or `NA`, possibly a vector, code(s) for missing data. Default is `NA`.
`notInLegis`	numeric or `NA`, possibly a vector, code(s) for the legislator not being in the legislature when a particular roll call was recorded (e.g., deceased, retired, yet to be elected).
`legis.names`	a vector of names of the legislators or individuals. If `data` is a `list` or `data.frame` and has a component named `legis.names`, then this will be used. Names will be generated if not supplied, or if there are fewer unique names supplied than legislators/subjects (rows of the roll call matrix).
`vote.names`	a vector of names or labels for the votes or items. If `data` is a `list` or `data.frame` and has a component named `vote.names`, then this will be used. Names will be generated if not supplied, or if there are fewer unique names supplied than votes/test-items (columns of the roll call matrix).

legis.data	a matrix or data.frame containing covariates specific to each legislator/test-taker; e.g., party affiliation, district-level covariates. If this object does not have the same number of rows as data, an error is returned.
vote.data	a matrix or data.frame containing covariates specific to each roll call vote or test item; e.g., a timestamp, the bill sponsor, descriptive text indicating the type of vote. If this object does not have the same number of row as the number of columns in data, an error is returned.
desc	character, a string providing an (optional) description of the data being used. If data is a list and contains an element named desc, then this will be used.
source	character, a string providing an (optional) description of where the roll call data originated (e.g., a URL or a short-form reference). Used in print and summary methods.

Details

See below for methods that operate on objects of class rollcall.

Value

An object of class rollcall, a list with the following components:

votes	a matrix containing voting decisions, with rows corresponding to legislators (test subjects) and columns to roll call votes (test items). Legislators (test subjects) and items (or votes) have been labeled in the dimnames attribute of this matrix, using the legis.names and/or vote.names arguments, respectively.
codes	a list with named components yea, nay, notInLegis and missing, each component a numeric vector (possibly of length 1 and possibly NA), indicating how entries in the votes component of the rollcall object should be considered. This list simply gathers up the values in the yea, nay, notInLegis and missing arguments passed to the function.
n	numeric, number of legislators, equal to dim(votes)[1]
m	numeric, number of votes, equal to dim(votes)[2]
legis.data	user-supplied data on legislators/test-subjects.
vote.data	user-supplied data on rollcall votes/test-items.
desc	any user-supplied description. If no description was provided, defaults desc defaults to NULL.
source	any user-supplied source information (e.g., a url or a short-form reference). If no description is provided, source defaults to NULL.

See Also

readKH for creating objects from files (possibly over the web), in the format used for data from the United States Congress used by Keith Poole and Howard Rosenthal (and others).

summary.rollcall, ideal for model fitting.

Examples

```
## generate some fake roll call data
set.seed(314159265)
fakeData <- matrix(sample(x=c(0,1),size=5000,replace=TRUE),
                   50,100)
rc <- rollcall(fakeData)
is(rc,"rollcall")        ## TRUE
rc                       ## print the rollcall object on screen

data(sc9497)             ## Supreme Court example data
rc <- rollcall(data=sc9497$votes,
               legis.names=sc9497$legis.names,
               desc=sc9497$desc)
summary(rc,verbose=TRUE)

## Not run:
## s107
## could use readKH for this
dat <- readLines("sen107kh.ord")
dat <- substring(dat,37)
mat <- matrix(NA,ncol=nchar(dat[1]),nrow=length(dat))
for(i in 1:103){
  mat[i,] <- as.numeric(unlist(strsplit(dat[i],
                                        split=character(0))))
}

s107 <- rollcall(mat,
                 yea=c(1,2,3),
                 nay=c(4,5,6),
                 missing=c(7,8,9),
                 notInLegis=0,
                 desc="107th U.S. Senate",
                 source="http://voteview.ucsd.edu")
summary(s107)

## End(Not run)
```

---

s109 *rollcall object, 109th U.S. Senate (2005-06).*

Description

A sample rollcall object, generated using a collection of the rollcalls of the 109th U.S. Senate (2005-2006).

Usage

```
data(s109)
```

Format

A rollcall object containing the recorded votes of the 109th U.S. Senate, plus information identifying the legislators and the rollcalls.

Details

Note the coding scheme used by Poole and Rosenthal; Yea (1), Nay (6) etc.

Source

Keith Poole's web site: https://legacy.voteview.com/senate109.htm

Originally scraped from the Senate's web site by Jeff Lewis (UCLA).

Information identifying the votes is available at https://voteview.com/static/data/out/rollcalls/S109_rollcalls.csv

Examples

```
require(pscl)
data(s109)
is(s109,"rollcall")    ## TRUE
s109                   ## print method for class rollcall
summary(s109)          ## summary method
summary(s109,verbose=TRUE)
## Not run:
## how s109 was created
require(pscl)
s109 <- readKH("https://voteview.com/static/data/out/votes/S109_votes.ord",
               desc="109th U.S. Senate",
               debug=TRUE)
url <- "https://voteview.com/static/data/out/rollcalls/S109_rollcalls.csv"

s109$vote.data <- data.frame(read.csv(file=url,header=TRUE))
s109$vote.data$date <- as.Date(s109$vote.data$date,
                               format="
dimnames(s109$votes)[[2]] <- paste(s109$vote.data$session,
                                   s109$vote.data$number,sep="-")

## End(Not run)
```

---

sc9497 *votes from the United States Supreme Court, from 1994-1997*

Description

This data set provides information on the United States Supreme Court from 1994-1997. Votes included are non-unanimous.

Usage

```
data(sc9497)
```

Format

A list containing the elements:

votes a matrix of the votes, 0=Nay, 1=Yea, NA=Abstained or missing data. The matrix columns are labeled with `vote.names` and the rows are labeled with `legis.names`.

legis.names a vector of the names of the nine Justices sitting on the court at this time.

party NULL; exists for consistency with House and Senate data sets.

state NULL; exists for consistency with House and Senate data sets.

district NULL; exists for consistency with House data sets.

id NULL; exists for consistency with House and Senate data sets.

vote.names a vector of strings numbering the cases simply to distinguish them from one another.

desc a description of the data set.

Source

Harold J. Spaeth (1999). *United States Supreme Court Judicial Database, 1953-1997 Terms*. Ninth edition. Inter-university Consortium for Political and Social Research. Ann Arbor, Michigan. http://www.icpsr.umich.edu/

---

seatsVotes *A class for creating seats-votes curves*

Description

Convert a vector of vote shares into a seats-vote curve object, providing estimates of partisan bias.

Usage

```
seatsVotes(x, desc = NULL, method = "uniformSwing")
```

Arguments

x	a vector of vote shares for a specific party (either proportions or percentages)
desc	descriptive text
method	how to simulate a seats-vote curve; the only supported method at this stage is `uniformSwing`.

Details

Simulation methods are required to induce a seats-votes curve given a vector of vote shares from one election. The uniform swing method simply slides the empirical distribution function of the vote shares "up" and "down", computing the proportion of the vote shares that lie above .5 (by construction, the winning percentage in a two-party election) for each new location of the vector of vote shares. That is, as the empirical CDF of the observed vote shares slides up or down, more or less seats cross the .5 threshold. A seats-votes curve is formed by plotting the seat share above .5 as a function of the average district-level vote share (a weakly monotone function, since the empirical CDF constitutes a set of sufficient statistics for this problem). The simulation is run so as to ensure that average district-level vote shares range between 0 and 1.

The extent to which the seats-votes curve departs from symmetry is known as bias. More specifically, the vertical displacement of the seats-votes curve from .5 when average district-level vote share is .5 is conventionally reported as an estimate of the bias of the electoral system.

Different methods produce different estimates of seats-votes curves and summary estimands such as bias. The uniform swing method is completely deterministic and does not produce any uncertainty assessment (e.g., confidence intervals etc).

Value

An object of class `seatsVotes`, with components

`s`	Estimated seat shares over the range of simulated average, district-level vote shares
`v`	Simulated average district-level vote shares
`x`	observed seat shares, with missing data removed
`desc`	user-supplied descriptive character string
`call`	a list of class `call`, the call to the function

Note

Additional methods to come later.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

References

Tufte, Edward R. 1973. The Relationship Between Seats and Votes in Two-Party Systems. *American Political Science Review*. 67(2):540-554.

Gelman, Andrew and Gary King. 1990. Estimating the Consequences of Electoral Redistricting. *Journal of the American Statistical Association*. 85:274-282.

Jackman, Simon. 1994. Measuring Electoral Bias: Australia, 1949-93. *British Journal of Political Science*. 24(3):319-357.

See Also

`plot.seatsVotes` for plotting methods.

Examples

```
data(ca2006)
x <- ca2006$D/(ca2006$D+ca2006$R)
sv <- seatsVotes(x,
                 desc="Democratic Vote Shares, California 2006 congressional elections")
```

`simpi` *Monte Carlo estimate of pi (3.14159265...)*

Description

Monte Carlo estimation of pi

Usage

```
simpi(n)
```

Arguments

`n`	integer, number of Monte Carlo samples, defaults to 1000

Details

A crude Monte Carlo estimate of π can be formed as follows. Sample from the unit square many times (i.e., each sample is formed with two independent draws from a uniform density on the unit interval). Compute the proportion p of sampled points that lie inside a unit circle centered on the origin; such points (x, y) have distance from the origin $d = \sqrt{x^2 + y^2}$ less than 1. Four times p is a Monte Carlo estimate of π. This function is a wrapper to a simple C function, bringing noticeable speed gains and memory efficiencies over implementations in native R.

Contrast this Monte Carlo method with Buffon's needle and refinements thereof (see the discussion in Ripley (1987, 193ff).

Value

the Monte Carlo estimate of π

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

References

Ripley, Brain D. 1987 [2006]. *Stochastic Simulation*. Wiley: Hoboken, New Jersey.

Examples

```
seed <- round(pi*10000)  ## hah hah hah
m <- 6
z <- rep(NA,m)
lim <- rep(NA,m)
for(i in 1:m){
  cat(paste("simulation for ",i,"\n"))
  set.seed(seed)
  timings <- system.time(z[i] <- simpi(10^i))
  print(timings)
  cat("\n")
  lim[i] <- qbinom(prob=pi/4,size=10^i,.975)/10^i * 4
}

## convert to squared error
z <-(z - pi)^2
lim <- (lim - pi)^2

plot(x=1:m,
     y=z,
     type="b",
     pch=16,
     log="y",
     axes=FALSE,
     ylim=range(z,lim),
     xlab="Monte Carlo Samples",
     ylab="Log Squared Error")
lines(1:m,lim,col="blue",type="b",pch=1)
legend(x="topright",
       legend=c("95% bound",
         "Realized"),
       pch=c(1,16),
       lty=c(1,1),
       col=c("blue","black"),
       bty="n")
axis(1,at=1:m,
     labels=c(expression(10^{1}),
       expression(10^{2}),
       expression(10^{3}),
       expression(10^{4}),
       expression(10^{5}),
       expression(10^{6})))
axis(2)
```

`state.info` *information about the American states needed for U.S. Congress*

Description

Numeric codes and names of 50 states and the District of Columbia, required to parse Keith Poole and Howard Rosenthal's collections of U.S. Congressional roll calls.

Usage

```
data(state.info)
```

Format

`icpsr` integer, numeric code for state used by the Inter-university Consortium for Political and Social Research

`state` character, name of state or `Washington D.C.`

`year` numeric or `NA`, year of statehood

Details

The function `readKH` converts the integer ICPSR codes into strings, via a table lookup in this data frame. Another table lookup in `state.abb` provides the 2-letter abbreviation commonly used in identifying American legislators, e.g., `KENNEDY,E (D-MA)`.

Source

Various ICPSR codebooks. http://www.icpsr.umich.edu

See Also

`state`

---

summary.ideal *summary of an ideal object*

Description

Provides a summary of the output from ideal point estimation contained in an object of class `ideal`.

Usage

```
## S3 method for class 'ideal'
summary(object, prob=.95,
              burnin=NULL,
              sort=TRUE,
              include.beta=FALSE,...)
```

Arguments

object	an object of class ideal.
prob	scalar, a proportion between 0 and 1, the content of the highest posterior density (HPD) interval to compute for the parameters
burnin	of the recorded MCMC samples, how many to discard as burnin? Default is NULL, in which case the value of burnin in the ideal object is used.
sort	logical, default is TRUE, indicating that the summary of the ideal points be sorted by the estimated posterior means (lowest to highest)
include.beta	whether or not to calculate summary statistics of beta, if beta is available. If the item parameters were not stored in the ideal object, then include.beta is ignored.
...	further arguments passed to or from other functions

Details

The test of whether a given discrimination parameter is distinguishable from zero first checks to see if the two most extreme quantiles are symmetric around .5 (e.g., as are the default value of .025 and .975). If so, the corresponding quantiles of the MCMC samples for each discrimination parameter are inspected to see if they have the same sign. If they do, then the corresponding discrimination parameter is flagged as distinguishable from zero; otherwise not.

Value

An item of class summary.ideal with elements:

object	the name of the ideal object as an unevaluated expression, produced by match.call()$object
xm	n by d matrix of posterior means for the ideal points
xsd	n by d matrix of posterior means for the ideal points
xHDR	n by 2 by d array of HDRs for the ideal points
bm	m by d+1 matrix of posterior means for the item parameters
bsd	m by d+1 matrix of posterior standard deviation for the item parameters
bHDR	m by 2 by d+1 array of HDRs for the item parameters
bSig	a list of length d, each component a vector of length m, of mode logical, equal to TRUE if the corresponding discrimination parameter is distinguishable from zero; see Details. If store.item was set to FALSE when ideal was invoked, then bSig is a list of length zero.
party.quant	if party information is available through the rollcall object that was used to run ideal, then party.quant gives the posterior mean of the legislators' ideal points by party, by dimension. If no party information is available, then party.quant=NULL.

Note

When specifying a value of burnin different from that used in fitting the ideal object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the n-th iteration stored in an ideal object will not be iteration n if the user specified thin>1 in the call to ideal. Here, iterations are tagged with their iteration number. Thus, if the user called ideal with thin=10 and burnin=100 then the stored iterations are numbered 100,110,120,.... Any future subsetting via a burnin refers to this iteration number.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

ideal

Examples

```
f <- system.file("extdata","id1.rda",package="pscl")
load(f)
summary(id1)

## Not run:
data(s109)
cl2 <- constrain.legis(s109,
                       x=list("KENNEDY (D MA)"=c(-1,0),
                         "ENZI (R WY)"=c(1,0),
                         "CHAFEE (R RI)"=c(0,-.5)),
                       d=2)
id2Constrained <- ideal(s109,
                        d=2,
                        priors=cl2,      ## priors (w constraints)
                        startvals=cl2,   ## start value (w constraints)
                        store.item=TRUE,
                        maxiter=5000,
                        burnin=500,
                        thin=25)

summary(id2Constrained,
        include.items=TRUE)

## End(Not run)
```

---

summary.rollcall *summarize a rollcall object*

Description

Provides a summary of the information about votes, legislators, etc in a rollcall object.

Usage

```
## S3 method for class 'rollcall'
summary(object,
                dropList=NULL,
                verbose=FALSE,debug=FALSE,...)

## S3 method for class 'summary.rollcall'
print(x, digits=1, ...)
```

Arguments

object	an rollcall object.
dropList	a list or alist, listing voting decisions, legislators and/or votes to be dropped from the summary; see dropRollCall for details.
verbose	logical, if TRUE, compute legislator-specific and vote-specific Yea/Nay/NA summaries
debug	logical, if TRUE, print messages to console during processing of the rollcall object
x	an object of class summary.rollcall
digits	number of decimal places in printed display
...	further arguments passed to or from other methods.

Value

An object of class summary.rollcall with the following elements (depending on the logical flag verbose):

n	number of legislators in the rollcall object, after processing the dropList
m	number of roll call votes in the rollcall object, after processing the dropList
codes	a list that describes how the voting decisions in the rollcall matrix (object$votes) map into "Yea" and "Nay" etc, after processing the dropList; see rollcall for more details
allVotes	a matrix containing a tabular breakdown of all votes in the rollcall matrix (object$votes), after processing the dropList
partyTab	a tabular breakdown of the legislators' party affiliations, after processing the dropList, and only if party affiliations are supplied asobject$legis.data$party; see rollcall for details
lopSided	a tabular summary of the frequency of lop-sided roll call votes in the rollcall object, again, after processing the dropList
legisTab	a tabular summary of each legislators' voting history
partyLoyalty	the proportion of times that each legislator votes the way that a majority of his or her fellow partisans did, provided party affiliations are available
voteTab	a tabular summary of each rollcall's votes
call	the matched call used to invoke summary.rollcall

See Also

rollcall

Examples

```
set.seed(314159265)
fakeData <- matrix(sample(x=c(0,1),size=1000,replace=TRUE),
                   10,100)
rc <- rollcall(fakeData)
rc

data(sc9497)
rc <- rollcall(sc9497)
summary(rc)

data(s109)
summary(s109)
summary(s109,verbose=TRUE)
```

---

tracex *trace plot of MCMC iterates, posterior density of legislators' ideal points*

Description

Produces a trace plot of the MCMC samples from the posterior density of legislators' ideal points.

Usage

```
tracex(object, legis=NULL, d=1, conf.int=0.95,
       multi = FALSE, burnin=NULL,span=.25,
       legendLoc="topright")
```

Arguments

object	an object of class ideal.
legis	a vector of either the names of legislators (or partial matches of the names as given in the dimnames of object$x.
d	numeric, either a scalar or a vector of length two, the dimension(s) to be traced.
conf.int	numeric, the level of the confidence interval on the posterior mean to be plotted.
multi	logical, multiple plotting panels, one per legislators? If FALSE (default) and length(d)==2, display traces for all selected legislators' ideal points on the one plot.
burnin	of the recorded MCMC samples, how many to discard as burnin? Default is NULL, in which case the value of burnin in the ideal object is used.

span	numeric, a proportion, the span to be used when calling loess to generate a moving average for trace plots when d=1
legendLoc	numeric or character, and possibly a vector, specifying where to place the legend when d=1; setting legendLoc=NULL will suppress the legend for all requested trace plots

Details

Produces a trace plot showing the history of the MCMC iterations for the ideal point of each of the legislators (partially) named in legis. For d=1, each trace plot includes a trace over iterations, the cumulative mean, a moving average, the MCMC-based estimate of the mean of the posterior, and a confidence interval (specified by conf.int) around the mean of the posterior (using the estimated quantiles) of the respective MCMC iterates). All of these values are calculated discarding the initial burnin iterations.

When d is a vector of length two, a 2-dimensional trace plot is displayed, with the d[1] dimension on the horizontal axis, and the d[2] dimension on the vertical axis.

When d=1, a legend will be placed on the plot; the option legendLoc controls the placing of the legend. legendLoc may be a vector, specifying a unique legend location for each requested trace plots. If legendLoc is of length 1, it will be replicated to have length equal to the number of requested trace plots.

See Also

ideal; pmatch for matching legislators' names. See legend for valid options to legendLoc.

Examples

```
data(s109)
f <- system.file("extdata","id1.rda",package="pscl")
load(f)
tracex(id1,legis="KENN")

## n.b., no such legislator named Thomas Bayes
tracex(id1,legis=c("KENN","BOX","KYL","Thomas Bayes"))


f <- system.file("extdata","id2.rda",package="pscl")
load(f)

tracex(id2,d=1,legis=c("KENNEDY","BOXER","KYL","Thomas Bayes"))
tracex(id2,d=2,legis=c("KENNEDY","BOXER","KYL","Thomas Bayes"))
tracex(id2,d=1:2,
       legis=c("KENNEDY","BOXER","KYL","Thomas Bayes"))

## partial matching
tracex(id2,d=1:2,
       legis=c("KENN","BOX","BID","SNO","SPEC","MCCA","KYL",
               "Thomas Bayes"),
       multi=TRUE)
```

`UKHouseOfCommons` *1992 United Kingdom electoral returns*

Description

Electoral returns, selected constituencies, 1992 general election for the British House of Commons

Usage

```
data(UKHouseOfCommons)
```

Format

A data frame with 521 observations on the following 12 variables.

`constituency` a character vector, name of the House of Commons constituency

`county` a character vector, county of the House of Commons constituency

`y1` a numeric vector, log-odds of Conservative to LibDem vote share

`y2` a numeric vector, log-odds of Labor to LibDem vote share

`y1lag` a numeric vector, `y1` from previous election

`y2lag` a numeric vector, `y2` from previous election

`coninc` a numeric vector, 1 if the incumbent is a Conservative, 0 otherwise

`labinc` a numeric vector, 1 if the incumbent is from the Labor Party, 0 otherwise

`libinc` a numeric vector, 1 if the incumbent is from the LibDems, 0 otherwise

`v1` a numeric vector, Conservative vote share (proportion of 3 party vote)

`v2` a numeric vector, Labor vote share (proportion of 3 party vote)

`v3` a numeric vector, LibDem vote share (proportion of 3 party vote)

Details

These data span only 521 of the 621 seats in the House of Commons at the time of 1992 election. Seats missing either a Conservative, Labor, or a LibDem candidate appear to have been dropped.

The original Katz and King data set does not have case labels. I used matches to an additional data source to recover a set of constituency labels for these data; labels could not recovered for two of the constituencies.

Source

Jonathan Katz; Gary King. 1999. "Replication data for: A Statistical Model of Multiparty Electoral Data", http://hdl.handle.net/1902.1/QIGTWZYTLZ

References

Katz, Jonathan and Gary King. 1999. "A Statistical Model for Multiparty Electoral Data". *American Political Science Review*. 93(1): 15-32.

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Chichester. Example 6.9.

Examples

```
data(UKHouseOfCommons)
tmp <- UKHouseOfCommons[,c("v1","v2","v3")]
summary(apply(tmp,1,sum))

col <- rep("black",dim(tmp)[1])
col[UKHouseOfCommons$coninc==1] <- "blue"
col[UKHouseOfCommons$labinc==1] <- "red"
col[UKHouseOfCommons$libinc==1] <- "orange"

library(vcd)
vcd::ternaryplot(tmp,
                 dimnames=c("Cons","Lab","Lib-Dem"),
                 labels="outside",
                 col=col,
                 pch=1,
                 main="1992 UK House of Commons Election",
                 cex=.75)
```

---

unionDensity *cross national rates of trade union density*

Description

Cross-national data on relative size of the trade unions and predictors, in 20 countries. Two of the predictors are highly collinear, and are the source of a debate between Stephens and Wallerstein (1991), later reviewed by Western and Jackman (1994).

Usage

```
data(unionDensity)
```

Format

- `union`numeric, percentage of the total number of wage and salary earners plus the unemployed who are union members, measured between 1975 and 1980, with most of the data drawn from 1979
- `left`numeric, an index tapping the extent to which parties of the left have controlled governments since 1919, due to Wilensky (1981).
- `size`numeric, log of labor force size, defined as the number of wage and salary earners, plus the unemployed

- concennumeric, percentage of employment, shipments, or production accounted for by the four largest enterprises in a particular industry, averaged over industries (with weights proportional to the size of the industry) and the resulting measure is normalized such that the United States scores a 1.0, and is due to Pryor (1973). Some of the scores on this variable are imputed using procedures described in Stephens and Wallerstein (1991, 945).

Source

Pryor, Frederic. 1973. *Property and Industrial Organization in Communist and Capitalist Countries*. Bloomington: Indiana University Press.

Stephens, John and Michael Wallerstein. 1991. Industrial Concentration, Country Size and Trade Union Membership. *American Political Science Review* 85:941-953.

Western, Bruce and Simon Jackman. 1994. Bayesian Inference for Comparative Research. *American Political Science Review* 88:412-423.

Wilensky, Harold L. 1981. Leftism, Catholicism, Democratic Corporatism: The Role of Political Parties in Recent Welfare State Development. In *The Development of Welfare States in Europe and America*, ed. Peter Flora and Arnold J. Heidenheimer. New Brunswick: Transaction Books.

References

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Hoboken, New Jersey.

Examples

```
data(unionDensity)
summary(unionDensity)
pairs(unionDensity,
      labels=c("Union\nDensity",
        "Left\nGovernment",
        "log Size of\nLabor Force",
        "Economic\nConcentration"),
      lower.panel=function(x,y,digits=2){
        r <- cor(x,y)
        par(usr=c(0,1,0,1))
        text(.5,.5,
             format(c(r,0.123456789),digits=digits)[1],
             cex=1.5)
      }
      )
ols <- lm(union ~ left + size + concen,
          data=unionDensity)
summary(ols)
```

`vectorRepresentation` *convert roll call matrix to series of vectors*

Description

Extract the information in a roll call matrix as a series of vectors with voting decision, a unique identifier for the legislator and a unique identifier for the roll call.

Usage

```
vectorRepresentation(object, dropList = list(codes = c("missing", "notInLegis")))
```

Arguments

object	an object of class rollcall
dropList	a dropList; see dropRollCall

Details

It is often the case that roll call matrices are sparse, say, when the roll call matrix has an "overlapping generations" structure; e.g., consider forming data by pooling across a long temporal sequence of legislatures such that relatively few of the legislators in the data set actually vote on any given roll call. In such a case, representing the data as a roll call matrix is not particularly helpful nor efficient, either for data summaries or modeling.

Value

A matrix with z rows, where z is the number of non-missing entries in object$votes, with 'missingness' defined by the codes component of the dropList. The matrix has 3 columns:

vote	the voting decision, either a 1 if the corresponding element of the roll call matrix object$votes is in the yea component of object$codes, or a 0 if the corresponding element of the roll call matrix is in the nay component of object$codes. Non-missing entries of the roll call matrix are not stored.
i	the row of the roll call matrix object$votes that supplied the voting decision; i.e., a unique identifier for the legislator generating this vote
j	the column of the roll call matrix object$votes that supplied the vote; i.e., a unique identifier for the vote.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

See Also

rollcall

Examples

```
data(s109)
y <- vectorRepresentation(s109)
apply(y,2,table,exclude=NULL)
```

vote92 *Reports of voting in the 1992 U.S. Presidential election.*

Description

Survey data containing self-reports of vote choice in the 1992 U.S. Presidential election, with numerous covariates, from the 1992 American National Election Studies.

Usage

```
data(vote92)
```

Format

A data frame with 909 observations on the following 10 variables.

`vote` a factor with levels `Perot` `Clinton` `Bush`

`dem` a numeric vector, 1 if the respondent reports identifying with the Democratic party, 0 otherwise.

`rep` a numeric vector, 1 if the respondent reports identifying with the Republican party, 0 otherwise

`female` a numeric vector, 1 if the respondent is female, 0 otherwise

`persfinance` a numeric vector, -1 if the respondent reports that their personal financial situation has gotten worse over the last 12 months, 0 for no change, 1 if better

`natlecon` a numeric vector, -1 if the respondent reports that national economic conditions have gotten worse over the last 12 months, 0 for no change, 1 if better

`clintondis` a numeric vector, squared difference between respondent's self-placement on a scale measure of political ideology and the respondent's placement of the Democratic candidate, Bill Clinton

`bushdis` a numeric vector, squared ideological distance of the respondent from the Republican candidate, President George H.W. Bush

`perotdis` a numeric vector, squared ideological distance of the respondent from the Reform Party candidate, Ross Perot

Details

These data are unweighted. Refer to the original data source for weights that purport to correct for non-representativeness and non-response.

Source

Alvarez, R. Michael and Jonathan Nagler. 1995. Economics, issues and the Perot candidacy: Voter choice in the 1992 Presidential election. *American Journal of Political Science*. 39:714-44.

Miller, Warren E., Donald R. Kinder, Steven J. Rosenstone and the National Election Studies. 1999. *National Election Studies, 1992: Pre-/Post-Election Study*. Center for Political Studies, University of Michigan: Ann Arbor, Michigan.

Inter-University Consortium for Political and Social Research. Study Number 1112. http://dx.doi.org/10.3886/ICPSR01112.

References

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Wiley: Hoboken, New Jersey. Examples 8.7 and 8.8.

Examples

```
data(vote92)
summary(vote92)
```

---

vuong *Vuong's non-nested hypothesis test*

Description

Compares two models fit to the same data that do not nest via Vuong's non-nested test.

Usage

```
vuong(m1, m2, digits = getOption("digits"))
```

Arguments

m1	model 1, an object inheriting from class glm, negbin or zeroinfl
m2	model 2, as for model 1
digits	significant digits in printed result

Details

The Vuong non-nested test is based on a comparison of the predicted probabilities of two models that do not nest. Examples include comparisons of zero-inflated count models with their non-zero-inflated analogs (e.g., zero-inflated Poisson versus ordinary Poisson, or zero-inflated negative-binomial versus ordinary negative-binomial). A large, positive test statistic provides evidence of the superiority of model 1 over model 2, while a large, negative test statistic is evidence of the superiority of model 2 over model 1. Under the null that the models are indistinguishable, the test statistic is asymptotically distributed standard normal.

Let $p_i = \hat{Pr}(y_i|M_1)$ be the predicted probabilities from model 1, evaluated conditional on the estimated MLEs. Let q_i be the corresponding probabilities from model 2. Then the Vuong statistic is $\sqrt{N}\bar{m}/s_m$ where $m_i = log(p_i) - log(q_i)$ and s_m is the sample standard deviation of m_i.

Two finite sample corrections are often considered, based on the Akaike (AIC) and Schwarz (BIC) penalty terms, based on the complexity of the two models. These corrections sometimes generate conflicting conclusions.

The function will fail if the models do not contain identical values in their respective components named y (the value of the response being modeled).

Value

nothing returned, prints 3 test statistics and p values and exits silently.

Author(s)

Simon Jackman <simon.jackman@sydney.edu.au>

References

Vuong, Q.H. 1989. Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica*. 57:307-333.

Examples

```
## Not run:
data("bioChemists")
## compare Poisson GLM and ZIP
glm1 <- glm(art ~ ., data = bioChemists, family = poisson)
zip <- zeroinfl(art ~ . | ., data = bioChemists, EM = TRUE)
vuong(glm1, zip)


## compare negbin with zero-inflated negbin
nb1 <- MASS::glm.nb(art ~ ., data=bioChemists)
zinb <- zeroinfl(art ~ . | ., data = bioChemists, dist = "negbin", EM = TRUE)
vuong(nb1, zinb)

## End(Not run)
```

---

zeroinfl *Zero-inflated Count Data Regression*

Description

Fit zero-inflated regression models for count data via maximum likelihood.

Usage

```
zeroinfl(formula, data, subset, na.action, weights, offset,
  dist = c("poisson", "negbin", "geometric"),
  link = c("logit", "probit", "cloglog", "cauchit", "log"),
  control = zeroinfl.control(...),
  model = TRUE, y = TRUE, x = FALSE, ...)
```

Arguments

formula	symbolic description of the model, see details.
data, subset, na.action	arguments controlling formula processing via model.frame.
weights	optional numeric vector of weights.
offset	optional numeric vector with an a priori known component to be included in the linear predictor of the count model. See below for more information on offsets.

dist	character specification of count model family (a log link is always used).
link	character specification of link function in the binary zero-inflation model (a binomial family is always used).
control	a list of control arguments specified via zeroinfl.control.
model, y, x	logicals. If TRUE the corresponding components of the fit (model frame, response, model matrix) are returned.
...	arguments passed to zeroinfl.control in the default setup.

Details

Zero-inflated count models are two-component mixture models combining a point mass at zero with a proper count distribution. Thus, there are two sources of zeros: zeros may come from both the point mass and from the count component. Usually the count model is a Poisson or negative binomial regression (with log link). The geometric distribution is a special case of the negative binomial with size parameter equal to 1. For modeling the unobserved state (zero vs. count), a binary model is used that captures the probability of zero inflation. in the simplest case only with an intercept but potentially containing regressors. For this zero-inflation model, a binomial model with different links can be used, typically logit or probit.

The formula can be used to specify both components of the model: If a formula of type y ~ x1 + x2 is supplied, then the same regressors are employed in both components. This is equivalent to y ~ x1 + x2 | x1 + x2. Of course, a different set of regressors could be specified for the count and zero-inflation component, e.g., y ~ x1 + x2 | z1 + z2 + z3 giving the count data model y ~ x1 + x2 conditional on (|) the zero-inflation model y ~ z1 + z2 + z3. A simple inflation model where all zero counts have the same probability of belonging to the zero component can by specified by the formula y ~ x1 + x2 | 1.

Offsets can be specified in both components of the model pertaining to count and zero-inflation model: y ~ x1 + offset(x2) | z1 + z2 + offset(z3), where x2 is used as an offset (i.e., with coefficient fixed to 1) in the count component and z3 analogously in the zero-inflation component. By the rule stated above y ~ x1 + offset(x2) is expanded to y ~ x1 + offset(x2) | x1 + offset(x2). Instead of using the offset() wrapper within the formula, the offset argument can also be employed which sets an offset only for the count model. Thus, formula = y ~ x1 and offset = x2 is equivalent to formula = y ~ x1 + offset(x2) | x1.

All parameters are estimated by maximum likelihood using optim, with control options set in zeroinfl.control. Starting values can be supplied, estimated by the EM (expectation maximization) algorithm, or by glm.fit (the default). Standard errors are derived numerically using the Hessian matrix returned by optim. See zeroinfl.control for details.

The returned fitted model object is of class "zeroinfl" and is similar to fitted "glm" objects. For elements such as "coefficients" or "terms" a list is returned with elements for the zero and count component, respectively. For details see below.

A set of standard extractor functions for fitted model objects is available for objects of class "zeroinfl", including methods to the generic functions print, summary, coef, vcov, logLik, residuals, predict, fitted, terms, model.matrix. See predict.zeroinfl for more details on all methods.

Value

An object of class "zeroinfl", i.e., a list with components including

coefficients	a list with elements "count" and "zero" containing the coefficients from the respective models,
residuals	a vector of raw residuals (observed - fitted),
fitted.values	a vector of fitted means,
optim	a list with the output from the optim call for minimizing the negative log-likelihood,
control	the control arguments passed to the optim call,
start	the starting values for the parameters passed to the optim call,
weights	the case weights used,
offset	a list with elements "count" and "zero" containing the offset vectors (if any) from the respective models,
n	number of observations (with weights > 0),
df.null	residual degrees of freedom for the null model (= n - 2),
df.residual	residual degrees of freedom for fitted model,
terms	a list with elements "count", "zero" and "full" containing the terms objects for the respective models,
theta	estimate of the additional θ parameter of the negative binomial model (if a negative binomial regression is used),
SE.logtheta	standard error for $\log(\theta)$,
loglik	log-likelihood of the fitted model,
vcov	covariance matrix of all coefficients in the model (derived from the Hessian of the optim output),
dist	character string describing the count distribution used,
link	character string describing the link of the zero-inflation model,
linkinv	the inverse link function corresponding to link,
converged	logical indicating successful convergence of optim,
call	the original function call,
formula	the original formula,
levels	levels of the categorical regressors,
contrasts	a list with elements "count" and "zero" containing the contrasts corresponding to levels from the respective models,
model	the full model frame (if model = TRUE),
y	the response count vector (if y = TRUE),
x	a list with elements "count" and "zero" containing the model matrices from the respective models (if x = TRUE),

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

References

Cameron, A. Colin and Pravin K. Trevedi. 1998. *Regression Analysis of Count Data.* New York: Cambridge University Press.

Cameron, A. Colin and Pravin K. Trivedi. 2005. *Microeconometrics: Methods and Applications*. Cambridge: Cambridge University Press.

Lambert, Diane. 1992. "Zero-Inflated Poisson Regression, with an Application to Defects in Manufacturing." *Technometrics*. **34**(1):1-14

Zeileis, Achim, Christian Kleiber and Simon Jackman 2008. "Regression Models for Count Data in R." *Journal of Statistical Software*, **27**(8). URL http://www.jstatsoft.org/v27/i08/.

See Also

zeroinfl.control, glm, glm.fit, glm.nb, hurdle

Examples

```
## data
data("bioChemists", package = "pscl")

## without inflation
## ("art ~ ." is "art ~ fem + mar + kid5 + phd + ment")
fm_pois <- glm(art ~ ., data = bioChemists, family = poisson)
fm_qpois <- glm(art ~ ., data = bioChemists, family = quasipoisson)
fm_nb <- MASS::glm.nb(art ~ ., data = bioChemists)

## with simple inflation (no regressors for zero component)
fm_zip <- zeroinfl(art ~ . | 1, data = bioChemists)
fm_zinb <- zeroinfl(art ~ . | 1, data = bioChemists, dist = "negbin")

## inflation with regressors
## ("art ~ . | ." is "art ~ fem + mar + kid5 + phd + ment | fem + mar + kid5 + phd + ment")
fm_zip2 <- zeroinfl(art ~ . | ., data = bioChemists)
fm_zinb2 <- zeroinfl(art ~ . | ., data = bioChemists, dist = "negbin")
```

---

zeroinfl.control *Control Parameters for Zero-inflated Count Data Regression*

Description

Various parameters that control fitting of zero-inflated regression models using zeroinfl.

Usage

```
zeroinfl.control(method = "BFGS", maxit = 10000, trace = FALSE,
  EM = FALSE, start = NULL, ...)
```

Arguments

method	characters string specifying the method argument passed to optim.
maxit	integer specifying the maxit argument (maximal number of iterations) passed to optim.
trace	logical or integer controlling whether tracing information on the progress of the optimization should be produced (passed to optim).
EM	logical. Should starting values be estimated by the EM (expectation maximization) algorithm? See details.
start	an optional list with elements "count" and "zero" (and potentially "theta") containing the coefficients for the corresponding component.
...	arguments passed to optim.

Details

All parameters in zeroinfl are estimated by maximum likelihood using optim with control options set in zeroinfl.control. Most arguments are passed on directly to optim, only trace is also used within zeroinfl and EM/start control the choice of starting values for calling optim.

Starting values can be supplied, estimated by the EM (expectation maximization) algorithm, or by glm.fit (the default). Standard errors are derived numerically using the Hessian matrix returned by optim. To supply starting values, start should be a list with elements "count" and "zero" and potentially "theta" (for negative binomial components only) containing the starting values for the coefficients of the corresponding component of the model.

Value

A list with the arguments specified.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

See Also

zeroinfl

Examples

```
## Not run:
data("bioChemists", package = "pscl")

## default start values
fm1 <- zeroinfl(art ~ ., data = bioChemists)

## use EM algorithm for start values
fm2 <- zeroinfl(art ~ ., data = bioChemists, EM = TRUE)

## user-supplied start values
fm3 <- zeroinfl(art ~ ., data = bioChemists,
```

```
  start = list(count = c(0.7, -0.2, 0.1, -0.2, 0, 0), zero = -1.7))

## End(Not run)
```

Index