Package ‘hmmr’

October 13, 2022

Version 1.0-0

Date 2021-05-26

Title ``Mixture and Hidden Markov Models with R'' Datasets and Example Code

Depends R (>= 4.0.0), depmixS4, stats4

Suggests

Description
Datasets and code examples that accompany our book Visser & Speekenbrink (2021), ``Mixture and Hidden Markov Models with R'', <https://depmix.github.io/hmmr/>.

License GPL (>= 2)

URL <https://depmix.github.io/hmmr/>

NeedsCompilation no

Author Ingmar Visser [aut, cre],
Maarten Speekenbrink [aut]

Maintainer Ingmar Visser <i.visser@uva.nl>

Repository CRAN

Date/Publication 2021-05-27 07:10:05 UTC

R topics documented:

`balance8` *Repeated measures on the balance scale*

Description

Data are from 8 repeated measurements of 25 balance scale items with 1004 participants ranging from 6-17 years of age. The data are aggregated over 5 item types: weight (w), distance (d), conflict weight (cw), conflict distance (cd), and conflict balance (cb).

Usage

```
data(balance8)
```

Format

A data frame with 8032 observations on the following 23 variables.

`id` the participant id number.

`age` the participant age in years.

`sex` a factor with levels `m` `f`

`group` the participant school group number; groups 3-8 are in primary school; corollary10 and 11 in secondary school.

`school` a factor with levels `primary` `secondary`

`time` the number of the repeated measurement.

`wc` the number of correct weight items.

`wi` the number of incorrect weight items.

`ws` the total number of weight items.

`dc` the number of correct distance items.

`di` the number of incorrect distance items.

`ds` the total number of distance items.

`cwc` the number of correct conflict-weight items.

`cwi` the number of incorrect conflict-weight items.

`cws` the total number of conflict-weight items.

`cdc` the number of correct conflict-distance items.

`cdi` the number of incorrect conflict-distance items.

`cds` the total number of conflict-distance items.

`cbc` the number of correct conflict-balance items.

`cbi` the number of incorrect conflict-balance items.

`cbs` the total number of conflict-balance items.

`totalCor` the total number of correct items.

`totalTrials` the total number of items.

Source

Brenda Jansen, University of Amsterdam. Unpublished data.

Examples

```
data(balance8)
```

`balance8pars` *Parameter estimates of models for the balance8 data set*

Description

Parameter estimates of hidden Markov models with 3-8 states for the balance8 data.

Usage

```
data(balance8pars)
```

Format

A (named) list with parameter estumates for models with 3-8 states. Each element contains additional attributes such as "message" reflecting the corresponding slots of the depmix.fitted object from which the parameters were extracted (using getpars).

Details

The list is generated with the following code:

```
data(balance8)
multstart <- function(model, nr=10, initIters=10, verbose=TRUE) {
  llbest <- as.numeric(logLik(model))
  bestmodel <- model
  for(i in 1:nr) {
    fmod <- fit(model, emcontrol=em.control(maxit=initIters))
if(verbose) print(paste(i,": ", logLik(fmod)))
    if(logLik(fmod) > llbest) {
      llbest <- logLik(fmod)
      bestmodel <- fmod
    }
  }
  bestmodel <- fit(bestmodel, emcontrol=em.control(random.start=FALSE))
  return(bestmodel)
}

set.seed(12)

hm3id <- depmix(list(cbind(wc,wi)~1,cbind(dc,di)~1,cbind(cwc,cwi)~1,
  cbind(cdc,cdi)~1,cbind(cbc,cbi)~1),
data=balance8, family=list(binomial("identity"),binomial("identity"),
binomial("identity"),binomial("identity"),binomial("identity")),
ntimes=rep(8,1004), ns=3,
respst=rep(0.5,15))

fhm3id <- multstart(hm3id)

hm4id <- depmix(list(cbind(wc,wi)~1,cbind(dc,di)~1,cbind(cwc,cwi)~1,
  cbind(cdc,cdi)~1,cbind(cbc,cbi)~1),
data=balance8, family=list(binomial("identity"),binomial("identity"),
binomial("identity"),binomial("identity"),binomial("identity")),
ntimes=rep(8,1004), ns=4,
respst=rep(0.5,20))

fhm4id <- multstart(hm4id)


hm5id <- depmix(list(cbind(wc,wi)~1,cbind(dc,di)~1,cbind(cwc,cwi)~1,
  cbind(cdc,cdi)~1,cbind(cbc,cbi)~1),
data=balance8, family=list(binomial("identity"),binomial("identity"),
binomial("identity"),binomial("identity"),binomial("identity")),
ntimes=rep(8,1004), ns=5,
respst=rep(0.5,25))

fhm5id <- multstart(hm5id)

hm6id <- depmix(list(cbind(wc,wi)~1,cbind(dc,di)~1,cbind(cwc,cwi)~1,
  cbind(cdc,cdi)~1,cbind(cbc,cbi)~1),
data=balance8, family=list(binomial("identity"),binomial("identity"),
binomial("identity"),binomial("identity"),binomial("identity")),
ntimes=rep(8,1004), ns=6,
respst=rep(0.5,30))
```

```
fhm6id <- multstart(hm6id)

hm7id <- depmix(list(cbind(wc,wi)~1,cbind(dc,di)~1,cbind(cwc,cwi)~1,
  cbind(cdc,cdi)~1,cbind(cbc,cbi)~1),
data=balance8, family=list(binomial("identity"),binomial("identity"),
binomial("identity"),binomial("identity"),binomial("identity")),
ntimes=rep(8,1004), ns=7,
respst=rep(0.5,35))

fhm7id <- multstart(hm7id)

set.seed(1)

hm8id <- depmix(list(cbind(wc,wi)~1,cbind(dc,di)~1,cbind(cwc,cwi)~1,
  cbind(cdc,cdi)~1,cbind(cbc,cbi)~1),
data=balance8, family=list(binomial("identity"),binomial("identity"),
binomial("identity"),binomial("identity"),binomial("identity")),
ntimes=rep(8,1004), ns=8,
respst=rep(0.5,40))

fhm8id <- multstart(hm8id)

balance8models <- list(fhm3id,fhm4id,fhm5id,fhm6id,fhm7id,fhm8id)

balance8pars <- list()
for(i in 1:length(balance8models)) {
  balance8pars[[i]] <- getpars(balance8models[[i]])
  attr(balance8pars[[i]], "message") <- balance8models[[i]]@message
  attr(balance8pars[[i]], "conMat") <- balance8models[[i]]@conMat
  attr(balance8pars[[i]], "lin.upper") <- balance8models[[i]]@lin.upper
  attr(balance8pars[[i]], "lin.lower") <- balance8models[[i]]@lin.lower
}
names(balance8pars) <- c("fhm3id","fhm4id","fhm5id","fhm6id","fhm7id","fhm8id")
```

Examples

```
data(balance8)
data(balance8pars)
# reconstruct the list of fitted models from the parameters
balance8models <- list()
for(i in 1:length(balance8pars)) {
  # define model
  mod <- depmix(list(cbind(wc,wi)~1,cbind(dc,di)~1,cbind(cwc,cwi)~1,
    cbind(cdc,cdi)~1,cbind(cbc,cbi)~1),
    data=balance8, family=list(binomial("identity"),binomial("identity"),
    binomial("identity"),binomial("identity"),binomial("identity")),
    ntimes=rep(8,1004), ns=attr(balance8pars[[i]],"nstates"))
  # set the parameters to the estimated ones
  mod <- setpars(mod, balance8pars[[i]])
  # convert to a depmix.fitted object
  mod <- as(mod,"depmix.fitted")
  # set slots of depmix.fitted object
  mod@message <- attr(balance8pars[[i]],"message")
  mod@conMat <- attr(balance8pars[[i]],"conMat")
  mod@lin.upper <- attr(balance8pars[[i]],"lin.upper")
  mod@lin.lower <- attr(balance8pars[[i]],"lin.lower")
  mod@posterior <- viterbi(mod)
  # add to list of models
  balance8models[[i]] <- mod
}
names(balance8models) <- c("fhm3id","fhm4id","fhm5id","fhm6id","fhm7id","fhm8id")
```

confint — *Confidence intervals Visser et al (2000)*

Description

Values from Table 1 of Visser et al (2000) with confidence intervals obtained using four different methods; 1) likelihood profiles, 2) bootstrap with 500 samples, 3) bootstrap with 1000 samples, 4) finite differences approximation of the hessian.

Usage

```
data(confint)
```

Format

A data.frame with 9 variables for each parameter of a 2-state hidden Markov model fitted to data `simplehmm`:

par parameter name, see Visser et al (2000) for details.

true true value of the parameter.

mle maximum likelihood estimate of the parameter.

prof the size of the confidence interval resulting from the profile likelihood method.

left the left hand boundary of the (95%) confidence interval using the likelihood profile method.

right the right hand boundary of the (95%) confidence interval using the likelihood profile method.

b500 the size of the confidence interval resulting from 500 bootstrap samples.

b1000 the size of the confidence interval resulting from 1000 bootstrap samples.

fdh the size of the confidence interval based on the finite differences approximation of the hessian.

Source

Ingmar Visser, Maartje E. J. Raijmakers, and Peter C. M. Molenaar (2000). Confidence intervals for hidden Markov model parameters. *British journal of mathematical and statistical psychology*, 53, p. 317-327.

Examples

```
data(confint)
```

`conservation` *Conservation of liquid*

Description

This data set is from a conservation of liquid task adminstered to 101 children aged 6-10, measured at 5 occasions. The data provided here are those analyzed in *Schmittmann et al. 2005* in Illustration 1, which is part of a larger data set described *Van der Maas, 1993.*

Usage

```
data(discrimination)
```

Format

A data frame with 101 rows with 6 variables:

- `sex` a factor giving the sex of the participants.
- `r1` participant responses at measurement occascion 1.
- `r2` participant responses at measurement occascion 2.
- `r3` participant responses at measurement occascion 3.
- `r4` participant responses at measurement occascion 4.
- `r5` participant responses at measurement occascion 5.

Source

van der Maas, H. L. J.. (1993). *Catastrophe analysis of stagewise cognitive development: Model, method and applications.* Doctoral dissertation (Dissertatie reeks 1993-2), University of Amsterdam.

Schmittmann, V. D., Dolan, C. V., van der Maas, H. L., & Neale, M. C. (2005). Discrete latent Markov models for normally distributed response data. *Multivariate Behavioral Research*, 40(4), 461-488.

`dccs` *Dimensional Change Card Sort Task Data*

Description

Data from 3-5 year old children performing on the dimensional change card sort task published in Van Bers et al (2011).

Usage

```
data(dccs)
data(dccslong)
```

Format

A data.frame consisting of the following variable:

`pp`	participant number
`ageM`	(numeric) age in months
`ageY`	(numeric) age in years
`ageGr`	(numeric) age group
`sex`	(numeric) age group
`nTrPre`	(numeric) number of trials completed pre-switch
`nTrPre`	(numeric) number of trials correct in pre-switch trials
`t1Post`	(numeric) indicator variable for post switch trial indicating correct (1) or incorrect (0)
`t2Post`	(numeric) indicator variable for post switch trial indicating correct (1) or incorrect (0)
`t3Post`	(numeric) indicator variable for post switch trial indicating correct (1) or incorrect (0)
`t4Post`	(numeric) indicator variable for post switch trial indicating correct (1) or incorrect (0)
`t5Post`	(numeric) indicator variable for post switch trial indicating correct (1) or incorrect (0)
`t6Post`	(numeric) indicator variable for post switch trial indicating correct (1) or incorrect (0)
`nCorPost`	(numeric) number of trials correct post-switch
`passPost`	(numeric) indicator variable whether post-switch test was passed

Source

Bianca M.C.W. van Bers, Ingmar Visser, Tessa J.P. van Schijndel, Dorothy J. Mandell and Maartje E.J. Raijmakers (2011). The dynamics of development on the Dimensional Change Card Sorting task. *Developmental Science, vol 14(5)*, 960-971.

Examples

```
# the original data was in wide format; for analysis with hidden Markov models,
# data in long format is more convenient; the following code was used to reshape
# the data from wide to long format (and make a selection of the columns)

data(dccs)
dccslong <- reshape(
dccs[,c("pp","ageM","sex","t1Post","t2Post","t3Post","t4Post","t5Post","t6Post")],
direction="long",
varying=list(4:9) )
dccslong <- dccslong[order(dccslong$pp),-6]
names(dccslong) <- c("pp","ageM","sex","trial","acc")
```

`dccs_boot_LR` *dccs boot LR*

Description

Example of a parametric bootstrap for model selection with the dccs data

Usage

```
data("dccs_boot_LR")
```

Format

A `boot` object

Details

The bootstrap sample was generated by the following code:

```
require(depmixS4)
      require(hmmr)
      require(boot)

      data(dccs)
      m2 <- mix(response=cbind(nCorPost,6-nCorPost)~1,nstates=2,
                data=dccs,family=binomial())
      set.seed(1234)
      fm2 <- fit(m2)
      set.seed(1234)
      fm3 <- fit(mix(response=cbind(nCorPost,6-nCorPost)~1,
                     nstates=3, data=dccs,family=binomial()))

      boot.fun <- function(model) {
        ok <- FALSE
        while(!ok) {
          bootdat <- data.frame(
            simulate(model)@response[[1]][[1]]@y)
          fboot2 <- try({
            mod <- mix(cbind(X1,X2)~1, nstates=2,
                       family=binomial(), data=bootdat)
            mod <- setpars(mod,getpars(fm2))
            fit(mod,emcontrol=em.control(random.start=FALSE),
                verbose=FALSE)
          })
          fboot3 <- try({
            mod <- mix(cbind(X1,X2)~1, nstates=3,
                       family=binomial(), data=bootdat)
            mod <- setpars(mod,getpars(fm3))
            fit(mod,emcontrol=em.control(random.start=FALSE),
                verbose=FALSE)
          })
          if(!inherits(fboot2,"try-error") &&
             !inherits(fboot3,"try-error") &&
             logLik(fboot3) >= logLik(fboot2)) ok <- TRUE
          # if(ok) if(logLik(fboot3) < logLik(fboot2)) ok <- FALSE
        }
        llratio(fboot3,fboot2)@value
      }
      set.seed(1234)
      dccs_boot_LR <- boot(fm2, boot.fun, R=10000, sim="parametric")
```

Examples

```
data(dccs_boot_LR)
```

---

disc42 *Discrimination Learning Data*

Description

This data set is a selection of six participants' responses from the `discrimination` data set.

Usage

```
data(disc42)
```

Format

A data frame consisting of the following variable:

`acc` a factor of accuracy scores (incorrect/correct)

Source

Maartje E. J. Raijmakers, Conor V. Dolan and Peter C. M. Molenaar (2001). Finite mixture distribution models of simple discrimination learning. *Memory \& Cognition*, vol 29(5).

Ingmar Visser, Verena D. Schmittmann, and Maartje E. J. Raijmakers (2007). Markov process models for discrimination learning. In: Kees van Montfort, Han Oud, and Albert Satorra (Eds.), *Longitudinal models in the behavioral and related sciences*, Mahwah (NJ): Lawrence Erlbaum Associates.

Verena D. Schmittmann, Ingmar Visser and Maartje E. J. Raijmakers (2006). Multiple learning modes in the development of rule-based category-learning task performance. *Neuropsychologia, vol 44(11)*, p. 2079-2091.

`discrimination` *Discrimination Learning Data*

Description

This data set is from a simple discrimation learning experiment. It consists of 192 binary series of responses of different lengths. This is a subset of the data described by *Raijmakers et al. (2001)*, and it is analyzed much more extensively using latent Markov models and depmix in *Schmittmann et al. (2006)* and *Visser et al. (2006)*..

Usage

```
data(discrimination)
```

Format

A data frame with a total of 3139 observations on the following variable:

`acc` a factor of accuracy scores (incorrect/correct)

Source

Maartje E. J. Raijmakers, Conor V. Dolan and Peter C. M. Molenaar (2001). Finite mixture distribution models of simple discrimination learning. *Memory \& Cognition*, vol 29(5).

Ingmar Visser, Verena D. Schmittmann, and Maartje E. J. Raijmakers (2007). Markov process models for discrimination learning. In: Kees van Montfort, Han Oud, and Albert Satorra (Eds.), *Longitudinal models in the behavioral and related sciences*, Mahwah (NJ): Lawrence Erlbaum Associates.

Verena D. Schmittmann, Ingmar Visser and Maartje E. J. Raijmakers (2006). Multiple learning modes in the development of rule-based category-learning task performance. *Neuropsychologia, vol 44(11)*, p. 2079-2091.

`FFBS_BinomialNormal` *Posterior (MCMC) samples for a hidden Markov model with a Binomial and Normal response using the forward-filtering backward-sampling algorithm.*

Description

Posterior (MCMC) samples for a hidden Markov model with a Binomial and Normal response using the forward-filtering backward-sampling algorithm.

Usage

```
FFBS_BinomialNormal(bin,norm,nstates,hyperPars=list(),ntimes,niter=1000,nburnin=0)
```

Arguments

`bin`	the Binomial response variable. As in the glm function, this can be a binary vector with 1 indicating success and 0 failure, a factor where the first level is considered a failure and all other leves success, or a matrix with the number of successes and failures in the columns.
`norm`	the Normal response variable. This should be a numeric vector.
`nstates`	the required number of states in the hidden Markov model.
`hyperPars`	a named `list` with values of the hyper-parameters. See details.
`ntimes`	the lengths of time series in arguments `bin` and `norm`; it defaults to assuming a single time series of length `length(bin)`.
`niter`	number of iterations to run the sampler for.
`nburnin`	number of initial samples to discard as burnin,

Details

This function runs the forward-filtering backwards-sampling MCMC algorithm for a hidden Markov model with a Binomial and Normal response variable. The response variables are assumed conditionally independent given the states.

The following conjugate prior distributions are used:

For the initial state probabilities, a Dirichlet prior with parameter vector `init_alpha`

For each row in the transition probability matrix, a Dirichlet prior is used. The parameters of these Dirichlet distributions are contained in the matrix `trans_alpha`.

For the probability of correct in the Binomial response, a Beta prior is used, with parameters `bin_alpha` and `bin_beta`.

For the mean and variance of the Normal response, a Normal-inverse-Gamma prior is used.

This function was written mainly for didactive purposes, not for speed (or compatibility with other packages which provide posterior samples).

Value

A named list with samples of the different parameters.

Author(s)

Maarten Speekenbrink

References

Visser, I., & Speekenbrink, M. (in preparation). Mixture and hidden Markov models in R.

Examples

```
## Not run:
  data(speed)
  set.seed(1)
  hyperPars <- list(norm_invsigma_scale=.01,norm_invsigma_shape=.01,norm_mu_sca=.1)
  mcmc_samples <- FFBS_BinomialNormal(speed$corr,speed$rt,nstates=2,
        ntimes=c(168,134,137),niter=500,hyperPars = hyperPars)

  plot(mcmc_samples$mu[,1])
  hist(mcmc_samples$mu[,1])

## End(Not run)
```

hmm *Fit hidden Markov and latent class models.*

Description

hmm fits a hidden Markov model to its first argument. lca fits a latent class model or mixture model to its first argument.

Both functions provide an easy user-interface to the functions provided in **depmixS4** by automagically setting some argument values.

Usage

```
hmm(data, nstates, fit = TRUE, ntimes = NULL, family = NULL, verbose=FALSE, ...)
lca(data, nclasses, fit = TRUE, family = NULL, verbose=FALSE, ...)
```

Arguments

data	(columns of) a data.frame or matrix like object.
nstates	the required number of states of the hidden Markov model.
nclasses	the required number of classes of the mixture or latent class model.
fit	logical indicating whether the model needs to be fitted or returned unfitted; the latter is necessary if one wants to set constraints on the parameters, which then requires using the fit function from **depmixS4**.
ntimes	the lengths of time series in argument data; it defaults to assuming a single time series of length nrow(data).
family	(a list of) name(s) of the distribution(s) to be used in fitting; if provided, it should have length of the number of the number of columns in data, see Details.
verbose	logical; when TRUE iteration information of the fitting process is printed.
...	not currently used.

Details

The distributions used in fitting models are the `multinomial` for `factor` data columns and `gaussian` for `numeric` data columns. Data columns are treated as conditionally independent variables. Use `makeDepmix` in the **depmixS4** package to specify multivariate distributions.

Value

`hmm` returns a `depmix` or `depmix.fitted` object depending on the value of the `fit` argument; `lca` similarly returns either a `mix` or `mix.fitted` object.

All these can be `print`'ed and `summary`'zed.

Author(s)

Ingmar Visser

References

Visser, I., & Speekenbrink, M. (2010). depmixS4: an R-package for hidden Markov models. Journal of Statistical Software, 36(7), 1-21.

Examples

```
data(conservation)

set.seed(1)
m2 <- lca(conservation$"r1", nclasses=2)
m2
summary(m2)

data(speed1)

set.seed(1)
hm2 <- hmm(speed1$"RT", nstates=2)
hm2
summary(hm2)
```

`IGT` *Iowa Gambling Task data*

Description

This data set contains responses of 30 participants on the Iowa Gambling Task.

Usage

```
data(IGT)
```

Format

A data frame with 3000 observations on the following variables.

`id` a factor with participant IDs

`trial` a numeric vector with trial numbers

`deck` a factor with the deck chosen on each trial (A-D)

`wager` a factor indicating whether participants wagered low or high

`win` a numeric variable with the amount won on each trial

`loss` a numeric variable with the amount lost on each trial

`gdeck` a factor indicating whether the chosen deck was good/advantageous (TRUE) or not (FALSE)

`fdeck` a factor indicating whether the chosen deck has a relatively high frequency of losses (TRUE) or not (FALSE)

Source

Konstantinidis, E. and Shanks, D.R. (2014). Don't Bet on it! Wagering as a Measure of Awareness in Decision Making under Uncertainty. *Journal of Experimental Psychology: General*, 143(6), 2111-2134.

Examples

```
data(IGT)
```

`MAR_simulation_results` *Missing at random (MAR) simulation results.*

Description

Results of a simulation study on the effect of missing data on estimates of parameters of a hidden Markov model. In the simulation, 1000 datasets were simulated, each consisting of 100 time series of length 50. The model generating the data was a hidden Markov model with 3 states. Missing values were generated for 25% of cases, independent of the hidden state (i.e. data can be considered missing at random.)

Usage

```
data("MAR_simulation_results")
```

Format

A `data.frame` with the following variables:

`parameter`	Parameter name
`true`	True value of parameter
`MAR_estimates_mean`	Average of parameter estimates for a model that assumes MAR.
`MAR_estimates_sd`	Standard deviation of parameter estimates for a model that assumes MAR.
`MAR_estimates_MAE`	Mean Absolute Error of parameter estimates for a model that assumes MAR.
`MNAR_estimates_mean`	Average of parameter estimates for a model that assumes MNAR.
`MNAR_estimates_sd`	Standard deviation of parameter estimates for a model that assumes MNAR.
`MNAR_estimates_MAE`	Mean Absolute Error of parameter estimates for a model that assumes MNAR.
`percMAE`	Relative MAE of the MNAR model over the MAR model (e.g. MAE_MNAR/MAE_MAR)

Details

This data frame was generated with the following code

```
library(depmixS4)

### Start simulation

nsim <- 1000
nrep <- 100
nt <- 50

set.seed(1234)
randomSeeds <- sample(seq(1,nsim*1000),nsim)
out <- rep(list(vector("list",3)),nsim)

prior <- c(8,1,1)
prior <- prior/sum(prior)
transition <- 5*diag(3) + 1
transition <- transition/rowSums(transition)
means <- c(-1,0,1)
sds <- c(3,3,3)
pmiss <- c(.25,.25,.25)

truepars1 <- c(prior,as.numeric(t(transition)),as.numeric(rbind(means,sds)))
truepars2 <- c(prior,as.numeric(t(transition)),as.numeric(rbind(means,sds,1-pmiss,pmiss)))

for(sim in 1:nsim) {
  set.seed(randomSeeds[sim])
  truestate <- matrix(nrow=nt,ncol=nrep)
  for(i in 1:nrep) {
    truestate[1,i] <- sample(1:3,size=1,prob=prior)
    for(t in 2:nt) {
      truestate[t,i] <- sample(1:3,size=1,prob=transition[truestate[t-1,i],])
    }
  }
  dat <- data.frame(trueState=as.numeric(truestate),trial=1:nt)
 dat$trueResponse <- rnorm(nrow(dat),mean=means[dat$trueState],sd=sds[dat$trueState])
  dat$missing <- rbinom(nrow(dat),size=1,prob=pmiss[dat$trueState])
  dat$response <- dat$trueResponse
  dat$response[dat$missing==1] <- NA

  set.seed(randomSeeds[sim])
 mod <- depmix(list(response~1),family=list(gaussian()),data=dat,nstates=3,ntimes=rep(nt,nrep))
  mod <- setpars(mod,truepars1)
  ok <- FALSE
  ntry <- 1
  while(!ok & ntry <= 50) {
   fmod <- try(fit(mod,emcontrol=em.control(maxit = 5000, random.start=FALSE),verbose=FALSE))
    if(!inherits(fmod,"try-error")) {
    if(fmod@message == "Log likelihood converged to within tol. (relative change)") ok <- TRUE
    }
    ntry <- ntry + 1
  }
 out[[sim]][[1]] <- list(pars=getpars(fmod),logLik=logLik(fmod),viterbi=posterior(fmod)[,1],trueStat

  set.seed(randomSeeds[sim])
 mod <- depmix(list(response~1,missing~1),family=list(gaussian(),multinomial("identity")),data=dat,n
  mod <- setpars(mod,truepars2)
  ok <- FALSE
  ntry <- 1
  while(!ok & ntry <= 50) {
   fmod <- try(fit(mod,emcontrol=em.control(maxit = 5000, random.start = FALSE),verbose=FALSE))
    if(!inherits(fmod,"try-error")) {
    if(fmod@message == "Log likelihood converged to within tol. (relative change)") ok <- TRUE
    }
    ntry <- ntry + 1
  }
 out[[sim]][[2]] <- list(pars=getpars(fmod),logLik=logLik(fmod),viterbi=posterior(fmod)[,1],trueStat
}

### End simulation

### Process results
library(reshape2)
library(dplyr)
library(tidyr)

simi <- out

bias1 <- matrix(0.0,ncol=length(truepars1),nrow=nsim)
```

```
bias2 <- matrix(0.0,ncol=length(truepars2),nrow=nsim)

colnames(bias1) <- names(simi[[1]][[1]][[1]])
colnames(bias2) <- names(simi[[1]][[2]][[1]])

pcorstate1 <- rep(0.0,nsim)
pcorstate2 <- rep(0.0,nsim)

for(sim in 1:nsim) {
  tmp <- simi[[sim]][[1]][[1]]
  pr <- tmp[1:3]
  trt <- matrix(tmp[4:12],ncol=3)
  ms <- tmp[c(13,15,17)]
  sds <- tmp[c(14,16,18)]
  ord <- order(ms)
 bias1[sim,] <- c(pr[ord],trt[ord,ord],as.numeric(rbind(ms[ord],sds[ord]))) - truepars1
 fsta <- recode(simi[[sim]][[1]]$viterbi,`1` = which(ord == 1), `2` = which(ord == 2), `3` = which(ord =
  pcorstate1[sim] <- sum(fsta == simi[[sim]][[1]]$trueState)/(nrep*nt)

  tmp <- simi[[sim]][[2]][[1]]
  pr <- tmp[1:3]
  trt <- matrix(tmp[4:12],ncol=3)
  ms <- tmp[c(13,17,21)]
  sds <- tmp[c(14,18,22)]
  ps0 <- tmp[c(15,19,23)]
  ps1 <- tmp[c(16,20,24)]
  ord <- order(ms)
 bias2[sim,] <- c(pr[ord],trt[ord,ord],as.numeric(rbind(ms[ord],sds[ord],ps0[ord],ps1[ord]))) - true
 fsta <- recode(simi[[sim]][[2]]$viterbi,`1` = which(ord == 1), `2` = which(ord == 2), `3` = which(ord =
  pcorstate2[sim] <- sum(fsta == simi[[sim]][[2]]$trueState)/(nrep*nt)
}

sim3_bias1 <- bias1
sim3_bias2 <- bias2
sim3_est1 <- t(t(bias1) + truepars1)
sim3_est2 <- t(t(bias2) + truepars2)
sim3_pcorstate1 <- pcorstate1
sim3_pcorstate2 <- pcorstate2

tmp <- as.data.frame(sim3_est1)
colnames(tmp) <- c("$\pi_1$","$\pi_2$","$\pi_3$","$a_{11}$","$a_{12}$","$a_{13}$","$a_{21}$","$a_{22
tmp$sim <- 1:nsim
tmp <- gather(tmp,key="param",value="estimate",-sim)
tmp$true <- rep(truepars1,each=nsim)
tmp$method <- "MAR"
tmp$variance <- "low"
tmp$missing <- "equal"
sim4_est1_long <- tmp
```

```
tmp <- as.data.frame(sim3_est2)
colnames(tmp) <- c("$\pi_1$","$\pi_2$","$\pi_3$","$a_{11}$","$a_{12}$","$a_{13}$","$a_{21}$","$a_{22
tmp$sim <- 1:nsim
tmp <- gather(tmp,key="param",value="estimate",-sim)
tmp$true <- rep(truepars2,each=nsim)
tmp$method <- "MNAR"
tmp$variance <- "low"
tmp$missing <- "equal"
sim4_est2_long <- tmp

all_long <- rbind(
  sim4_est1_long,
  subset(sim4_est2_long,!(param
)

my_table <- all_long
  group_by(param, method, variance, missing)
  summarize(true = mean(true),
            mean = mean(estimate),
            sd = sd(estimate),
            bias = mean(abs(true - estimate)/abs(true)),
            MAE = mean(abs(true - estimate)))
  gather(measure,value,-c(param,method,variance,missing))
  recast(param + variance + missing  ~  method + measure)

MAR_simulation_results <- my_table
  filter(variance == "low" & missing == "equal")
  dplyr::select(param,MAR_true,paste0("MAR_",c("mean","sd","MAE"),sep=""),
                paste0("MNAR_",c("mean","sd","MAE"),sep=""))
  mutate(percMAE = MNAR_MAE/MAR_MAE)

colnames(MAR_simulation_results) <- c("parameter", "true", "MAR_estimates_mean",
  "MAR_estimates_sd", "MAR_estimates_MAE", "MNAR_estimates_mean",
  "MNAR_estimates_sd", "MNAR_estimates_MAE", "percMAE")
MAR_simulation_results <- MAR_simulation_results[c(1:3,7:9,4:6,10:21),]
```

Examples

```
data(MAR_simulation_results)
```

`MNAR_simulation_results`

Missing not at random (MNAR) simulation results.

Description

Results of a simulation study on the effect of missing data on estimates of parameters of a hidden Markov model. In the simulation, 1000 datasets were simulated, each consisting of 100 time series of length 50. The model generating the data was a hidden Markov model with 3 states. Missing values were generated for 5% of cases in state 1, 25% of cases in state 2, and 50% of cases in state 3 (i.e. data can be considered missing not at random, as missingness is state-dependent).

Usage

```
data("MNAR_simulation_results")
```

Format

A `data.frame` with the following variables:

`parameter`	Parameter name
`true`	True value of parameter
`MAR_estimates_mean`	Average of parameter estimates for a model that assumes MAR.
`MAR_estimates_sd`	Standard deviation of parameter estimates for a model that assumes MAR.
`MAR_estimates_MAE`	Mean Absolute Error of parameter estimates for a model that assumes MAR.
`MNAR_estimates_mean`	Average of parameter estimates for a model that assumes MNAR.
`MNAR_estimates_sd`	Standard deviation of parameter estimates for a model that assumes MNAR.
`MNAR_estimates_MAE`	Mean Absolute Error of parameter estimates for a model that assumes MNAR.
`percMAE`	Relative MAE of the MNAR model over the MAR model (e.g. MAE_MNAR/MAE_MAR)

Details

This data frame was generated with the following code

```
library(depmixS4)

### Start simulation

nsim <- 1000
nrep <- 100
nt <- 50

set.seed(1234)
randomSeeds <- sample(seq(1,nsim*1000),nsim)
out <- rep(list(vector("list",3)),nsim)

prior <- c(8,1,1)
prior <- prior/sum(prior)
transition <- 5*diag(3) + 1
transition <- transition/rowSums(transition)
```

```
means <- c(-1,0,1)
sds <- c(3,3,3)
pmiss <- c(.05,.25,.5)

truepars1 <- c(prior,as.numeric(t(transition)),as.numeric(rbind(means,sds)))
truepars2 <- c(prior,as.numeric(t(transition)),as.numeric(rbind(means,sds,1-pmiss,pmiss)))

for(sim in 1:nsim) {
  set.seed(randomSeeds[sim])
  truestate <- matrix(nrow=nt,ncol=nrep)
  for(i in 1:nrep) {
    truestate[1,i] <- sample(1:3,size=1,prob=prior)
    for(t in 2:nt) {
      truestate[t,i] <- sample(1:3,size=1,prob=transition[truestate[t-1,i],])
    }
  }
  dat <- data.frame(trueState=as.numeric(truestate),trial=1:nt)
 dat$trueResponse <- rnorm(nrow(dat),mean=means[dat$trueState],sd=sds[dat$trueState])
  dat$missing <- rbinom(nrow(dat),size=1,prob=pmiss[dat$trueState])
  dat$response <- dat$trueResponse
  dat$response[dat$missing==1] <- NA

  set.seed(randomSeeds[sim])
 mod <- depmix(list(response~1),family=list(gaussian()),data=dat,nstates=3,ntimes=rep(nt,nrep))
  mod <- setpars(mod,truepars1)
  ok <- FALSE
  ntry <- 1
  while(!ok & ntry <= 50) {
   fmod <- try(fit(mod,emcontrol=em.control(maxit = 5000, random.start=FALSE),verbose=FALSE))
    if(!inherits(fmod,"try-error")) {
    if(fmod@message == "Log likelihood converged to within tol. (relative change)") ok <- TRUE
    }
    ntry <- ntry + 1
  }
 out[[sim]][[1]] <- list(pars=getpars(fmod),logLik=logLik(fmod),viterbi=posterior(fmod)[,1],trueStat

  set.seed(randomSeeds[sim])
 mod <- depmix(list(response~1,missing~1),family=list(gaussian(),multinomial("identity")),data=dat,n
  mod <- setpars(mod,truepars2)
  ok <- FALSE
  ntry <- 1
  while(!ok & ntry <= 50) {
   fmod <- try(fit(mod,emcontrol=em.control(maxit = 5000, random.start = FALSE),verbose=FALSE))
    if(!inherits(fmod,"try-error")) {
    if(fmod@message == "Log likelihood converged to within tol. (relative change)") ok <- TRUE
    }
    ntry <- ntry + 1
  }
  out[[sim]][[2]] <- list(pars=getpars(fmod),logLik=logLik(fmod),viterbi=posterior(fmod)[,1],trueStat
}

### End simulation

### Process results
library(reshape2)
library(dplyr)
library(tidyr)

simi <- out

bias1 <- matrix(0.0,ncol=length(truepars1),nrow=nsim)
bias2 <- matrix(0.0,ncol=length(truepars2),nrow=nsim)

colnames(bias1) <- names(simi[[1]][[1]][[1]])
colnames(bias2) <- names(simi[[1]][[2]][[1]])

pcorstate1 <- rep(0.0,nsim)
pcorstate2 <- rep(0.0,nsim)

for(sim in 1:nsim) {
  tmp <- simi[[sim]][[1]][[1]]
  pr <- tmp[1:3]
  trt <- matrix(tmp[4:12],ncol=3)
  ms <- tmp[c(13,15,17)]
  sds <- tmp[c(14,16,18)]
  ord <- order(ms)
  bias1[sim,] <- c(pr[ord],trt[ord,ord],as.numeric(rbind(ms[ord],sds[ord]))) - truepars1
  fsta <- recode(simi[[sim]][[1]]$viterbi,`1` = which(ord == 1), `2` = which(ord == 2), `3` = which(ord =
  pcorstate1[sim] <- sum(fsta == simi[[sim]][[1]]$trueState)/(nrep*nt)

  tmp <- simi[[sim]][[2]][[1]]
  pr <- tmp[1:3]
  trt <- matrix(tmp[4:12],ncol=3)
  ms <- tmp[c(13,17,21)]
  sds <- tmp[c(14,18,22)]
  ps0 <- tmp[c(15,19,23)]
  ps1 <- tmp[c(16,20,24)]
  ord <- order(ms)
  bias2[sim,] <- c(pr[ord],trt[ord,ord],as.numeric(rbind(ms[ord],sds[ord],ps0[ord],ps1[ord]))) - true
  fsta <- recode(simi[[sim]][[2]]$viterbi,`1` = which(ord == 1), `2` = which(ord == 2), `3` = which(ord =
  pcorstate2[sim] <- sum(fsta == simi[[sim]][[2]]$trueState)/(nrep*nt)
}

sim2_bias1 <- bias1
sim2_bias2 <- bias2
sim2_est1 <- t(t(bias1) + truepars1)
```

```
sim2_est2 <- t(t(bias2) + truepars2)
sim2_pcorstate1 <- pcorstate1
sim2_pcorstate2 <- pcorstate2

tmp <- as.data.frame(sim2_est1)
colnames(tmp) <- c("$\pi_1$","$\pi_2$","$\pi_3$","$a_{11}$","$a_{12}$","$a_{13}$","$a_{21}$","$a_{22
tmp$sim <- 1:nsim
tmp <- gather(tmp,key="param",value="estimate",-sim)
tmp$true <- rep(truepars1,each=nsim)
tmp$method <- "MAR"
tmp$variance <- "low"
tmp$missing <- "equal"
sim2_est1_long <- tmp

tmp <- as.data.frame(sim2_est2)
colnames(tmp) <- c("$\pi_1$","$\pi_2$","$\pi_3$","$a_{11}$","$a_{12}$","$a_{13}$","$a_{21}$","$a_{22
tmp$sim <- 1:nsim
tmp <- gather(tmp,key="param",value="estimate",-sim)
tmp$true <- rep(truepars2,each=nsim)
tmp$method <- "MNAR"
tmp$variance <- "low"
tmp$missing <- "equal"
sim2_est2_long <- tmp

all_long <- rbind(
  sim2_est1_long,
  subset(sim2_est2_long,!(param
)

my_table <- all_long
  group_by(param, method, variance, missing)
  summarize(true = mean(true),
            mean = mean(estimate),
            sd = sd(estimate),
            bias = mean(abs(true - estimate)/abs(true)),
            MAE = mean(abs(true - estimate)))
  gather(measure,value,-c(param,method,variance,missing))
  recast(param + variance + missing  ~  method + measure)

MAR_simulation_results <- my_table
  filter(variance == "low" & missing == "equal")
  dplyr::select(param,MNAR_true,paste0("MAR_",c("mean","sd","MAE"),sep=""),
                paste0("MNAR_",c("mean","sd","MAE"),sep=""))
  mutate(percMAE = MNAR_MAE/MAR_MAE)

colnames(MNAR_simulation_results) <- c("parameter", "true", "MAR_estimates_mean",
  "MAR_estimates_sd", "MAR_estimates_MAE", "MNAR_estimates_mean",
  "MNAR_estimates_sd", "MNAR_estimates_MAE", "percMAE")
```

```
MNAR_simulation_results <- MNAR_simulation_results[c(1:3,7:9,4:6,10:21),]
```

Examples

```
data(MNAR_simulation_results)
```

---

perth *Perth dams water levels.*

Description

Data from the Water Corporation of Western Australia. They state the following about these data on their website: "Streamflow is the amount of water entering our dams from our catchments and is measured by changing water storage levels." This dataset has the annual averages of these storage levels.

Usage

```
data(perth)
```

Format

A data.frame consisting of the following variable:

`water` water level (in GL)

`year` year

`wtmin1` water level in the previous year (GL)

Source

These data are provided by the Water Corporation of Western Australia and can be found here: https://www.watercorporation.com.au/water-supply/rainfall-and-dams/streamflow/streamflowhistorical

Examples

```
# the data is first changed to a timeseries object and then plotted
data(perth)
wts <- ts(perth$water,start=1911)
plot(wts,ylab="GL", main="Perth dams water inflow", xlab="year", frame=FALSE, xaxp=c(1910,2020,10))
```

`SEsamples` *Bootstrap Samples for Simple 2-State Model*

Description

Parametric bootstrap samples for a 2-state hidden Markov model used to compute standard errors.

Usage

```
data("SEsamples")
```

Format

A matrix with 1000 rows for each sample, 12 columns for each parameter of the model, including the parameters that are fixed at their boundary values.

Details

The bootstrap sample was generated by the following code:

```
require(depmixS4)
library(hmmr)
data(simplehmm)

# define the model
set.seed(214)
mod1 <- depmix(obs~1,data=simplehmm,nstates=2,
family=multinomial("identity"), respst=c(.6,0,.4,0,.2,.8), trst=runif(4), inst=c(1,0))

# fit the model
fm1 <- fit(mod1,emcontrol=em.control(random.start=FALSE))

# compute bootstrap samples
nsamples <- 1000
SEsamples <- matrix(0,ncol=npar(fm1),nrow=nsamples)

for(i in 1:nsamples) {
sample <- simulate(fm1)
fmsam <- fit(sample,emcontrol=em.control(random.start=FALSE))
SEsamples[i,] <- getpars(fmsam)
}
```

Examples

```
data(SEsamples)
# standard errors
bootses <- apply(SEsamples,2,sd)
```

```
bootses[which(bootses==0)] <- NA
bootses
# compare with standard errors from finite differences
library(hmmr)
data(simplehmm)
# define the model
set.seed(214)
mod1 <- depmix(obs~1,data=simplehmm,nstates=2,
family=multinomial("identity"), respst=c(.6,0,.4,0,.2,.8), trst=runif(4), inst=c(1,0))
# fit the model
fm1 <- fit(mod1,emcontrol=em.control(random.start=FALSE))
ses <- cbind(standardError(fm1),bootses)
ses
```

---

simplehmm *Hmm toy data set from Visser et al (2000)*

Description

Data are 2000 observations generated from a 2-state hidden Markov model with three observation categories. These data were used in the simuation study in Visser et al. (2000) for testing the quality of several methods of computing parameter standard errors.

Usage

```
data(simplehmm)
```

Format

A dataframe with 2000 observations on a single variable:

`obs` a factor with levels 1, 2, and 3.

Source

Ingmar Visser, Maartje E. J. Raijmakers, and Peter C. M. Molenaar (2000). Confidence intervals for hidden Markov model parameters. *British journal of mathematical and statistical psychology*, 53, p. 317-327.

Examples

```
data(simplehmm)
set.seed(1)
md2 <- hmm(simplehmm, 2)
summary(md2)
```

---

speed1 *Speed Accuracy Switching Data*

Description

This data set is a bivariate series of response times and accuracy scores of a single participant switching between slow/accurate responding and fast guessing on a lexical decision task. The slow and accurate responding, and the fast guessing can be modelled using two states, with a switching regime between them. The dataset further contains a third variable called Pacc, representing the relative pay-off for accurate responding, which is on a scale of zero to one. The value of Pacc was varied during the experiment to induce the switching. This data set is a from participant A in experiment 1a from Dutilh et al (2011). The data here is the first series of 168 trials. The `speed` data set in the `depmixS4` package has two more series of 134 and 137 trials respectively.

Usage

```
data(speed1)
```

Format

A data frame with 168 observations on the following 3 variables.

`RT` a numeric vector of response times (log ms)

`ACC` a numeric vector of accuracy scores (0/1)

`Pacc` a numeric vector of the pay-off for accuracy

Source

Gilles Dutilh, Eric-Jan Wagenmakers, Ingmar Visser, & Han L. J. van der Maas (2011). A phase transition model for the speed-accuracy trade-off in response time experiments. *Cognitive Science*, 35:211-250.

Examples

```
data(speed1)
```

---

speed_boot_LR *speed boot LR*

Description

Example of a parametric bootstrap for model selection

Usage

```
data("speed_boot_LR")
```

Format

A `boot` object

Details

The bootstrap sample was generated by the following code:

```
require(depmixS4)
      require(hmmr)
      require(boot)

      data(speed1)
      set.seed(5)
      spmix2 <- mix(RT~1, data=speed1, nstates=2)
      fspmix2 <- fit(spmix2,verbose=FALSE)

      set.seed(5)
      fspmix3 <- fit(mix(RT~1,data=speed1,nstates=3))

      speed.fun.LR <- function(model) {
        ok <- FALSE
        while(!ok) {
          bootdat <- data.frame(RT = simulate(model)@response[[1]][[1]]@y)
          fboot2 <- try({
            mod <- mix(RT~1,data=bootdat,nstates=2)
            mod <- setpars(mod,getpars(fspmix2))
            fit(mod,emcontrol=em.control(random.start=FALSE),verbose=FALSE)
          },TRUE)
          fboot3 <- try({
            mod <- mix(RT~1,data=bootdat,nstates=3)
            mod <- setpars(mod,getpars(fspmix3))
            fit(mod,emcontrol=em.control(random.start=FALSE),verbose=FALSE)
          },TRUE)
       if(!inherits(fboot2,"try-error") & !inherits(fboot3,"try-error")) ok <- TRUE
        }
        if(ok & logLik(fboot3) < logLik(fboot2)) ok <- FALSE
        while(!ok) {
          cat("trying different starting-values")
          fboot3 <- try({
            mod <- mix(RT~1,data=bootdat,nstates=3)
            mod <- setpars(mod,getpars(fspmix3))
            fit(mod,verbose=FALSE)
          },TRUE)
       if(!inherits(fboot3,"try-error") & logLik(fboot3) > logLik(fboot2)) ok <- TRUE
        }
        llratio(fboot3,fboot2)@value
      }
      set.seed(5)
```

```
        speed_boot_LR <- boot(fspmix2,speed.fun.LR,R=1000,sim="parametric")
```

Examples

```
data(speed_boot_LR)
```

---

speed_boot_LR_extra   *speed boot LR*

Description

Example of a parametric bootstrap for model selection

Usage

```
data("speed_boot_LR")
```

Format

A `boot` object

Details

The bootstrap sample was generated by the following code:

```
require(depmixS4)
      require(hmmr)
      require(boot)

      data(speed1)
      set.seed(5)
      spmix2 <- mix(RT~1, data=speed1, nstates=2)
      fspmix2 <- fit(spmix2,verbose=FALSE)

      set.seed(5)
      fspmix3 <- fit(mix(RT~1,data=speed1,nstates=3))

      speed.fun.LR <- function(model) {
        ok <- FALSE
        while(!ok) {
          bootdat <- data.frame(RT = simulate(model)@response[[1]][[1]]@y)
          fboot2 <- try({
            mod <- mix(RT~1,data=bootdat,nstates=2)
            mod <- setpars(mod,getpars(fspmix2))
            fit(mod,emcontrol=em.control(random.start=FALSE),verbose=FALSE)
          },TRUE)
          fboot3 <- try({
            mod <- mix(RT~1,data=bootdat,nstates=3)
            mod <- setpars(mod,getpars(fspmix3))
            fit(mod,emcontrol=em.control(random.start=FALSE),verbose=FALSE)
          },TRUE)
      if(!inherits(fboot2,"try-error") & !inherits(fboot3,"try-error")) ok <- TRUE
        }
        if(ok & logLik(fboot3) < logLik(fboot2)) ok <- FALSE
        while(!ok) {
          cat("trying different starting-values")
          fboot3 <- try({
            mod <- mix(RT~1,data=bootdat,nstates=3)
            mod <- setpars(mod,getpars(fspmix3))
            fit(mod,verbose=FALSE)
          },TRUE)
      if(!inherits(fboot3,"try-error") & logLik(fboot3) > logLik(fboot2)) ok <- TRUE
        }
        llratio(fboot3,fboot2)@value
      }
      set.seed(5)
      speed_boot_LR <- boot(fspmix2,speed.fun.LR,R=1000,sim="parametric")

      set.seed(7)
      speed_boot_LR_extra <- boot(fspmix2,speed.fun.LR,R=4000,
          sim="parametric",parallel="multicore",ncpus=6)
      (1+sum(c(speed_boot_LR$t,speed_boot_LR_extra$t) >
          llratio(fspmix3,fspmix2)@value))/
          ((speed_boot_LR$R+speed_boot_LR_extra$R)+1)
```

Examples

```
data(speed_boot_LR)
```

---

speed_boot_par *speed boot par*

Description

Example of a parametric bootstrap for paraneter inference

Usage

```
data("speed_boot_par")
```

Format

A boot object

Details

The bootstrap sample was generated by the following code:

```
require(depmixS4)
require(hmmr)
require(boot)

data(speed1)
set.seed(5)
spmix2 <- mix(RT~1, data=speed1, nstates=2)
fspmix2 <- fit(spmix2,verbose=FALSE)

# define a function to produce a bootstrap sample
speed.rg <- function(data,mle) {
  simulate(data)
}
# define what to do with a sample (i.e. estimate parameters)
speed.fun <- function(data) {
  getpars(fit(data,verbose=FALSE,emcontrol=
              em.control(random.start=FALSE)))
}
# produce 1000 bootstrap samples (may take some time!)
speed_boot_par <- boot(fspmix2,speed.fun,R=1000,sim="parametric",
                   ran.gen = speed.rg)
```

Examples

```
data(speed_boot_par)
# confidence intervals
confint <- apply(speed_boot_par$t,2,quantile,probs=c(.025,.975))
colnames(confint) <- c("p1","p2","m1","sd1","m2","sd2")
confint
```

WPT *Weather Prediction Task Data*

Description

This data set contains responses of 11 Parkinsons' patients and 13 age-matched controls on the Weather Prediction Task. Both groups were tested twice. The PD patients were either on or off dopaminergic medication.

Usage

```
data(WPT)
```

Format

A data.frame with 9600 observations on the following variables.

`id` a factor with participant IDs

`group` a factor with group IDs (Parksinson's patient or control)

`med` a factor indicating, for the PD patients, whether they were on dopaminergic medicine or not

`occ` a numeric vector with testing occassions

`trial` a numeric vector with trial numbers

`c1` a numeric (binary) vector indicating whether the first cue was present (1) or not (0)

`c2` a numeric (binary) vector indicating whether the second cue was present (1) or not (0)

`c3` a numeric (binary) vector indicating whether the third cue was present (1) or not (0)

`c4` a numeric (binary) vector indicating whether the fourth cue was present (1) or not (0)

`y` a factor with the actual outcome (Rainy or Fine)

`r` a factor with participants' prediction of the outcome

Source

Speekenbrink, M., Lagnado, D. A., Wilkinson, L., Jahanshahi, M., & Shanks, D. R. (2010). Models of probabilistic category learning in Parkinson's disease: Strategy use and the effects of L-dopa. *Journal of Mathematical Psychology*, *54*, 123-136.

Corresponding author: m.speekenbrink@ucl.ac.uk

Examples

```
data(WPT)

# set up predictors for the different strategies
WPT$sngl <- 0 # singleton strategy
WPT$sngl[WPT$c1 == 1 & rowSums(WPT[,c("c1","c2","c3","c4")]) == 1] <- -1
WPT$sngl[WPT$c2 == 1 & rowSums(WPT[,c("c1","c2","c3","c4")]) == 1] <- -1
WPT$sngl[WPT$c3 == 1 & rowSums(WPT[,c("c1","c2","c3","c4")]) == 1] <- 1
WPT$sngl[WPT$c4 == 1 & rowSums(WPT[,c("c1","c2","c3","c4")]) == 1] <- 1
WPT$sc1 <- 1 - 2*WPT$c1
WPT$sc2 <- 1 - 2*WPT$c2
WPT$sc3 <- -1 + 2*WPT$c3
WPT$sc4 <- -1 + 2*WPT$c4
WPT$mc <- sign(-WPT$c1 - WPT$c2 + WPT$c3 + WPT$c4)

rModels <- list(
list(GLMresponse(formula=r~1,data=WPT,family=binomial())),
list(GLMresponse(formula=r~sngl-1,data=WPT,family=binomial())),
list(GLMresponse(formula=r~sc1-1,data=WPT,family=binomial())),
list(GLMresponse(formula=r~sc2-1,data=WPT,family=binomial())),
list(GLMresponse(formula=r~sc3-1,data=WPT,family=binomial())),
list(GLMresponse(formula=r~sc4-1,data=WPT,family=binomial())),
list(GLMresponse(formula=r~mc-1,data=WPT,family=binomial()))
)
```

```
transition <- list()
for(i in 1:7) {
  transition[[i]] <- transInit(~1,nstates=7,family=multinomial(link="identity"))
}

inMod <- transInit(~1,ns=7,data=data.frame(rep(1,48)),family=multinomial("identity"))

mod <- makeDepmix(response=rModels,transition=transition,
prior=inMod,ntimes=rep(200,48),stationary=TRUE)

fmod <- fit(mod)
```

Index