

Package ‘depmix’

November 9, 2009

Version 0.9.7

Date 2009-11-02

Title Dependent Mixture Models

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

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

Depends R (>= 2.9.1), MASS

Suggests Rdonlp2

Description Fit (multigroup) mixtures of latent Markov models on mixed categorical and continuous (timeseries) data

License GPL

Repository CRAN

Repository/R-Forge/Project depmix

Repository/R-Forge/Revision 312

Date/Publication 2009-11-09 15:57:06

R topics documented:

depmix-internal	2
discrimination	3
dmm	4
fitdmm	7
generate	12
markovdata	14
mgdmm	16
mixdmm	18
speed	19

Index	21
--------------	-----------

 depmix-internal *Depmix utility functions*

Description

These functions are used internally by depmix functions. They should not be called directly unless you know what you're doing.

Usage

```

checkSetRecode(dat, dmm, tdcov, printlevel=1)
recode(dat, xm)
fblo(x, i, bigB)
fbuo(x, i, bigB)
ppar(x, z)
recitt(itemtypes)
pp(x)
np(x)
pa2conr(x)
paridx(nstates, itemtypes, mat, idx1=0, idx2=0, it=0, comp=1, group=1)
fresp(x, pars)
bdiag(x)
cl2st(cluster, dat, dmm)
cl2stob(cluster, dat, dmm)
kmstart(dat, dmm)
poststart(dat, dmm)
tr2stin(sttr)

```

Arguments

<code>dat, xm</code>	See markovdata help.
<code>dmm, itemtypes, nstates</code>	See dmm help.
<code>cluster</code>	Some clustering of the data, eg from kmeans or posterior estimates that can be used to arrive at starting values for parameters.
<code>sttr</code>	transition matrix starting values.
<code>printlevel, tdcov</code>	See depmix help.
<code>x, mat, idx1, idx2, it, comp, group</code>	A vector/matrix(name)/indices et cetera.
<code>i, z, bigB, pars</code>	More internal stuff.

Details

Function `bdiag` takes as argument a list of matrices and returns the blockdiagonal matrix formed from these, and the other entries padded with zeroes. This function is from package `assist` by Chunlei Ke and Yuedong Wang.

Value

Most of these functions are used for their side-effect, ie sending stuff to C-routines, or returning recoded stuff (data, itemtypes) et cetera.

Author(s)

Ingmar Visser <i.visser@uva.nl>

`discrimination` *Discrimination Learning Data*

Description

This data set is from a simple discrimination 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

An object of class `markovdata`.

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.

Description

`dmm` creates an object of class `dmm`, a dependent mixture model.

`lca` creates an object of class `dmm`, `lca`, a latent class model or an independent mixture model.

Usage

```
dmm(nstates, itemtypes, modname = NULL, fixed = NULL,
    stval = NULL, conrows = NULL, conpat = NULL, tdfix =
    NULL, tdst = NULL, linmat = NULL, snames = NULL,
    inames = NULL)
## S3 method for class 'dmm':
summary(object, specs=FALSE, precision=3, se=NULL, ...)

lca(nclasses, itemtypes, modname = NULL, fixed = NULL,
    stval = NULL, conrows = NULL, conpat = NULL,
    linmat = NULL, snames = NULL, inames = NULL)
```

Arguments

<code>nstates</code>	The number of latent states/classes of the model.
<code>nclasses</code>	The number of classes of an <code>lca</code> model, ie the number of states in a <code>dmm</code> model. They are now called classes because they do not change over time.
<code>itemtypes</code>	A vector of length <code>nitems</code> providing the type of measurement, 1 for gaussian data, 2 for a binary item, $n > 3$ for categorical items with n answer possibilities. Answer categories are assumed to be unordered categorical. Ordinal responses can be implemented using inequality and/or linear constraints.
<code>modname</code>	A character string with the name of the model, good when fitting many models. Components of mixture models keep their own names. Names are printed in the summary. Boring default names are provided.
<code>fixed</code>	A vector of length the number of parameters of the model indicating whether parameters are fixed (0) or not (>0). This may be identical to <code>conpat</code> (see below).
<code>stval</code>	Start values of the parameters. These will be random if not specified. Start values must be specified (for all parameters) if there are fixed parameters.
<code>conrows</code>	Argument <code>conrows</code> can be used to specify general constraints between parameters. See details below.
<code>conpat</code>	Argument <code>conpat</code> can be used to specify fixed parameters and equality constraints. It can not be used in conjunction with <code>fixed</code> . See details below.

<code>tdfix, tdst</code>	The first is a logical vector indicating (with 1's) which parameters are dependent on covariates (it should have length <code>npars</code>). <code>tdst</code> provides the starting values for the regression parameters. Using <code>tdcov=TRUE</code> in <code>fitdmm</code> will actually fit the regression parameters. The covariate itself has to be specified in the data as "covariate" (see help on <code>markovdata</code>) and should be scaled to 0-1.
<code>linmat</code>	A complete matrix of linear constraints. This argument is intended for internal use only, it is used by the fit routine to re-create the model with the fitted parameter values. Warning: use of this argument results in complete replacement of the otherwise created matrix <code>A</code> , which contains e.g. sum constraints for transition matrix parameters. If <code>linmat</code> is provided, make sure it is correct, otherwise strange results may occur in fitting models.
<code>snames</code>	Names for the states may be provided in <code>snames</code> . Defaults are <code>State1</code> , <code>State2</code> etc. These are printed in the summary.
<code>inames</code>	Names for items may be provided in <code>inames</code> . Defaults are <code>Item1</code> , <code>Item2</code> etc. They are printed in the summary.
<code>dmm</code>	Object of class <code>dmm</code> .
<code>precision</code>	Precision sets the number of digits to be printed in the summary functions.
<code>se</code>	Vector with standard errors, these are passed on from the <code>summary.fit</code> function if and when <code>ses</code> are available.
<code>specs, ...</code>	Internal use.
<code>object</code>	An object of class <code>dmm</code> .

Details

The function `dmm` creates an object of class `dmm` and sets random initial parameter values if these are not provided. Even though `dmm` is not a mixture of Markov models, the mixture parameter is included in the parameter vector. This is important when specifying constraints. Parameters are ordered as follows: the first parameter(s) are the mixing proportions of the mixture of Markov and/or latent class models. I.e., when a single latent class model or a single Markov chain is fitted, this mixture proportion has value 1.0 and is fixed in estimation. After the mixing proportions, the next parameters in the parameter vector are the transition matrix parameters, the square of `nstates` in row-major order. That is, first the transition probabilities from state 1 to all the other states are given, then the probabilities from state 2 to all the other states etc. Next are the observation matrix parameters. These are provided consecutively for each state/class. In a trichotomous item model with two states has 6 observation parameters; the first three are the probabilities of observing category 1, 2 and 3 respectively in state 1 (which sum to one), and then similarly for state 2. As another example: suppose we have model for one binary item and one gaussian item, in that order, we would have 4 observation parameters for each state, first the probabilities of observing a symbol from category 1 or 2 in state 1, the two parameters, the mean and standard deviation for state 1, and then the same state 2 (see the example in `fitdmm` with data from `rudy`). Finally the initial state probabilities are provided, in the order of the states. In the case of a latent class model or a finite mixture model, these parameters are usually denote as the mixture proportions.

Linear constraints can be set using arguments `conrows` and `conpat`. `conrows` must be contain `nc` by `npars` values, in row major order, with `nc` the number of constraints to be specified. `conrows` is used to define general linear constraints. A row of `conrows` must contain the partial derivatives of a general linear constraint with respect to each of the parameters. Suppose we want the constraint

$x_1 - 2x_2 = 0$, one row of conrows should contain a 1 in position one and -2 in position two and zeroes in the remaining positions. In the function `mixdmm` `conrows` is understood to specify linear constraints on the mixing proportions only. As a consequence, it is not possible to easily constrain parameters between components of a mixture model.

`compat` can be used as a shortcut for both fixed and conrows. It must be a single vector of length `npars` containing 0's (zeroes) for fixed parameters, 1's (ones) for free parameters and higher numbers for possibly equality constrained parameters. E.g. `compat=c(1, 1, 0, 2, 2, 3, 3, 3)` would indicate that pars 1 and 2 are freely estimated, par 3 is fixed at its startvalue (which must be provided in this case), par 4 and 5 are to be estimated equal and pars 6, 7 and 8 are also to be estimated equal.

Value

`dmm` returns an object of class `dmm` which has its own summary method. This will print the parameter values, itemtypes, number of (free) parameters, and the number of states. There is no print method. Using `print` will print all fields of the model which is a list of the following:

<code>modname</code>	See above.
<code>nstates</code>	See above
<code>snames</code>	See above.
<code>nitems</code>	The number of items(=length(itemtypes)).
<code>itemtypes</code>	See above.
<code>inames</code>	See above.
<code>npars</code>	The total parameter count of the model.
<code>nparstotal</code>	The total number of parameters of when the covariate parameters are included.
<code>freepars</code>	The number of freely estimated parameters (it is computed as <code>sum(as.logical(fixed))-rank(qr(A))</code>).
<code>freeparsnotd</code>	The number of freely estimated parameters (it is computed as <code>sum(as.logical(fixed))-rank(qr(A))</code>); this version without the covariate parameters.
<code>pars</code>	A vector of length <code>npars</code> containing parameter values.
<code>fixed</code>	<code>fixed</code> is a (logical) vector of length <code>npars</code> specifying which parameters are fixed and which are not.
<code>A</code>	The matrix <code>A</code> contains the general linear constraints of the model. <code>nrow(A)</code> is the number of linear constraints. <code>A</code> starts with a number of rows for the sum constraints for the transition, observation and initial state parameters, after which the user provided constraints are added.
<code>bu,bl</code>	<code>bu</code> and <code>bl</code> represent the upper and lower bounds of the parameters and the constraints. These vectors are each of length <code>npars + nrow(A)</code> .
<code>bllin,bulin</code>	The lower and upper bounds of the linear constraints.
<code>td,tdin,tdtr,tdob,tdfit</code>	Logicals indicating whether there covariates, in which parameters they are, and whether they are estimated or not (the latter is used to decide whether to print those values or not).
<code>st</code>	Logical indicating whether the model has user specified starting values.

`lca` returns an object of class `dmm`, `lca`, and is otherwise identical to a `dmm` object. The only difference is that the transition matrix parameters are irrelevant, and consequently they are not printed in the summary function.

Author(s)

Ingmar Visser <i.visser@uva.nl>

References

On hidden Markov models: Lawrence R. Rabiner (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77-2, p. 267-295.

On latent class models: A. L. McCutcheon (1987). *Latent class analysis*. Sage Publications.

See Also

[mixdmm](#) on defining mixtures of `dmm`'s, [mgdmm](#) for defining multi group models, and [generate](#) for generating data from models.

Examples

```
# create a 2 state model with one continuous and one binary response
# with start values provided in st
st <- c(1,0.9,0.1,0.2,0.8,2,1,0.7,0.3,5,2,0.2,0.8,0.5,0.5)
mod <- dmm(nsta=2,itemt=c(1,2), stval=st)
summary(mod)

# 2 class latent class model with equal conditional probabilities in each class
stvc=c(1,rep(c(0.9,0.1),5),rep(c(0.1,0.9),5),0.5,0.5)
# here the conditional probs of the first item are set equal to those in
# the subsequent items
conpat=c(1,rep(c(2,3),5),rep(c(4,5),5),1,1)
lc=lca(ncl=2,itemtypes=rep(2,5),conpat=conpat,stv=stvc)
summary(lc)
```

Description

`fitdmm` fits mixtures of hidden/latent Markov models on arbitrary length time series of mixed categorical and continuous data. This includes latent class models and finite mixture models (for time series of length 1), which are in effect independent mixture models.

`posterior` computes the most likely latent state sequence for a given dataset and model.

Usage

```

fitdmm(dat, dmm, printlevel = 1, poster = TRUE, tdcov = 0,
       ses = TRUE, method = "optim", vfactor=15, der = 1, iterlim = 100,
       kmst = !dmm$st, kmrep = 5, postst = FALSE)
loglike(dat, dmm, tdcov = 0, grad = FALSE, hess = FALSE, set
       = TRUE, grInd = 0, sca = 1, printlevel = 1)
posterior(dat, dmm, tdcov=0, printlevel=1)
computeSes(dat, dmm)
bootstrap(object, dat, samples=100, pvalonly=0, ...)
## S3 method for class 'fit':
summary(object, precision=3, fd=1, ...)
onliner(object, precision=3)

```

Arguments

<code>dat</code>	An object (or list of objects) of class <code>md</code> , see <code>markovdata</code> . If <code>dat</code> is a list of objects of class <code>md</code> a multigroup model is fitted on these data sets.
<code>dmm</code>	An object (or a list of objects) of class <code>dmm</code> , see <code>dmm</code> . If <code>dmm</code> is a list of objects of class <code>dmm</code> , these are taken to components of a mixture of <code>dmm</code> 's model and will be coerced to class <code>mixdmm</code> . In any case, the model that is fitted a multigroup mixture of <code>dmm</code> 's with default <code>ngroups=1</code> and number of <code>components=1</code> .
<code>printlevel</code>	<code>printlevel</code> controls the output provided by the C-routines that are called to optimize the parameters. The default of 1 provides minimal output: just the initial and final loglikelihood of the model. Setting higher values will provide more output on the progress the iterations.
<code>poster</code>	By default posteriors are computed, the result of which can be found in <code>fit\$post</code> .
<code>method</code>	This is the optimization algorithm that is used. <code>donlp2</code> from the <code>Rdonlp2</code> package is the default method. There is optional support for <code>NPSOL</code> .
<code>der</code>	Specifies whether derivatives are to be used in optimization.
<code>vfactor</code>	<code>vfactor</code> controls optimization in <code>optim</code> and <code>nlm</code> . Since in those routines there is no possibility for enforcing constraints, constraints are enforced by adding a penalty term to the loglikelihood. The penalty term is printed at the end of optimization if it is not close enough to zero. This may have several reasons. When parameters are estimated at bounds for example. This can be solved by fixing those parameters on their boundary values. When this is not acceptable <code>vfactor</code> may be increased such that the penalty is larger and the probability that they actually hold in the fitted model is correspondingly higher.
<code>tdcov</code>	Logical, when set to <code>TRUE</code> , given that the model and data have covariates, the corresponding parameters will be estimated.
<code>ses</code>	Logical, determines whether standard errors are computed after optimization.
<code>iterlim</code>	The iteration limit for <code>npsol</code> , defaults to 100, which may be too low for large models.
<code>grad</code>	logical; if <code>TRUE</code> the gradients are returned.

<code>hess</code>	logical; if TRUE the hessian is returned; it is not implemented currently and hence setting it to true will produce a warning.
<code>set</code>	Whith the default value TRUE, the data and models parameters are sent to the C/C++ routines before computing the loglikelihood. When set is FALSE, this is not done. If an incorrect model was set earlier in the C-routines this may cause serious errors and/or crashes.
<code>sca</code>	If set to -1.0 the negative loglikelihood, gradients and hessian are returned.
<code>object</code>	An object of class <code>fit</code> , ie the return value of <code>fitdmm</code> .
<code>kmst, postst</code>	These arguments control the generation of starting values by kmeans and posterior estimates respectively.
<code>kmrep</code>	If no starting values are provided, <code>kmrep</code> sets of starting values are generated using kmeans in appropriate cases. The best resulting set of starting values is optimized further.
<code>grInd</code>	Logical argument; if TRUE, individual contributions of each independent realization to the gradient vector will be returned.
<code>fd</code>	Print the finite difference based standard errors in the summary if both those and bootstrapped standard errors are available.
<code>samples</code>	The number of samples to be used in bootstrapping.
<code>pvalonly</code>	Logical, if 1 only a bootstrapped pvalue is returned and not fitted paramaters to compute standard errors, optimization is truncated when the loglikelihood is better than the original loglikelihood.
<code>precision</code>	Precision sets the number of digits to be printed in the summary functions.
<code>...</code>	Used in summary.

Details

The function `fitdmm` optimizes the parameters of a mixture of `dmms` using a general purpose optimization routine subject to linear constraints on the parameters.

Value

`fitdmm` returns an object of class `fit` which has a summary method that prints the summary of the fitted model, and the following fields:

<code>date, timeUsed, totMem</code>	The date that the model was fitted, the time it took to so and the memory usage.
<code>loglike</code>	The loglikelihood of the fitted model.
<code>aic</code>	The AIC of the fitted model.
<code>bic</code>	The BIC of the fitted model.
<code>mod</code>	The fitted model.
<code>post</code>	See function <code>posterior</code> for details.

`loglike` returns a list of the following:

<code>logl</code>	The loglikelihood.
-------------------	--------------------

`gr, grset` `gr` contains the gradients. `grset` is a logical vector giving information as to which gradients are set, currently all gradients are set except the gradients for the mixing proportions.

`hs, hsset` `hs` contains the hessian. `hsset` is a logical giving information as to which elements are computed.

`posterior` returns lists of the following:

`states` A matrix of dimension $2 + \text{sum}(\text{nstates})$ by $\text{sum}(\text{length}(\text{ntimes}))$ containing in the first column the a posteriori component, in the second column the a posteriori state and in the remaining column the posterior probabilities of all states.

`comp` Contains the posterior component number for each independent realization; all ones for a single component model.

`computeSes` returns a vector of length `npars` with the standard errors and a matrix `hs` with the hessian used to compute them. The routine is not fail safe and can produce errors, ie when the (corrected) hessian is singular; a warning is issued when the hessian is close to being singular.

`bootstrap` returns an object of class `fit` with three extra fields, the bootstrapped standard errors, `bse`, a matrix with goodness-of-fit measures of the bootstrap samples, ie `logl`, AIC and BIC and `pbetter`, which is the proportion of bootstrap samples that resulted in better fits than the original model.

`summary.fit` pretty-prints the outputs.

`onliner` returns a vector of `loglike`, `aic`, `bic`, `mod$npars`, `mod$freepars`, `date`.

Note

`fitdmm` fits time series of arbitrary length and mixtures of `dmm`s, where, to the best of my knowledge, other packages are limited due to the different optimization routines that are commonly used for these types of models.

Author(s)

Ingmar Visser <i.visser@uva.nl>, Development of this package was supported by European Commission grant 51652 (NEST) and by a VENI grant from the Dutch Organization for Scientific Research (NWO).

References

Lawrence R. Rabiner (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77-2, p. 267-295.

Theodore C. Lystig and James P. Hughes (2002). Exact computation of the observed information matrix for hidden Markov models. *Journal of Computational and Graphical Statistics*.

See Also

[dmm,markovdata](#)

Examples

```

# COMBINED RT AND CORRECT/INCORRECT SCORES from a 'switching' experiment

data(speed)
mod <- dmm(nsta=2,itemt=c(1,2)) # gaussian and binary items
fit1 <- fitdmm(dat=speed,dmm=mod)
summary(fit1)

# add some constraints using conpat
conpat=rep(1,15)
conpat[1]=0
conpat[14:15]=0
conpat[8:9]=0
# use starting values from the previous model fit, except for the guessing
# parameters which should really be 0.5
stvc=c(1,.896,.104,.084,.916,5.52,.20,.5,.5,6.39,.24,.098,.90,0,1)
mod=dmm(nstates=2,itemt=c("n",2),stval=stvc,conpat=conpat)

fit2 <- fitdmm(dat=speed,dmm=mod)
summary(fit2)

# add covariates to the model to incorporate the fact the accuracy pay off changes per trial
# 2-state model with covariates + other constraints
conpat=rep(1,15)
conpat[1]=0
conpat[8:9]=0
conpat[14:15]=0
conpat[2]=2
conpat[5]=2
stvc=c(1,0.9,0.1,0.1,0.9,5.5,0.2,0.5,0.5,6.4,0.25,0.9,0.1,0,1)
tdfix=rep(0,15)
tdfix[2:5]=1
stcov=rep(0,15)
stcov[2:5]=c(-0.4,0.4,0.15,-0.15)

mod<-dmm(nstates=2,itemt=c("n",2),stval=stvc,conpat=conpat,tdfix=tdfix,tdst=stcov,modname="tw")

fit3 <- fitdmm(dat=speed,dmm=mod,tdcov=1,der=0,ses=0,vfa=80)
summary(fit3)

# split the data into three time series
data(speed)
r1=markovdata(dat=speed[1:168,],item=itemtypes(speed))
r2=markovdata(dat=speed[169:302,],item=itemtypes(speed))
r3=markovdata(dat=speed[303:439,],item=itemtypes(speed))

# define 2-state model with constraints
conpat=rep(1,15)
conpat[1]=0
conpat[8:9]=0
conpat[14:15]=0

```

```

stv=c(1,0.9,0.1,0.1,0.9,5.5,0.2,0.5,0.5,6.4,0.25,0.9,0.1,0,1)
mod<-dmm(nstates=2,itemt=c("n",2),stval=stv,conpat=conpat)

# define 3-group model with equal transition parameters, and no
# equalities between the obser parameters
mgr <-mgdmm(dmm=mod,ng=3,trans=TRUE,obser=FALSE)

fitmg <- fitdmm(dat=list(r1,r2,r3),dmm=mgr)
summary(fitmg)

# LEARNING DATA AND MODELS (with absorbing states)

data(discrimination)

# all or none model with error prob in the learned state
fixed = c(0,0,0,1,1,1,1,0,0,0,0)
stv = c(1,1,0,0.03,0.97,0.1,0.9,0.5,0.5,0,1)
allor <- dmm(nstates=2,itemtypes=2,fixed=fixed,stval=stv,modname="All-or-none")

# Concept identification model: learning only after an error
st=c(1,1,0,0,0,0.5,0.5,0.5,0.25,0.25,0.05,0.95,0,1,1,0,0.25,0.375,0.375)
# fix some parameters
fx=rep(0,19)
fx[8:12]=1
fx[17:19]=1
# add a couple of constraints
conr1 <- rep(0,19)
conr1[9]=1
conr1[10]=-1
conr2 <- rep(0,19)
conr2[18]=1
conr2[19]=-1
conr3 <- rep(0,19)
conr3[8]=1
conr3[17]=-2
conr=c(conr1,conr2,conr3)
cim <- dmm(nstates=3,itemtypes=2,fixed=fx,conrows=conr,stval=st,modname="CIM")

# define a mixture of the above models ...
mix <- mixdmm(dmm=list(allor,cim),modname="MixAllCim")

# ... and fit it on the combined data discrimination
fitmix <- fitdmm(discrimination,mix)
summary(fitmix)

```

Description

`generate` generates a dataset according to a given `dmm`.

Usage

```
generate(ntimes, dmm, nreal=1)
```

Arguments

<code>ntimes</code>	The number of repeated measurements, ie the length of the time series (this may be a vector containing the lengths of independent realizations).
<code>dmm</code>	Object of class <code>dmm</code> or <code>mixdmm</code> .
<code>nreal</code>	The number of independent realizations that is to generated. Each of them will have the dimension of <code>ntimes</code> ; all this does is replace <code>ntimes</code> by <code>rep(ntimes,nreal)</code> .

Details

`generate` generates a date set of the specified dimensions `ntimes` and `nreal` using the parameter values in `dmm`, which should be an object of class `dmm` or `mixdmm`. `generate` does not handle multi group models, which can be run separately.

This function is used in the `bootstrap`'ping routine to compute standard errors based on parametric bootstraps.

Value

Generate returns an object of class `markovdata`. The return object has an attribute called `instates`, a vector with the starting states of each realization. When the model is a mixture the return has another attribute `incomp` containing the components of each realization.

Author(s)

Ingmar Visser <i.visser@uva.nl>

See Also

[dmm](#), [markovdata](#)

Examples

```
# create a 2 state model with one continuous and one binary response
# with start values provided in st
st <- c(1,0.9,0.1,0.2,0.8,2,1,0.7,0.3,5,2,0.2,0.8,0.5,0.5)
mod <- dmm(nsta=2,itemt=c(1,2), stval=st)

# generate two series of lengths 100 and 50 respectively using above model
gen<-generate(c(100,50),mod)
```

```
summary(gen)
plot(gen)
```

```
markovdata
```

```
Specifying Markov data objects
```

Description

Markovdata creates an object of class md, to be used by `fitdmm`.

Usage

```
markovdata(dat, itemtypes, nitens = length(itemtypes), ntimes =
  length(as.matrix(dat))/nitens, replicates = rep(1,
  length(ntimes)), inames = NULL, dname = NULL, xm =
  NA)

## S3 method for class 'md':
summary(object, ...)
## S3 method for class 'md':
plot(x, nitens = 1:(min(5, dim(x)[2])),
  nind = 1:(min(5, length(attributes(x)$ntimes))), ...)
## S3 method for class 'md':
print(x, ...)

dname(object)
ntimes(object)
itemtypes(object)
replicates(object)

ncov(object)
inames(object)
nitens(object)
ind(object)
```

Arguments

<code>dat</code>	An R object to be coerced to markovdata, a data frame or matrix.
<code>itemtypes</code>	A vector providing the types of measurement with possible values ‘continuous’, ‘categorical’, and ‘covariate’. This is mainly only used to rearrange the data when there are covariates in such a way that the covariate is in the last column. Only one covariate is supported in estimation of models.

<code>ntimes</code>	The number of repeated measurements, ie the length of the time series (this may be a vector containing the lengths of independent realizations). It defaults the number of rows of the data frame or data matrix.
<code>replicates</code>	Using this argument case weights can be provided. This is particularly usefull in eg latent class analysis with categorical variables when there usually are huge numbers of replicates, ie identical response patterns. <code>depmix</code> computes the raw data log likelihood for each case separately. Thus, when there are many replicates of a case a lot of computation time is saved by specifying case weights instead of providing the full data set.
<code>inames</code>	The names of items. These default to the column names of matrices or dataframes.
<code>dname</code>	The name of the dataset, used in summary, print and plot functions.
<code>xm</code>	<code>xm</code> is the missing data code. It can be any value but zero. Missing data are recoded into NA.
<code>object, x</code>	An object of class <code>md</code> .
<code>...</code>	Further arguments passed on to plot and summary.
<code>nitems, nind</code>	In the plot function, these arguments control which data are to be plotted, ie <code>nitems</code> indicates a range of items, and <code>nind</code> a range of realizations, respectively.

Details

The function `markovdata` coerces a given data frame or matrix to be an object of class `md` such that it can be used in `fitdmm`. The `md` object has its own summary, print and plot methods.

The functions `dname`, `itemtypes`, `ntimes`, and `replicates` retrieve the respective attributes with these names; similarly `ncov`, `nitems`, `inames`, and `ind` retrieve the number of covariates, the number of items (the number of columns of the data), the column names and the number of independent realizations respectively.

Value

An `md`-object is a matrix of dimensions `sum(ntimes)` by `nitems`, containing the measured variables and covariates rearranged such that the covariate appears in the last column. The column names are `inames` and the matrix has three further attributes:

<code>dname</code>	The name of the data set.
<code>itemtypes</code>	See above.
<code>ntimes</code>	See above. This will be a vector computed as <code>ntimes=rep(ntimes,nreal)</code> .
<code>replicates</code>	The number of replications of each case, used as weigths in computing the log likelihood.

Author(s)

Ingmar Visser <i.visser@uva.nl>

See Also

[dmm](#), [depmix](#)

Examples

```

x=rnorm(100,10,2)
y=ifelse(runif(100)<0.5,0,1)
z=matrix(c(x,y),100,2)
md=markovdata(z,itemtypes=c("cont","cat"))
summary(md)

data(speed)
summary(speed)
plot(speed,nind=2)

# split the data into three data sets
# (to perform multi group analysis)
r1=markovdata(dat=speed[1:168,],item=itemtypes(speed))
r2=markovdata(dat=speed[169:302,],item=itemtypes(speed))
r3=markovdata(dat=speed[303:439,],item=itemtypes(speed))
summary(r2)

```

mgdmm

Multi group model specification

Description

mgdmm creates an object of class mgd, a multi-group model, from a given model of either class dmm or class mixdmm or lists of these.

Usage

```

mgdmm(dmm,ng=1,modname=NULL,trans=FALSE,obser=FALSE,init=FALSE,conpat=NULL)
## S3 method for class 'mgd':
summary(object, specs=FALSE, precision=3, se=NULL, ...)

```

Arguments

modname	A character string with the name of the model, good when fitting many models. Components of mixture models keep their own names. Names are printed in the summary. Boring default names are provided.
dmm	Object (or list of objects) of class dmm; see details below.
ng	Number of groups for a multigroup model.
trans, obser, init	Logical arguments specify whether transition parameters, observation parameters and initial state parameters should be estimated equal across groups.
conpat	Can be used to specify general linear constraints. See dmm for details.

precision	Precision sets the number of digits to be printed in the summary functions.
se	Vector with standard errors, these are passed on from the summary.fit function if and when ses are available.
specs, ...	Internal use.
object	An object of class mgd.

Details

The function `mgdmm` can be used to define an `mgd`-model or multi group `dmm`. Its default behavior is to create `ng` copies of the `dmm` argument, thereby providing identical starting values for each group's model. If the `dmm` argument is a list of models of length `ng`, the starting values of those models will be used instead. This may save quite some cpu time when fitting large models by providing the parameter values of separately fitted models as starting values. Currently, `depmix` does not automatically generate starting values for multi group models.

Value

`mgdmm` returns an object of class `mgd` which contains all the fields of an object of class `dmm` and the following extra:

<code>ng</code>	<code>ng</code> is the number of groups in the multigroup model.
<code>mixmod</code>	<code>mixmod</code> is a list of length <code>ng</code> of <code>mixdmm</code> models for each group.
<code>itemtypes</code>	See above.
<code>npars, freepars, pars, fixed, A, bl, bu</code>	The same as above but now for the combined model, here <code>npars</code> equals the sum of <code>npars</code> of the component models plus the mixing proportions.

Author(s)

Ingmar Visser <i.visser@uva.nl>

See Also

[dmm](#) on defining single component models, and [mixdmm](#) for defining mixtures of `dmm`'s.

Examples

```
# create a 2 state model with one continuous and one binary response
# with start values provided in st
st <- c(1,0.9,0.1,0.2,0.8,2,1,0.7,0.3,5,2,0.2,0.8,0.5,0.5)
mod <- dmm(nsta=2,itemt=c(1,2), stval=st)

# define 3-group model with equal transition parameters, and no
# equalities between the obser parameters
mgr <- mgdmm(dmm=mod,ng=3,trans=TRUE,obser=FALSE)
summary(mgr)
```

mixdmm *Mixture of dmm's specification*

Description

`mixdmm` creates an object of class `mixdmm`, ie a mixture of `dmm`'s, given a list of component models of class `dmm`.

Usage

```
mixdmm(dmm, modname=NULL, mixprop=NULL, conrows=NULL)
## S3 method for class 'mixdmm':
summary(object, specs=FALSE, precision=3, se=NULL, ...)
```

Arguments

<code>dmm</code>	A list of <code>dmm</code> objects to form the mixture.
<code>modname</code>	A character string with the name of the model, good when fitting many models. Components of mixture models keep their own names. Names are printed in the summary. Boring default names are provided.
<code>conrows</code>	Argument <code>conrows</code> can be used to specify general constraints between parameters.
<code>mixprop</code>	Argument <code>mixprop</code> can be used to set the initial values of the mixing proportions of a mixture of <code>dmm</code> 's.
<code>precision</code>	Precision sets the number of digits to be printed in the summary functions.
<code>object</code>	An object of class <code>mixdmm</code> .
<code>specs, ...</code>	Internal use. Not functioning currently.
<code>se</code>	Vector with standard errors, these are passed on from the <code>summary.fit</code> function if and when <code>ses</code> are available.

Details

The function `mixdmm` can be used to define a mixture of `dmm`'s by providing a list of such objects as argument to this function. See the `dmm` helpfile on how to use the `conrows` argument. Note that it has to be of length `npars`, ie including all parameters of the model and not just the mixing proportions.

Value

`mixdmm` returns an object of class `mixdmm` which has the same fields as a `dmm` object. In addition it has the following fields:

<code>nrcomp</code>	The number of components of the mixture model.
<code>mod</code>	A list of the component models, that is a list of objects of class <code>dmm</code> .

Author(s)

Ingmar Visser <i.visser@uva.nl>

See Also

[dmm](#) on defining single component models, and [mgdmm](#) on defining multi group models. See [generate](#) for generating data.

Examples

```
# define component 1
# all or none model with error prob in the learned state
fixed = c(0,0,0,1,1,1,1,1,0,0,0,0)
stv = c(1,1,0,0.07,0.93,0.9,0.1,0.5,0.5,0,1)
allor <- dmm(nstates=2,itemtypes=2,fixed=fixed,stval=stv,modname="All-or-none")

# define component 2
# Concept identification model: learning only after an error
st=c(1,1,0,0,0,0.5,0.5,0.5,0.25,0.25,0.8,0.2,1,0,0,1,0.25,0.375,0.375)
# fix some parameters
fx=rep(0,19)
fx[8:12]=1
fx[17:19]=1
# add a couple of constraints
conr1 <- rep(0,19)
conr1[9]=1
conr1[10]=-1
conr2 <- rep(0,19)
conr2[18]=1
conr2[19]=-1
conr3 <- rep(0,19)
conr3[8]=1
conr3[17]=-2
conr=c(conr1,conr2,conr3)
cim <- dmm(nstates=3,itemtypes=2,fixed=fx,conrows=conr,stval=st,modname="CIM")

# define a mixture of the above component models
mix <- mixdmm(dmm=list(allor,cim),modname="MixAllCim")
summary(mix)
```

speed

Speed Accuracy Switching Data

Description

This data set is a bivariate series of reaction times and accuracy scores of a single subject switching between slow and 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 subset of data from experiment 2 in *Van der Maas et al, 2005*.

Usage

```
data(speed)
```

Format

An object of class `markovdata`.

Source

Han L. J. Van der Maas, Conor V. Dolan and Peter C. M. Molenaar (2005), Phase Transitions in the Trade-Off between Speed and Accuracy in Choice Reaction Time Tasks. *Manuscript in revision*.

Index

- *Topic **datagen**
 - generate, 12
- *Topic **datasets**
 - discrimination, 3
 - speed, 19
- *Topic **data**
 - markovdata, 14
- *Topic **models**
 - depmix-internal, 2
 - dmm, 4
 - fitdmm, 7
 - mgdmm, 16
 - mixdmm, 18

- bdiag (*depmix-internal*), 2
- bootstrap (*fitdmm*), 7

- checkSetRecode (*depmix-internal*), 2
- cl2st (*depmix-internal*), 2
- cl2stob (*depmix-internal*), 2
- computeSes (*fitdmm*), 7

- depmix, 15
- depmix (*fitdmm*), 7
- depmix-internal, 2
- discrimination, 3
- dmm, 4, 10, 13, 15–17, 19
- dname (*markovdata*), 14

- fblo (*depmix-internal*), 2
- fbuo (*depmix-internal*), 2
- fitdmm, 7, 14, 15
- fresp (*depmix-internal*), 2

- generate, 7, 12, 19

- inames (*markovdata*), 14
- ind (*markovdata*), 14
- itemtypes (*markovdata*), 14

- kmstart (*depmix-internal*), 2

- lca (*dmm*), 4
- lcm (*dmm*), 4
- loglike (*fitdmm*), 7

- markovdata, 10, 13, 14
- mgdmm, 7, 16, 19
- mixdmm, 7, 17, 18

- ncov (*markovdata*), 14
- nitems (*markovdata*), 14
- np (*depmix-internal*), 2
- ntimes (*markovdata*), 14

- oneline (*fitdmm*), 7

- pa2conr (*depmix-internal*), 2
- paridx (*depmix-internal*), 2
- plot.md (*markovdata*), 14
- plot.ts2 (*markovdata*), 14
- posterior (*fitdmm*), 7
- poststart (*depmix-internal*), 2
- pp (*depmix-internal*), 2
- ppar (*depmix-internal*), 2
- print.md (*markovdata*), 14

- recitt (*depmix-internal*), 2
- recode (*depmix-internal*), 2
- replicates (*markovdata*), 14

- speed, 19
- summary.dmm (*dmm*), 4
- summary.fit (*fitdmm*), 7
- summary.md (*markovdata*), 14
- summary.mgd (*mgdmm*), 16
- summary.mixdmm (*mixdmm*), 18

- tr2stin (*depmix-internal*), 2