Package ‘depmix’

February 22, 2016

Version 0.9.14
Date 2016-02-11
Title Dependent Mixture Models
Author Ingmar Visser <i.visser@uva.nl>
Maintainer Ingmar Visser <i.visser@uva.nl>
Depends R (>= 3.1.2), MASS
Suggests Rdonlp2
Additional_repositories http://R-Forge.R-project.org
Description Fits (multigroup) mixtures of latent or hidden Markov models on mixed categori-
cal and continuous (timeseries) data. The Rdonlp2 package can optionally be used for optimiza-
tion of the log-likelihood and is available from R-forge.
License GPL-3
Repository CRAN
Repository/R-Forge/Project depmix
Repository/R-Forge/Revision 651
Repository/R-Forge/DateTimeStamp 2016-02-12 07:50:29
Date/Publication 2016-02-22 11:54:19

R topics documented:

depmix-internal .............................................. 2
discrimination ............................................... 3
dmm .......................................................... 4
fiddmm ......................................................... 7
generate ...................................................... 13
markovdata ................................................... 14
mgdmm ......................................................... 16
mixdmm ....................................................... 18
speed ......................................................... 20

Index 21
**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**

```r
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,lt=0,comp=1,group=1)
fresp(x,pars)
bdiag(x)
c12st(cluster,dat,dmm)
c12stob(cluster,dat,dmm)
kmstart(dat,dmm)
poststart(dat,dmm)
tr2stin(sttr)
```

**Arguments**

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

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


**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**

```r
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, ...)
```

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

**Arguments**

- `nstates` The number of latent states/classes of the model.
- `nclasses` The number of classes of an `lca` model, i.e., the number of states in a `dmm` model. They are now called classes because they do not change over time.
- `itemtypes` A vector of length `nitems` 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.
- `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.
- `fixed` 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 `conpat` (see below).
- `stval` 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.
- `conrows` Argument `conrows` can be used to specify general constraints between parameters. See details below.
- `conpat` Argument `conpat` can be used to specify fixed parameters and equality constraints. It cannot be used in conjunction with `fixed`. See details below.
The first is a logical vector indicating (with 1’s) which parameters are dependent
on covariates (it should have length npars). tdst provides the starting values for
the regression parameters. Using tdcov=TRUE in fitdmm will actually fit the
regression parameters. The covariate itself has to be specified in the data as
"covariate" (see help on markovdata) and should be scaled to 0-1.

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 param-
eter values. Warning: use of this argument results in complete replacement of
the otherwise created matrix A, which contains e.g. sum contraints for transition
matrix parameters. If linmat is provided, make sure it is correct, otherwise
strange results may occur in fitting models.

Names for the states may be provided in snames. Defaults are State1, State2 etc.
These are printed in the summary.

Names for items may be provided in inames. Defaults are Item1, Item2 etc.
They are printed in the summary.

Object of class dmm.

Precision sets the number of digits to be printed in the summary functions.

Vector with standard errors, these are passed on from the summary.fit function
if and when ses are available.

Internal use.

An object of class dmm.

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 it 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. I.e 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 conmat. conrows must be contain nc
by npars values, in row major order, with nc the number of contraints 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
x1 -2*x2=0, one row of conrows should contain a 1 in position one and -2 in position 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.

conpat 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. conpat=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 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:

modname See above.
nstates See above
snames See above.
nitens The number of items(=length(itemtypes)).
itemtypes See above.
inames See above.
npars The total parameter count of the model.
nparstotal The total number of parameters when the covariate parameters are included.
freepars The number of freely estimated parameters (it is computed as sum(as.logical(fixed))-
rank(qr(A)).
freeparsnotd The number of freely estimated parameters (it is computed as sum(as.logical(fixed))-
rank(qr(A)); this version without the covariate parameters.
pars A vector of length npars containing parameter values.
fixed fixed is a (logical) vector of length npars specifying which parameters are fixed
and which are not.
A The matrix A contains the general linear constraints of the model. nrow(A)
is the number of linear constraints. A 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.

bu,bl bu and bl represent the upper and lower bounds of the parameters and the con-
straints. These vectors are each of length npars + nrow(A).
blin,blin The lower and upper bounds of the linear constraints.
td,tdin,tdtr,tdob,tdfit
Logicals indicating whehter 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).
st Logical indicating whether the model has user specified starting values.
1ca returns an object of class dmm, 1ca, 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

See Also
mixdmm on defining mixtures of dmm's, mgdmm for defining multi group models, and generate for generating data from models.

Examples

```r
# 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
stv=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=stv)
summary(lc)
```

fitdmm  

**Fitting Dependent Mixture Models**

**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 = 1,dmm$st, kmrep = 5, postst = FALSE)
lolike(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, ...)
oneliner(object,precision=3)

Arguments

dat An object (or list of objects) of class md, see markovdata. If dat is a list of objects
of class md a multigroup model is fitted on these data sets.
dmm An object (or a list of objects) of class dmm, see dmm. If dmm is a list of objects
of class dmm, these are taken to components of a mixture of dmm’s model and
will be coerced to class mixdmm. In any case, the model that is fitted a multigroup
mixture of dmm’s with default ngroups=1 and number of components=1.
printlevel printlevel 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.
poster By default posteriors are computed, the result of which can be found in fit$post.
method This is the optimization algorithm that is used. donlp2 from the Rdonlp2 pack-
age is the default method. There is optional support for NPSOL.
der Specifies whether derivatives are to be used in optimization.
vfactor vfactor controls optimization in optim and nlm. 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
vfactor may be increased such that the penalty is larger and the probability that
they actually hold in the fitted model is correspondingly higher.
tdcov Logical, when set to TRUE, given that the model and data have covariates, the
 corresponding parameters will be estimated.
seq Logical, determines whether standard errors are computed after optimization.
iterlim The iteration limit for npsol, defaults to 100, which may be too low for large
models.
grad logical; if TRUE the gradients are returned.
fitdmm

hess logical; if TRUE the hessian is returned; it is not implemented currently and hence setting it to true will produce a warning.

set With 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.

sca If set to -1.0 the negative loglikelihood, gradients and hessian are returned.

object An object of class fit, ie the return value of fitdmm.

kmst,postst These arguments control the generation of starting values by kmeans and posterior estimates respectively.

kmrep If no starting values are provided, kmrep sets of starting values are generated using kmeans in appropriate cases. The best resulting set of starting values is optimized further.

grInd Logical argument; if TRUE, individual contributions of each independent realization to the gradient vector will be returned.

fd Print the finite difference based standard errors in the summary if both those and bootstrapped standard errors are available.

samples The number of samples to be used in bootstrapping.

pvalonly Logical, if 1 only a bootstrapped pvalue is returned and not fitted parameters to compute standard errors, optimization is truncated when the loglikelihood is better than the original loglikelihood.

precision Precision sets the number of digits to be printed in the summary functions.

... Used in summary.

Details

The function fitdmm optimizes the parameters of a mixture of dmm's 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:

date, timeUsed, totMem The date that the model was fitted, the time it took to so and the memory usage.

loglike The loglikelihood of the fitted model.

aic The AIC of the fitted model.

bic The BIC of the fitted model.

mod The fitted model.

post See function posterior for details.

loglike returns a list of the following:

logl 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+sum(nstates) by sum(length(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.

oneliner 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


See Also
dmm,markovdata
Examples

```r
# COMBINED RT AND CORRECT/INCORRECT SCORES from a 'switching' experiment
data(speed)
mod <- dmm(nsta=2,itemt=c(1,2)) # gaussian and binary items
ll <- loglike(speed,mod)
fit1 <- fitdmm(dat=speed,dmm=mod)
summary(fit1)
ll <- loglike(speed,fit1)

# bootstrap
## Not run:
pst <- posterior(dat=speed,dmm=fit1)
bs <- bootstrap(fit1,speed,samples=50)

## End(Not run) # end not run

# 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
stv=c(1,0.09,0.01,0.1,0.09,0.5,0.2,0.5,0.5,0.6,0.39,0.24,0.09,0.9,0.1,0.1)
mod=dmm(nstates=2,itemt=c("n",2),stval=stv,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

## Not run:
conpat=rep(1,15)
conpat[1]=0
conpat[8:9]=0
conpat[14:15]=0
conpat[2]=2
conpat[5]=2
stv=c(1,0.9,0.1,0.1,0.9,0.5,0.2,0.5,0.5,0.6,0.4,0.25,0.9,0.1,0.1)
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=stv,conpat=conpat,tdfix=tdfix,tdst=stcov,
modname="towboth+cov")
```
fit3 <- fitdmm(dat=speed,dmm=mod,tdecov=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(1.0,0.9,0.1,0.1,0.9,0.5,0.2,0.5,0.5,0.6,0.4,0.2,0.2,0.9,0.1,0.1)
mod<dmm(nstates=2,item=conpat,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)

## End(Not run) # end not run

## LEARNING DATA AND MODELS (with absorbing states)

## Not run:
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(1,1,0,0,0,0.03,0.97,0.1,0.9,0.5,0.5,0.1)
allor <- dmm(nstates=2,itemtypes=2,fixed=fixed,stval=stv,modname="All-or-none")

# Concept identification model: learning only after an error
st=c(1,0,1,0,0,0.5,0.5,0.25,0.25,0.05,0.95,0.0,1,0,0,25,0.375,0.375)
# 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)
```r
conr3[8]=1
corr3[17]=-2
corr=c(conr1,corr2,conr3)
cim <- dmm(nstates=3,itemtypes=2,fix=fx,conrows=conr,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)

## End(Not run) # end not run
```

---

**generate**

*Generate data from a dependent mixture model*

**Description**

generate generates a dataset according to a given dmm.

**Usage**

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

**Arguments**

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

**Details**

generate generates a dataset 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 &lt;i.visser@uva.nl&gt;

See Also

dmm, markovdata

Examples

```r
# 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

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

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

```r
## S3 method for class 'md'
plot(x, nitems = 1:(min(5, dim(x)[2])),
    nind = 1:(min(5,length(attributes(x)$ntimes))),...)```
The function \texttt{markovdata} coerces a given data frame or matrix to be an object of class \texttt{md} such that it can be used in \texttt{fitdmm}. The \texttt{md} object has its own summary, print and plot methods.

The functions \texttt{dname}, \texttt{itemtypes}, \texttt{ntimes}, and \texttt{replicates} retrieve the respective attributes with these names; similarly \texttt{ncov}, \texttt{nitems}, \texttt{inames}, and \texttt{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 \( \text{sum(ntimes)} \) by \( \text{nitems} \), containing the measured variables and covariates rearranged such that the covariate appears in the last column. The column names are \( \text{inames} \) and the matrix has three further attributes:

- **dnname**: The name of the data set.
- **itemtypes**: See above.
- **ntimes**: See above. This will be a vector computed as \( \text{ntimes}=\text{rep(ntimes,nreal)} \).
- **replicates**: The number of replications of each case, used as weights in computing the log likelihood.

Author(s)

Ingmar Visser <i.visser@uva.nl>

See Also

`dmm`, `depmix`

Examples

```r
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)
```

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**

```r
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:

- `ng` Number of groups in the multigroup model.
- `mixmod` `mixmod` is a list of length `ng` of `mixdmm` models for each group.
- `itemtypes` See above.
- `npars, freepars, pars, fixed, A, b1, bu` The same as above but now for the combined model, here `npars` equals the sum of `npars` 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

dmm A list of dmm objects to form the mixture.
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.
conrows Argument conrows can be used to specify general constraints between parameters.
mixprop Argument mixprop can be used to set the initial values of the mixing proportions of a mixture of dmm's.
Precision sets the number of digits to be printed in the summary functions.

An object of class `mixdmm`.

Internal use. Not functioning currently.

Vector with standard errors, these are passed on from the summary.fit function if and when ses 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:

- `nrcomp`: The number of components of the mixture model.
- `mod`: A list of the component models, that is a list of objects of class `dmm`.

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

```r
# define component 1
# 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(1,1,0,0.07,0.93,0.9,0.1,0.5,0.5,0,1)
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(1,1,0,0,0.5,0.5,0.25,0.25,0.25,0.25,0.25,0.375,0.375)
# fix some parameters
fx=rep(0,19)
fx[8:12]=1
fx[17:19]=1
# add a couple of constraints
corr1 <- rep(0,19)
corr1[9]=1
corr1[10]=-1
corr2 <- 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

<table>
<thead>
<tr>
<th>Speed</th>
<th>Accuracy</th>
<th>Switching</th>
<th>Data</th>
</tr>
</thead>
</table>

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, 13

*Topic datasets
discrimination, 3
speed, 20

*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
c12St (depmix-internal), 2
c12stob (depmix-internal), 2
computeSes (fitdmm), 7

depmix, 16
depmix (fitdmm), 7
depmix-internal, 2
discrimination, 3
dmm, 4, 10, 14, 16–19
dname (markovdata), 14

fblo (depmix-internal), 2
fbuo (depmix-internal), 2
fitdmm, 7, 14, 15
fresp (depmix-internal), 2
generate, 7, 13, 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, 14, 14
mgdmm, 7, 16, 19
mixdmm, 7, 18, 18
ncov (markovdata), 14
nitems (markovdata), 14
np (depmix-internal), 2
ntimes (markovdata), 14
oneliner (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, 20
summary.dmm (dmm), 4
summary.fit (fitdmm), 7
summary.md (markovdata), 14
summary.mgd (mgdmm), 16
summary.mixdmm (mixdmm), 18
tr2stoin (depmix-internal), 2