

Package ‘depmixS4’

July 2, 2014

Version 1.3-2

Date 2014-02-04

Title Dependent Mixture Models - Hidden Markov Models of GLMs and Other Distributions in S4

Author Ingmar Visser <i.visser@uva.nl>, Maarten Speekenbrink <m.speekenbrink@ucl.ac.uk>

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

Depends R (>= 3.0.1), nnet, MASS, Rsolnp

Imports stats, stats4, methods

Suggests gamlss, gamlss.dist

Description Fit latent (hidden) Markov models on mixed categorical and continuous (time series) data, otherwise known as dependent mixture models

License GPL (>= 2)

URL <http://depmix.r-forge.r-project.org/>

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-02-04 16:16:00

R topics documented:

depmixS4-package	2
balance	3
depmix	4
depmix-class	8
depmix-methods	9
depmix.fitted-class	12
depmix.sim-class	13
em.control	14
fit	16

forwardbackward	20
GLMresponse	22
llratio	23
makeDepmix	24
mix	29
mix-class	31
mix.fitted-class	32
mix.sim-class	33
posterior	34
response-class	35
response-classes	36
responses	37
simulate	39
sp500	41
speed	42
transInit	43

Index	45
--------------	-----------

depmixS4-package	<i>depmixS4 provides classes for specifying and fitting hidden Markov models</i>
------------------	--

Description

depmixS4 is a framework for specifying and fitting dependent mixture models, otherwise known as hidden or latent Markov models. Optimization is done with the EM algorithm or optionally with Rdonlp2 when (general linear (in-)equality) constraints on the parameters need to be incorporated. Models can be fitted on (multiple) sets of observations. The response densities for each state may be chosen from the GLM family, or a multinomial. User defined response densities are easy to add; for the latter an example is given for the ex-gauss distribution as well as the multivariate normal distribution.

Mixture or latent class (regression) models can also be fitted; these are the limit case in which the length of observed time series is 1 for all cases.

Details

Package: depmixS4
 Type: Package
 Version: 1.3-0
 Date: 2013-09-17
 License: GPL

Model fitting is done in two steps; first, models are specified through the `depmix` function (or the `mix` function for mixture and latent class models), which both use standard `glm` style arguments to specify the observed distributions; second, the model needs to be fitted by using the `fit` func-

tion; imposing constraints is done through the fit function. Standard output includes the optimized parameters and the posterior densities for the states and the optimal state sequence.

For full control and the possibility to add new response distributions, check the [makeDepmix](#) help page.

Author(s)

Ingmar Visser & Maarten Speekenbrink

Maintainer: i.visser@uva.nl

References

Ingmar Visser and Maarten Speekenbrink (2010). `depmixS4`: An R Package for Hidden Markov Models. *Journal of Statistical Software*, 36(7), p. 1-21.

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

[depmix](#), [fit](#)

Examples

```
# create a 2 state model with one continuous and one binary response
data(speed)
mod <- depmix(list(rt~1,corr~1),data=speed,nstates=2,family=list(gaussian(),multinomial()))
# print the model, formulae and parameter values (ie the starting values)
mod
```

balance

Balance Scale Data

Description

Balance scale data of four distance items from 779 participants; participants ages are included.

Usage

```
data(balance)
```

Format

A data frame with 779 observations on the following variables. The full dataset is described and analyzed extensively in Jansen & Van der Maas (2002). The trichotomous data are left, balance, right. The dichotomous version of the data is scored correct, incorrect.

sex Participants sex.
agedays Age in days.
age Age in years.
t1 Trichotomously scored distance item.
t2 Trichotomously scored distance item.
t3 Trichotomously scored distance item.
t4 Trichotomously scored distance item.
d1 Dichotomously scored distance item.
d2 Dichotomously scored distance item.
d3 Dichotomously scored distance item.
d4 Dichotomously scored distance item.

Source

Brenda Jansen & Han van der Maas (2002). The development of children's rule use on the balance scale task. *Journal of experimental Child Psychology*, 81, p. 383-416.

Examples

```
data(balance)
```

depmix	<i>Dependent Mixture Model Specification</i>
--------	--

Description

depmix creates an object of class depmix, a dependent mixture model, otherwise known as hidden Markov model. For a short description of the package see [depmixS4](#). See the vignette for an introduction to hidden Markov models and the package.

Usage

```
depmix(response, data=NULL, nstates, transition=~1, family=gaussian(),  
prior=~1, initdata=NULL, respstart=NULL, trstart=NULL, instart=NULL,  
ntimes=NULL,...)
```

Arguments

response	The response to be modeled; either a formula or a list of formulae (in the multivariate case); this interfaces to the <code>glm</code> and other distributions. See 'Details'.
data	An optional <code>data.frame</code> to interpret the variables in the response and transition arguments.
nstates	The number of states of the model.
transition	A one-sided formula specifying the model for the transitions. See 'Details'.
family	A family argument for the response. This must be a list of family's if the response is multivariate.
prior	A one-sided formula specifying the density for the prior or initial state probabilities.
initdata	An optional <code>data.frame</code> to interpret the variables occurring in <code>prior</code> . The number of rows of this <code>data.frame</code> must be equal to the number of cases being modeled, <code>length(ntimes)</code> . See 'Details'.
respstart	Starting values for the parameters of the response models.
trstart	Starting values for the parameters of the transition models.
instart	Starting values for the parameters of the prior or initial state probability model.
ntimes	A vector specifying the lengths of individual, i.e. independent, time series. If not specified, the responses are assumed to form a single time series, i.e. <code>ntimes=nrow(data)</code> . If the <code>data</code> argument has an attribute <code>ntimes</code> , then this is used. The first example in <code>fit</code> uses this argument.
...	Not used currently.

Details

The function `depmix` creates an S4 object of class `depmix`, which needs to be fitted using `fit` to optimize the parameters.

The response model(s) are by default created by call(s) to `GLMresponse` using the `formula` and the `family` arguments, the latter specifying the error distribution. See `GLMresponse` for possible values of the `family` argument for `glm`-type responses (ie a subset of the `glm` family options, and the multinomial). Alternative response distributions are specified by using the `makeDepmix` function. Its help page has examples of specifying a model with a multivariate normal response, as well as an example of adding a user-defined response model, in this case for the `ex-gauss` distribution.

If `response` is a list of formulae, the response's are assumed to be independent conditional on the latent state.

The transitions are modeled as a multinomial logistic model for each state. Hence, the transition matrix can be modeled using time-varying covariates. The prior density is also modeled as a multinomial logistic. Both of these models are created by calls to `transInit`.

Starting values for the initial, transition, and response models may be provided by their respective arguments. NB: note that the starting values for the initial and transition models as well as of the multinomial logit response models are interpreted as probabilities, and internally converted to multinomial logit parameters. The order in which parameters must be provided can be easily studied by using the `setpars` and `getpars` functions.

Linear constraints on parameters can be provided as argument to the `fit` function.

The `print` function prints the formulae for the response, transition and prior models along with their parameter values.

Missing values are allowed in the data, but missing values in the covariates lead to errors.

Value

`depmix` returns an object of class `depmix` which has the following slots:

<code>response</code>	A list of a list of response models; the first index runs over states; the second index runs over the independent responses in case a multivariate response is provided.
<code>transition</code>	A list of <code>transInit</code> models, ie multinomial logistic models with length the number of states.
<code>prior</code> <code>dens, trDens, init</code>	A multinomial logistic model for the initial state probabilities. See depmix-class help for details. For internal use.
<code>stationary</code>	Logical indicating whether the transitions are time-dependent or not; for internal use.
<code>ntimes</code>	A vector containing the lengths of independent time series.
<code>nstates</code>	The number of states of the model.
<code>nresp</code>	The number of independent responses.
<code>npars</code>	The total number of parameters of the model. Note: this is <i>not</i> the degrees of freedom because there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior probabilities.

Note

Models are not fitted; the return value of `depmix` is a model specification without optimized parameter values. Use the `fit` function to optimize parameters, and to specify additional constraints.

Author(s)

Ingmar Visser & Maarten Speekenbrink

References

- Ingmar Visser and Maarten Speekenbrink (2010). `depmixS4`: An R Package for Hidden Markov Models. *Journal of Statistical Software*, 36(7), p. 1-21.
- Lawrence R. Rabiner (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77-2, p. 267-295.

See Also

[fit](#), [transInit](#), [GLMresponse](#), [depmix-methods](#) for accessor functions to `depmix` objects.

For full control see the `makeDepmix` help page and its example section for the possibility to add user-defined response distributions.

Examples

```

# create a 2 state model with one continuous and one binary response
# ntimes is used to specify the lengths of 3 separate series
data(speed)
mod <- depmix(list(rt~1,corr~1),data=speed,nstates=2,
  family=list(gaussian(),multinomial("identity")),ntimes=c(168,134,137))
# print the model, formulae and parameter values
mod
set.seed(1)
# fit the model by calling fit
fm <- fit(mod)

# Volatility of S & P 500 returns
# (thanks to Chen Haibo for providing this example)

data(sp500)

# fit some models
msp <- depmix(logret~1,nstates=2,data=sp500)
set.seed(1)
fmmsp <- fit(msp)

# plot posterior state sequence for the 2-state model
plot(ts(posterior(fmmsp)[,2]), start=c(1950,2),deltat=1/12),ylab="probability",
main="Posterior probability of state 1 (volatile, negative markets).",
frame=FALSE)

## Not run:

# this creates data with a single change point with Poisson data
set.seed(3)
y1 <- rpois(50,1)
y2 <- rpois(50,2)
ydf <- data.frame(y=c(y1,y2))

# fit models with 1 to 3 states
m1 <- depmix(y~1,ns=1,family=poisson(),data=ydf)
set.seed(1)
fm1 <- fit(m1)
m2 <- depmix(y~1,ns=2,family=poisson(),data=ydf)
set.seed(1)
fm2 <- fit(m2)
m3 <- depmix(y~1,ns=3,family=poisson(),data=ydf)
set.seed(1)
fm3 <- fit(m3,em=em.control(maxit=500))

# plot the BICs to select the proper model
plot(1:3,c(BIC(fm1),BIC(fm2),BIC(fm3)),ty="b")

## End(Not run)

```

```
## Not run:
# similar to the binomial model, data may also be entered in
# multi-column format where the n for each row can be different
dt <- data.frame(y1=c(0,1,1,2,4,5),y2=c(1,0,1,0,1,0),y3=c(4,4,3,2,1,1))
# specify a mixture model ...
m2 <- mix(cbind(y1,y2,y3)~1,data=dt,ns=2,family=multinomial("identity"))
set.seed(1)
fm2 <- fit(m2)
# ... or dependent mixture model
dm2 <- depmix(cbind(y1,y2,y3)~1,data=dt,ns=2,family=multinomial("identity"))
set.seed(1)
fdm2 <- fit(dm2)

## End(Not run)
```

depmix-class

Class "depmix"

Description

A [depmix](#) model.

Slots

response: List of list of response objects.

transition List of [transInit](#) objects.

prior: [transInit](#) object.

dens: Array of dimension $\text{sum}(\text{ntimes}) * \text{nresp} * \text{nstates}$ providing the densities of the observed responses for each state.

trDens: Array of dimension $\text{sum}(\text{ntimes}) * \text{nstates}$ providing the probability of a state transition depending on the predictors.

init: Array of dimension $\text{length}(\text{ntimes}) * \text{nstates}$ with the current predictions for the initial state probabilities.

homogeneous: Logical indicating whether the transitions are time-dependent or not; for internal use.

ntimes: A vector containing the lengths of independent time series; if data is provided, $\text{sum}(\text{ntimes})$ must be equal to $\text{nrow}(\text{data})$.

nstates: The number of states of the model.

nresp: The number of independent responses.

npars: The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

Accessor Functions

The following functions should be used for accessing the corresponding slots:

npar: The number of parameters of the model.

nresp: The number of responses.

nstates: The number of states.

ntimes: The vector of independent time series lengths.

Author(s)

Ingmar Visser

depmix-methods	<i>'depmix' and 'mix' methods.</i>
----------------	------------------------------------

Description

Various methods for depmix and mix objects.

Usage

```
## S4 method for signature 'depmix'
logLik(object,method=c("fb","lystig","classification"),na.allow=TRUE)
## S4 method for signature 'mix'
logLik(object,method=c("fb","lystig","classification"),na.allow=TRUE)

## S4 method for signature 'depmix.fitted.classLik'
logLik(object,method=c("classification","fb","lystig"),na.allow=TRUE)
## S4 method for signature 'mix.fitted.classLik'
logLik(object,method=c("classification","fb","lystig"),na.allow=TRUE)

## S4 method for signature 'depmix'
nobs(object, ...)
## S4 method for signature 'mix'
nobs(object, ...)

## S4 method for signature 'depmix'
npar(object)
## S4 method for signature 'mix'
npar(object)

## S4 method for signature 'depmix'
freepars(object)
## S4 method for signature 'mix'
freepars(object)
```

```
## S4 method for signature 'depmix'
setpars(object,values, which="pars",...)
## S4 method for signature 'mix'
setpars(object,values, which="pars",...)

## S4 method for signature 'depmix'
getpars(object,which="pars",...)
## S4 method for signature 'mix'
getpars(object,which="pars",...)

## S4 method for signature 'depmix'
getmodel(object,which="response",state=1,number=1)
## S4 method for signature 'mix'
getmodel(object,which="response",state=1,number=1)
```

Arguments

object	A depmix or mix object.
values	To be used in setpars to set new parameter values; see the example.
method	The log likelihood can be computed by either the forward backward algorithm (Rabiner, 1989), or by the method of Lystig and Hughes, 2002. The former is the default and implemented in a fast C routine. The forward-backward routine also computes the state and transition smoothed probabilities, which are not directly necessary for the log likelihood. Those smoothed variables, and the forward and backward variables are accessible through the forwardbackward function. When method="classification", the classification likelihood is computed, which is the likelihood of the data assuming the state sequence is known and equal to the maximum a posteriori state sequence. The MAP state sequence is available through the viterbi function. The classification likelihood is computed by default when calling the logLik method on an a model fitted by maximising the classification likelihood.
na.allow	Allow missing observations? When set to FALSE, the logLik method will return NA in the presence of missing observations. When set to TRUE, missing values will be ignored when computing the likelihood. When observations are partly missing (when a multivariate observation has missing values on only some of its dimensionis), this may give unexpected results.
which	<p>getpars function: The default "pars" returns a vector of all parameters of a depmix object; the alternative value "fixed" returns a logical vector of the same length indicating which parameters are fixed. The setpars functions sets parameters to new values; setpars also recomputes the dens, trans and init slots of depmix objects. Note that the getpars and setpars functions for depmix objects simply call the functions of the same name for the response and transition models.</p> <p>getmodel function: possible values are "response" (the default), "prior" and "transition" to return the corresponding submodels.</p>

state	In <code>getmodel</code> this determines the submodel to be returned (together with number in the case of response models); when <code>which="transition"</code> , <code>getmodel</code> returns the transition submodel for <code>state=state</code> .
number	In <code>getmodel</code> this determines the "response" model to be returned from state <code>state</code> .
...	Not used currently.

Value

<code>logLik</code>	returns a <code>logLik</code> object with attributes <code>df</code> and <code>nobs</code> .
<code>nobs</code>	returns the number of observations (used in computing the BIC).
<code>npar</code>	returns the number of parameters of a model.
<code>freepars</code>	returns the number of non-redundant parameters.
<code>setpars</code>	returns a (dep-)mix object with new parameter values.
<code>getpars</code>	returns a vector with the current parameter values.
<code>getmodel</code>	returns a submodel of a (dep-)mix model; the prior model, one of the transition models (determined by argument <code>state</code>) or one of the response models (determined by arguments <code>state</code> and <code>number</code>).

Author(s)

Ingmar Visser

Examples

```
# create a 2 state model with one continuous and one binary response
data(speed)
mod <- depmix(list(rt~1,corr~1),data=speed,nstates=2,family=list(gaussian(),multinomial()))

getmodel(mod,"response",2,1)

getmodel(mod,"prior")

# get the loglikelihood of the model
logLik(mod)

# to see the ordering of parameters to use in setpars
mod <- setpars(mod, value=1:npar(mod))
mod

# to see which parameters are fixed (by default only baseline parameters in
# the multinomial logistic models for the transition models and the initial
# state probabilities model)
mod <- setpars(mod, getpars(mod,which="fixed"))
mod
```

depmix.fitted-class *Class "depmix.fitted" (and "depmix.fitted.classLik")*

Description

A fitted `depmix` model.

Slots

A `depmix.fitted` object is a `depmix` object with three additional slots, here is the complete list:

response: List of list of response objects.

transition List of `transInit` objects.

prior: `transInit` object.

dens: Array of dimension $\text{sum}(\text{ntimes}) * \text{nresp} * \text{nstates}$ providing the densities of the observed responses for each state.

trDens: Array of dimension $\text{sum}(\text{ntimes}) * \text{nstates}$ providing the probability of a state transition depending on the predictors.

init: Array of dimension $\text{length}(\text{ntimes}) * \text{nstates}$ with the current predictions for the initial state probabilities.

stationary: Logical indicating whether the transitions are time-dependent or not; for internal use.

ntimes: A vector containing the lengths of independent time series; if data is provided, $\text{sum}(\text{ntimes})$ must be equal to $\text{nrow}(\text{data})$.

nstates: The number of states of the model.

nresp: The number of independent responses.

nparams: The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

message: This provides some information on convergence, either from the EM algorithm or from `Rdonlp2`.

conMat: The linear constraint matrix, which has zero rows if there were no constraints.

lin.lower The lower bounds on the linear constraints.

lin.upper The upper bounds on the linear constraints.

posterior: Posterior (Viterbi) state sequence.

Details

The print function shows some convergence information, and the summary method shows the parameter estimates.

Extends

depmix.fitted extends the "depmix" class directly. depmix.fitted.classLik is similar to depmix.fitted, the only difference being that the model is fitted by maximising the classification likelihood.

Author(s)

Ingmar Visser & Maarten Speekenbrink

depmix.sim-class	Class "depmix.sim"
------------------	--------------------

Description

A depmix.sim model. The depmix.sim class directly extends the depmix class, and has an additional slot for the true states. A depmix.sim model can be generated by simulate(mod, ...), where mod is a depmix model.

Slots

response: List of list of response objects.

transition List of transInit objects.

prior: transInit object.

dens: Array of dimension sum(ntimes)*nresp*nstates providing the densities of the observed responses for each state.

trDens: Array of dimension sum(ntimes)*nstates providing the probability of a state transition depending on the predictors.

init: Array of dimension length(ntimes)*nstates with the current predictions for the initial state probabilities.

homogeneous: Logical indicating whether the transitions are time-dependent or not; for internal use.

ntimes: A vector containing the lengths of independent time series; if data is provided, sum(ntimes) must be equal to nrow(data).

nstates: The number of states of the model.

nresp: The number of independent responses.

npars: The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

states: A matrix with the true states.

Accessor Functions

The following functions should be used for accessing the corresponding slots:

`npar`: The number of parameters of the model.

`nresp`: The number of responses.

`nstates`: The number of states.

`ntimes`: The vector of independent time series lengths.

Author(s)

Maarten Speekenbrink

em.control

Control parameters for the EM algorithm

Description

Set control parameters for the EM algorithm.

Usage

```
em.control(maxit = 500, tol = 1e-08, crit = c("relative", "absolute"),
random.start = TRUE, classification = c("soft", "hard"))
```

Arguments

<code>maxit</code>	The maximum number of iterations.
<code>tol</code>	The tolerance level for convergence. See Details.
<code>crit</code>	Sets the convergence criterion to "relative" or "absolute" change of the log-likelihood. See Details.
<code>random.start</code>	This is used for a (limited) random initialization of the parameters. See Details.
<code>classification</code>	Type of classification to states used. See Details.

Details

The argument `crit` sets the convergence criterion to either the relative change in the log-likelihood or the absolute change in the log-likelihood. The relative likelihood criterion (the default) assumes convergence on iteration i when $\frac{\log L(i) - \log L(i-1)}{\log L(i-1)} < tol$. The absolute likelihood criterion assumes convergence on iteration i when $\log L(i) - \log L(i-1) < tol$. Use `crit="absolute"` to invoke the latter convergence criterion. Note that in that case, optimal values of the tolerance parameter `tol` scale with the value of the log-likelihood (and these are not changed automatically).

Argument `random.start` This is used for a (limited) random initialization of the parameters. In particular, if `random.start=TRUE`, the (posterior) state probabilities are randomized at iteration 0

(using a uniform distribution), i.e. the γ variables (Rabiner, 1989) are sampled from the Dirichlet distribution with a (currently fixed) value of $\alpha = 0.1$; this results in values for each row of γ that are quite close to zero and one; note that when these values are chosen at zero and one, the initialization is similar to that used in `kmeans`. Random initialization is useful when no initial parameters can be given to distinguish between the states. It is also useful for repeated estimation from different starting values.

Argument `classification` is used to choose either soft (default) or hard classification of observations to states. When using soft classification, observations are assigned to states with a weight equal to the posterior probability of the state. When using hard classification, observations are assigned to states according to the maximum a posteriori (MAP) states (i.e., each observation is assigned to one state, which is determined by the Viterbi algorithm in the case of `depmix` models). As a result, the EM algorithm will find a local maximum of the classification likelihood (Celeux & Govaert, 1992). Warning: hard classification is an experimental feature, especially for hidden Markov models, and its use is currently not advised.

Value

`em.control` returns a list of the control parameters.

Author(s)

Ingmar Visser & Maarten Speekenbrink

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.

Gilles Celeux and Gerard Govaert (1992). A classification EM algorithm for clustering and two stochastic versions. *Computational Statistics and Data Analysis*, 14, p. 315-332.

Examples

```
# using "hard" assignment of observations to the states, we can maximise the
# classification likelihood instead of the usual marginal likelihood
data(speed)
mod <- depmix(list(rt~1,corr~1),data=speed,nstates=2,
  family=list(gaussian(),multinomial("identity")),ntimes=c(168,134,137))
set.seed(1)
# fit the model by calling fit
fmod <- fit(mod,emcontrol=em.control(classification="hard"))
# can get rather different solutions with different starting values...
set.seed(3)
fmod2 <- fit(mod,emcontrol=em.control(classification="hard"))
```

fit

*Fit 'depmix' or 'mix' models***Description**

fit optimizes parameters of `depmix` or `mix` models, optionally subject to general linear (in)equality constraints.

Usage

```
## S4 method for signature 'mix'
fit(object, fixed=NULL, equal=NULL,
     conrows=NULL, conrows.upper=NULL, conrows.lower=NULL,
     method=NULL, verbose=TRUE,
     emcontrol=em.control(),
     solnpctrl=list(rho = 1, outer.iter = 400, inner.iter = 800,
                   delta = 1e-7, tol = 1e-8),
     donlpcntrl=donlp2Control(),
     ...)

## S4 method for signature 'mix.fitted'
summary(object,which="all")

## S4 method for signature 'depmix.fitted'
summary(object,which="all")
```

Arguments

<code>object</code>	An object of class (dep-)mix.
<code>fixed</code>	Vector of mode logical indicating which parameters should be fixed.
<code>equal</code>	Vector indicating equality constraints; see Details.
<code>conrows</code>	Rows of a general linear constraint matrix; see Details.
<code>conrows.upper</code> , <code>conrows.lower</code>	Upper and lower bounds for the linear constraints; see Details.
<code>method</code>	The optimization method; mostly determined by constraints.
<code>verbose</code>	Should optimization information be displayed on screen?
<code>emcontrol</code>	Named list with control parameters for the EM algorithm (see <code>em.control</code>).
<code>solnpctrl</code>	Control parameters passed to the 'rsolnp' optimizer; see <code>solnp</code> for explanation and defaults used there.
<code>donlpcntrl</code>	Control parameters passed to 'donlp' optimizer; see <code>?donlp2Control</code> for explanation and defaults used there; this can be used to tweak optimization but note that extra output is not returned.

which	Should summaries be provided for "all" submodels? Options are "prior", "response", and for fitted depmix models also "transition".
...	Further arguments passed on to the optimization methods.

Details

Models are fitted by the EM algorithm if there are no constraints on the parameters. Aspects of the EM algorithm can be controlled through the `emcontrol` argument; see details in [em.control](#). Otherwise the general optimizers `solnp`, the default (from package `Rsolnp`) or `donlp2` (from package `Rdonlp2`) are used which handle general linear (in-)equality constraints. These optimizers are selected by setting `method='rsolnp'` or `method='donlp'` respectively.

Three types of constraints can be specified on the parameters: fixed, equality, and general linear (in-)equality constraints. Constraint vectors should be of length `npar(object)`; note that this hence includes redundant parameters such as the base category parameter in multinomial logistic models which is always fixed at zero. See help on [getpars](#) and [setpars](#) about the ordering of parameters.

The `equal` argument is used to specify equality constraints: parameters that get the same integer number in this vector are estimated to be equal. Any integers can be used in this way except 0 and 1, which indicate fixed and free parameters, respectively.

Using `solnp` (or `donlp2`), a Newton-Raphson scheme is employed to estimate parameters subject to linear constraints by imposing:

$$bl \leq A * x \leq bu,$$

where x is the parameter vector, bl is a vector of lower bounds, bu is a vector of upper bounds, and A is the constraint matrix.

The `conrows` argument is used to specify rows of A directly, and the `conrows.lower` and `conrows.upper` arguments to specify the bounds on the constraints. `conrows` must be a matrix of `npar(object)` columns and one row for each constraint (a vector in the case of a single constraint). Examples of these three ways of constraining parameters are provided below.

Note that when specifying constraints that these should respect the fixed constraints inherent in e.g. the multinomial logit models for the initial and transition probabilities. For example, the baseline category coefficient in a multinomial logit model is fixed on zero.

[llratio](#) performs a log-likelihood ratio test on two fitted models; the first object should have the largest degrees of freedom (find out by using [freepars](#)).

Value

`fit` returns an object of class `depmix.fitted` which contains the original `depmix` object, and further has slots:

message: Convergence information.

conMat: The constraint matrix A , see Details.

posterior: The posterior state sequence (computed with the viterbi algorithm), and the posterior probabilities (delta probabilities in Rabiner, 1989, notation).

The `print` method shows the message along with the likelihood and AIC and BIC; the `summary` method prints the parameter estimates.

Posterior densities and the viterbi state sequence can be accessed through [posterior](#).

As fitted models are depmixS4 models, they can be used as starting values for new fits, for example with constraints added. Note that when refitting already fitted models, the constraints, if any, are not added automatically, they have to be added again.

Author(s)

Ingmar Visser & Maarten Speekenbrink

References

Some of the below models for the speed data are reported in:

Ingmar Visser, Maartje E. J. Raijmakers and Han L. J. van der Maas (2009). Hidden Markov Models for Individual Time Series. In: Jaan Valsiner, Peter C. M. Molenaar, M. C. D. P. Lyra, and N. Chaudhary (editors). *Dynamic Process Methodology in the Social and Developmental Sciences*, chapter 13, pages 269–289. New York: Springer.

Examples

```
data(speed)

# 2-state model on rt and corr from speed data set
# with Pacc as covariate on the transition matrix
# ntimes is used to specify the lengths of 3 separate series
mod1 <- depmix(list(rt~1,corr~1),data=speed,transition=~Pacc,nstates=2,
family=list(gaussian(),multinomial("identity")),ntimes=c(168,134,137))
# fit the model
set.seed(3)
fmod1 <- fit(mod1)
fmod1 # to see the logLik and optimization information
# to see the parameters
summary(fmod1)

# same model, now with missing data
## Not run:
speed[2,1] <- NA
speed[3,2] <- NA

# 2-state model on rt and corr from speed data set
# with Pacc as covariate on the transition matrix
# ntimes is used to specify the lengths of 3 separate series
mod1ms <- depmix(list(rt~1,corr~1),data=speed,transition=~Pacc,nstates=2,
family=list(gaussian(),multinomial("identity")),ntimes=c(168,134,137))
# fit the model
set.seed(3)
fmod1ms <- fit(mod1ms)

## End(Not run)

# instead of the normal likelihood, we can also maximise the "classification" likelihood
# this uses the maximum a posteriori state sequence to assign observations to states
# and to compute initial and transition probabilities.
```

```

fmod1b <- fit(mod1,emcontrol=em.control(classification="hard"))
fmod1b # to see the logLik and optimization information

# FIX SOME PARAMETERS

# get the starting values of this model to the optimized
# values of the previously fitted model to speed optimization

pars <- c(unlist(getpars(fmod1)))

# constrain the initial state probs to be 0 and 1
# also constrain the guessing probs to be 0.5 and 0.5
# (ie the probabilities of corr in state 1)
# change the ones that we want to constrain
pars[1]=0
pars[2]=1 # this means the process will always start in state 2
pars[13]=0.5
pars[14]=0.5 # the corr parameters in state 1 are now both 0, corresponding the 0.5 prob
mod2 <- setpars(mod1,pars)

# fix the parameters by setting:
free <- c(0,0,rep(c(0,1),4),1,1,0,0,1,1,1,1)
# fit the model
fmod2 <- fit(mod2,fixed=!free)

# likelihood ratio insignificant, hence fmod2 better than fmod1
llratio(fmod1,fmod2)

# ADDING SOME GENERAL LINEAR CONSTRAINTS

# set the starting values of this model to the optimized
# values of the previously fitted model to speed optimization

## Not run:

pars <- c(unlist(getpars(fmod2)))
pars[4] <- pars[8] <- -4
pars[6] <- pars[10] <- 10
mod3 <- setpars(mod2,pars)

# start with fixed and free parameters
conpat <- c(0,0,rep(c(0,1),4),1,1,0,0,1,1,1,1)
# constrain the beta's on the transition parameters to be equal
conpat[4] <- conpat[8] <- 2
conpat[6] <- conpat[10] <- 3

fmod3 <- fit(mod3,equal=conpat)

llratio(fmod2,fmod3)

# above constraints can also be specified using the conrows argument as follows

```

```

conr <- matrix(0,2,18)
# parameters 4 and 8 have to be equal, otherwise stated, their diffence should be zero,
# and similarly for parameters 6 & 10
conr[1,4] <- 1
conr[1,8] <- -1
conr[2,6] <- 1
conr[2,10] <- -1

# note here that we use the fitted model fmod2 as that has appropriate
# starting values
fmod3b <- fit(mod3,conrows=conr,fixed=!free) # using free defined above

## End(Not run)

data(balance)
# four binary items on the balance scale task
mod4 <- mix(list(d1~1,d2~1,d3~1,d4~1), data=balance, nstates=2,
family=list(multinomial("identity"),multinomial("identity"),
multinomial("identity"),multinomial("identity")))

set.seed(1)
fmod4 <- fit(mod4)

## Not run:

# add age as covariate on class membership by using the prior argument
mod5 <- mix(list(d1~1,d2~1,d3~1,d4~1), data=balance, nstates=2,
family=list(multinomial("identity"),multinomial("identity"),
multinomial("identity"),multinomial("identity")),
prior=~age, initdata=balance)

set.seed(1)
fmod5 <- fit(mod5)

# check the likelihood ratio; adding age significantly improves the goodness-of-fit
llratio(fmod5,fmod4)

## End(Not run)

```

forwardbackward

Forward and backward variables

Description

Compute the forward and backward variables of a depmix object.

Usage

```
## S4 method for signature 'mix'  
forwardbackward(object, return.all=TRUE, useC=TRUE, ...)
```

Arguments

object	A depmix object.
return.all	If FALSE, only gamma and xi and the log likelihood are returned (which are the only variables needed in using EM).
useC	Flag used to set whether the C-code is used to compute the forward, backward, gamma and xi variables or not; the R-code is basically obsolete (but retained for now for debugging purposes).
...	Not currently used.

Value

forwardbackward returns a list of the following (the variables are named after the notation from Rabiner, 1989):

alpha	The forward variables.
beta	The backward variables.
gamma	The smoothed transition probabilities.
xi	The smoothed state probabilities.
sca	The scale factors (called lambda in Rabiner, 1989).
logLike	The log likelihood (computed as <code>sum(log(sca))</code>).

If `return.all=FALSE`, only gamma, xi and the log likelihood are returned.

Author(s)

Maarten Speekenbrink & Ingmar Visser

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.

Examples

```
# add some later
```

GLMresponse

*Methods for creating depmix response models***Description**

Create GLMresponse objects for [depmix](#) models using formulae and family objects.

Usage

```
GLMresponse(formula, data=NULL, family=gaussian(), pstart=NULL,
fixed=NULL, prob=TRUE, ...)
```

```
## S4 method for signature 'response'
getdf(object)
```

Arguments

formula	A model formula .
data	An optional data.frame to interpret the variables from the formula argument in.
family	A family object;
pstart	Starting values for the coefficients and other parameters, e.g. the standard deviation for the gaussian() family.
fixed	Logical vector indicating which paramters are to be fixed.
prob	Logical indicating whether the starting values for multinomial() family models are probabilities or logistic parameters (see details).
object	Object of class response.
...	Not used currently.

Details

GLMresponse is the default driver for specifying response distributions of depmix models. It uses the familiar formula interface from [glm](#) to specify how responses depend on covariates/predictors.

Currently available options for the family argument are binomial, gaussian, poisson, Gamma, and multinomial. Except for the latter option, the GLMresponse model is an interface to the glm functions of which the functionality is used: predict, fit and density functions.

The multinomial model takes as link functions `mlogit`, the default, and then uses functionality from the `nnet` package to fit multinomial logistic models; using `mlogit` as link allows only `n=1` models to be specified, i.e. a single observation for each occasion; it also takes `identity` as a link function. The latter is typically faster and is hence preferred when no covariates are present.

See the [responses](#) help page for examples.

Value

GLMresponse returns an object of class GLMresponse which extends the [response-class](#).
getdf returns the number of free parameters of a response model.

Author(s)

Ingmar Visser & Maarten Speekenbrink

See Also

[makeDepmix](#) has an example of specifying a model with a multivariate normal response and an example of how to add a user-defined response model, in particular an ex-gauss distribution used for the speed data.

llratio	<i>Log likelihood ratio test on two fitted models</i>
---------	---

Description

Performs a log likelihood ratio test on two fitted depmix models.

Usage

```
llratio(basemodel, constrainedmodel, ...)
```

Arguments

basemodel	Fitted model with a logLik method.
constrainedmodel	Fitted model with a logLik method.
...	Not currently used.

Value

llratio returns an object of class llratio which has slots:

value	: Minus twice the loglikelihood difference.
df	: The degrees of freedom, ie the difference in number of freely estimated parameters between the models.

The print method shows the value, the degrees of freedom and the corresponding p-value under the chisquared distribution.

Author(s)

Ingmar Visser

makeDepmix	<i>Dependent Mixture Model Specification: full control and adding response models</i>
------------	---

Description

makeDepmix creates an object of class depmix. This function is meant for full control, e.g. specifying each response model and the transition and prior models 'by hand'. For the default easier specification of models, please see [depmix](#). This function is meant for specifying one's own response models.

Usage

```
makeDepmix(response, transition, prior, ntimes = NULL, homogeneous = TRUE,
stationary = NULL, ...)
```

Arguments

response	A two-dimensional list of response models. See 'Details'.
transition	A list of transition models, each created by a call to transInit . The length of this list should be the number of states of the model.
prior	The initial state probabilities model; created through a call to transInit .
ntimes	A vector specifying the lengths of individual, ie independent, time series. If not specified, the responses are assumed to form a single time series.
homogeneous	Logical indicating whether the transition models include time-varying covariates; used internally to determine the dimensions of certain arrays, notably trDens.
stationary	This argument should no longer be used; if not NULL, the value of stationary is copied to the homogeneous argument, with a warning. In future versions this argument may be dropped or used for different purposes, i.e., for specifying models in which the initial state probabilities are constrained to be the stationary distribution of the transition matrix.
...	Not used currently.

Details

The function makeDepmix creates an S4 object of class depmix, which needs to be fitted using [fit](#) to optimize the parameters. This function is provided to have full control, eg by specifying one's own response models with distributions that are not provided.

The response model(s) should be created by call(s) to [GLMresponse](#), [MVNresponse](#) (see example below) or user-defined response models (see example below) that should extend the [response-class](#) and have the following methods: dens, predict and optionally fit. The fit function should have an argument w, providing the weights. If the fit function is not provided, optimization should be done

by using `Rdonlp` (use `method="donlp"` in calling `fit` on the `depmix` model, note that this is *not* the default). The first index of response models runs over the states of the model, and the second index over the responses to be modeled.

Value

See the [depmix](#) help page for the return value, a `depmix` object.

Author(s)

Ingmar Visser & Maarten Speekenbrink

See Also

[fit](#), [transInit](#), [GLMresponse](#), [depmix-methods](#) for accessor functions to `depmix` objects.

Examples

```
# below example recreates the same model as on the fit help page in a roundabout way
# there we had:
# mod1 <- depmix(list(rt~1,corr~1),data=speed,transition=~Pacc,nstates=2,
# family=list(gaussian(),multinomial("identity")),ntimes=c(168,134,137))

data(speed)

rModels <- list(
  list(
    GLMresponse(formula=rt~1,data=speed,family=gaussian()),
    GLMresponse(formula=corr~1,data=speed,family=multinomial("identity"))
  ),
  list(
    GLMresponse(formula=rt~1,data=speed,family=gaussian()),
    GLMresponse(formula=corr~1,data=speed,family=multinomial("identity"))
  )
)

transition <- list()
transition[[1]] <- transInit(~Pacc,nstates=2,data=speed)
transition[[2]] <- transInit(~Pacc,nstates=2,data=speed)

inMod <- transInit(~1,ns=2,data=data.frame(rep(1,3)),family=multinomial("identity"))
mod <- makeDepmix(response=rModels,transition=transition,prior=inMod,
ntimes=c(168,134,137),homogeneous=FALSE)

set.seed(3)
fm1 <- fit(mod)
fm1
summary(fm1)

# generate data from two different multivariate normal distributions
```

```

m1 <- c(0,1)
sd1 <- matrix(c(1,0.7,.7,1),2,2)
m2 <- c(1,0)
sd2 <- matrix(c(2,.1,.1,1),2,2)
set.seed(2)
y1 <- mvrnorm(50,m1,sd1)
y2 <- mvrnorm(50,m2,sd2)
# this creates data with a single change point
y <- rbind(y1,y2)

# now use makeDepmix to create a depmix model for this bivariate normal timeseries
rModels <- list()
rModels[[1]] <- list(MVNresponse(y~1))
rModels[[2]] <- list(MVNresponse(y~1))

trstart=c(0.9,0.1,0.1,0.9)

transition <- list()
transition[[1]] <- transInit(~1,nstates=2,data=data.frame(1),pstart=c(trstart[1:2]))
transition[[2]] <- transInit(~1,nstates=2,data=data.frame(1),pstart=c(trstart[3:4]))

instart=runif(2)
inMod <- transInit(~1,ns=2,ps=instart,data=data.frame(1))

mod <- makeDepmix(response=rModels,transition=transition,prior=inMod)

fm2 <- fit(mod,emc=em.control(random=FALSE))

# where is the switch point?
plot(as.ts(posterior(fm2)[,2]))

# in below example we add the exgaus distribution as a response model and fit
# this instead of the gaussian distribution to the rt slot of the speed data
# most of the actual computations for the exgaus distribution is done by calling
# functions from the gamlss family of packages; see their help pages for
# interpretation of the mu, nu and sigma parameters that are fitted below

## Not run:
require(gamlss)
require(gamlss.dist)

data(speed)
rt <- speed$rt

# define a response class which only contains the standard slots, no additional slots
setClass("exgaus", contains="response")

# define a generic for the method defining the response class

setGeneric("exgaus", function(y, pstart = NULL, fixed = NULL, ...) standardGeneric("exgaus"))

# define the method that creates the response class

```

```

setMethod("exgaus",
  signature(y="ANY"),
  function(y,pstart=NULL,fixed=NULL, ...) {
    y <- matrix(y,length(y))
    x <- matrix(1)
    parameters <- list()
    npar <- 3
    if(is.null(fixed)) fixed <- as.logical(rep(0,npar))
    if(!is.null(pstart)) {
      if(length(pstart)!=npar) stop("length of 'pstart' must be ",npar)
      parameters$mu <- pstart[1]
      parameters$sigma <- log(pstart[2])
      parameters$nu <- log(pstart[3])
    }
    mod <- new("exgaus",parameters=parameters,fixed=fixed,x=x,y=y,npar=npar)
    mod
  }
)

setMethod("show","exgaus",
  function(object) {
    cat("Model of type exgaus (see ?gamlss for details) \n")
    cat("Parameters: \n")
    cat("mu: ", object@parameters$mu, "\n")
    cat("sigma: ", object@parameters$sigma, "\n")
    cat("nu: ", object@parameters$nu, "\n")
  }
)

setMethod("dens","exgaus",
  function(object,log=FALSE) {
    dexGAUS(object@y, mu = predict(object),
    sigma = exp(object@parameters$sigma), nu = exp(object@parameters$nu), log = log)
  }
)

setMethod("getpars","response",
  function(object,which="pars",...) {
    switch(which,
      "pars" = {
        parameters <- numeric()
        parameters <- unlist(object@parameters)
        pars <- parameters
      },
      "fixed" = {
        pars <- object@fixed
      }
    )
    return(pars)
  }
)

```

```

setMethod("setpars", "exgaus",
  function(object, values, which="pars", ...) {
    npar <- npar(object)
    if(length(values)!=npar) stop("length of 'values' must be",npar)
    # determine whether parameters or fixed constraints are being set
nms <- names(object@parameters)
switch(which,
  "pars"= {
    object@parameters$mu <- values[1]
    object@parameters$sigma <- values[2]
    object@parameters$nu <- values[3]
  },
  "fixed" = {
    object@fixed <- as.logical(values)
  }
)
  names(object@parameters) <- nms
  return(object)
)

setMethod("fit", "exgaus",
  function(object,w) {
    if(missing(w)) w <- NULL
    y <- object@y
    fit <- gamlss(y~1,weights=w,family=exGAUS(),
control=gamlss.control(n.cyc=100,trace=FALSE),
mu.start=object@parameters$mu,
sigma.start=exp(object@parameters$sigma),
nu.start=exp(object@parameters$nu))
pars <- c(fit$mu.coefficients,fit$sigma.coefficients,fit$nu.coefficients)
object <- setpars(object,pars)
object
}
)

setMethod("predict", "exgaus",
  function(object) {
    ret <- object@parameters$mu
    return(ret)
  }
)

rModels <- list(
  list(
    exgaus(rt,pstart=c(5,.1,.1)),
    GLMresponse(formula=corr~1, data=speed,
family=multinomial("identity"), pstart=c(0.5,0.5))
  ),
  list(
    exgaus(rt,pstart=c(6,.1,.1)),
    GLMresponse(formula=corr~1, data=speed,
family=multinomial("identity"), pstart=c(.1,.9))
  )
)

```

```

)
)

trstart=c(0.9,0.1,0.1,0.9)
transition <- list()
transition[[1]] <- transInit(~Pacc,nstates=2,data=speed,pstart=c(trstart[1:2],0,0))
transition[[2]] <- transInit(~Pacc,nstates=2,data=speed,pstart=c(trstart[3:4],0,0))

instart=c(0.5,0.5)
inMod <- transInit(~1,ns=2,ps=instart,family=multinomial("identity"), data=data.frame(rep(1,3)))

mod <- makeDepmix(response=rModels,transition=transition,prior=inMod,ntimes=c(168,134,137),
homogeneous=FALSE)

fm3 <- fit(mod,emc=em.control(rand=FALSE))
summary(fm3)

## End(Not run)

```

mix

Mixture Model Specification

Description

`mix` creates an object of class `mix`, an (independent) mixture model (as a limit case of dependent mixture models in which all observed time series are of length 1), otherwise known latent class or mixture model. For a short description of the package see [depmixS4](#).

Usage

```

mix(response, data=NULL, nstates, family=gaussian(),
prior=~1, initdata=NULL, respstart=NULL, instart=NULL,...)

```

Arguments

<code>response</code>	The response to be modeled; either a formula or a list of formulae in the multivariate case; this interfaces to the <code>glm</code> distributions. See 'Details'.
<code>data</code>	An optional <code>data.frame</code> to interpret the variables in the response and transition arguments.
<code>nstates</code>	The number of states of the model.
<code>family</code>	A family argument for the response. This must be a list of family's if the response is multivariate.
<code>prior</code>	A one-sided formula specifying the density for the prior or initial state probabilities.

<code>initdata</code>	An optional <code>data.frame</code> to interpret the variables occurring in prior. The number of rows of this <code>data.frame</code> must be equal to the number of cases being modeled. See 'Details'.
<code>respstart</code>	Starting values for the parameters of the response models.
<code>instart</code>	Starting values for the parameters of the prior or initial state probability model.
<code>...</code>	Not used currently.

Details

The function `mix` creates an S4 object of class `mix`, which needs to be fitted using `fit` to optimize the parameters.

The response model(s) are by default created by call(s) to `GLMresponse` using the `formula` and the `family` arguments, the latter specifying the error distribution. See `GLMresponse` for possible values of the `family` argument for `glm`-type responses (ie a subset of the `glm` family options, and the multinomial). Alternative response distributions are specified by using the `makeDepmix` function. Its help page has examples of specifying a model with a multivariate normal response, as well as an example of adding a user-defined response model, in this case for the `ex-gauss` distribution.

If `response` is a list of formulae, the response's are assumed to be independent conditional on the latent state.

The prior density is modeled as a multinomial logistic. This model is created by a call to `transInit`. Starting values may be provided by the respective arguments. The order in which parameters must be provided can be easily studied by using the `setpars` and `getpars` functions.

Linear constraints on parameters can be provided as argument to the `fit` function.

The print function prints the formulae for the response and prior models along with their parameter values.

Value

`mix` returns an object of class `mix` which has the following slots:

<code>response</code>	A list of a list of response models; the first index runs over states; the second index runs over the independent responses in case a multivariate response is provided.
<code>prior</code>	A multinomial logistic model for the initial state probabilities.
<code>dens,init</code>	See <code>mix-class</code> help for details. For internal use.
<code>ntimes</code>	A vector made by <code>rep(1, nrow(data))</code> ; for internal use only.
<code>nstates</code>	The number of states of the model.
<code>nresp</code>	The number of independent responses.
<code>npars</code>	The total number of parameters of the model. Note: this is <i>not</i> the degrees of freedom because there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior probabilities.

Author(s)

Ingmar Visser

References

A. L. McCutcheon (1987). *Latent class analysis*. Sage Publications.

See Also

[fit](#), [transInit](#), [GLMresponse](#), [depmix-methods](#) for accessor functions to depmix objects.

Examples

```
# four binary items on the balance scale task
data(balance)

# define a latent class model
instart=c(0.5,0.5)
set.seed(1)
respstart=runif(16)
# note that ntimes argument is used to make this a mixture model
mod <- mix(list(d1~1,d2~1,d3~1,d4~1), data=balance, nstates=2,
family=list(multinomial(),multinomial(),multinomial(),multinomial()),
respstart=respstart,instart=instart)
# to see the model
mod
```

mix-class

Class "mix"

Description

A [mix](#) model.

Objects from the Class

Objects can be created by calls to [mix](#).

Slots

response: List of list of response objects.

prior: [transInit](#) object; model for the prior probabilities, also unconditional probabilities

dens: Array of dimension $\text{sum}(\text{ntimes}) * \text{nresp} * \text{nstates}$ providing the densities of the observed responses for each state.

init: Array of dimension $\text{length}(\text{ntimes}) * \text{nstates}$ with the current predictions for the initial state probabilities.

nstates: The number of states (classes) of the model.

nresp: The number of independent responses.

ntimes: A vector of 1's for each case; for internal use.

npars: The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

Accessor Functions

The following functions should be used for accessing the corresponding slots:

npars: The number of parameters of the model.

nresp: The number of responses.

nstates: The number of states.

ntimes: The vector of independent time series lengths.

Author(s)

Ingmar Visser

Examples

```
showClass("mix")
```

```
mix.fitted-class      Class "mix.fitted" (and "mix.fitted.classLik")
```

Description

A fitted `mix` model.

Slots

A `mix.fitted` object is a `mix` object with three additional slots, here is the complete list:

response: List of list of response objects.

prior: `transInit` object.

dens: Array of dimension $\text{sum}(\text{ntimes}) * \text{nresp} * \text{nstates}$ providing the densities of the observed responses for each state.

init: Array of dimension $\text{length}(\text{ntimes}) * \text{nstates}$ with the current predictions for the initial state probabilities.

ntimes: A vector containing the lengths of independent time series; if data is provided, $\text{sum}(\text{ntimes})$ must be equal to $\text{nrow}(\text{data})$.

nstates: The number of states of the model.

nresp: The number of independent responses.

npars: The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

message: This provides some information on convergence, either from the EM algorithm or from `Rdonlp2`.

conMat: The linear constraint matrix, which has zero rows if there were no constraints.

lin.lower The lower bounds on the linear constraints.

lin.upper The upper bounds on the linear constraints.

posterior: Posterior (Viterbi) state sequence.

Details

The print function shows some convergence information, and the summary method shows the parameter estimates.

Extends

Class `"mix"` directly. `mix.fitted.classLik` is similar to `mix.fitted`, the only difference being that the model is fitted by maximising the classification likelihood.

Author(s)

Ingmar Visser & Maarten Speekenbrink

Examples

```
showClass("mix.fitted")
```

mix.sim-class

Class "mix.sim"

Description

A `mix.sim` model. The `mix.sim` class directly extends the `mix` class, and has an additional slot for the true states. A `mix.sim` model can be generated by `simulate(mod, ...)`, where `mod` is a `mix` model.

Slots

response: List of list of response objects.

prior: `transInit` object.

dens: Array of dimension `sum(ntimes)*nresp*nstates` providing the densities of the observed responses for each state.

init: Array of dimension `length(ntimes)*nstates` with the current predictions for the initial state probabilities.

ntimes: A vector containing the lengths of independent time series; not applicable for `mix` objects, i.e. this is a vector of 1's.

nstates: The number of states/classes of the model.

nresp: The number of independent responses.

npar: The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

states: A matrix with the true states/classes.

Accessor Functions

The following functions should be used for accessing the corresponding slots:

npar: The number of parameters of the model.

nresp: The number of responses.

nstates: The number of states.

ntimes: The vector of independent time series lengths.

Author(s)

Ingmar Visser

posterior

Posterior states/classes

Description

Return the posterior states for a fitted (dep-)mix object. In the case of a latent class or mixture model these are the class probabilities.

Usage

```
## S4 method for signature 'depmix.fitted'
posterior(object)
## S4 method for signature 'mix.fitted'
posterior(object)
```

Arguments

object A (dep-)mix object.

Value

posterior : Returns a data.frame with nstates(object) + 1 columns; the first column has the viterbi states, the other columns have the delta probabilities, see Rabiner (1989).

Author(s)

Ingmar Visser

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.

Examples

```
# add some later
```

response-class	<i>Class "response"</i>
----------------	-------------------------

Description

A generic response model for `depmix` models.

Arguments

`object` An object of class "response".

Details

This class offers a framework from which to build specific response models such as glm based responses or multinomial responses. For extensibility, objects with class response should have at least methods: `dens` to return the dens'ity of responses, and `getpars` and `setpars` methods to get and set parameters.

The `constr` slot is used for information on constraints that are inherently part of a model; the only such constraints which are currently used are 1) the sum constraint in multinomial models with identity link, and 2) a lower bound of zero of `sd` parameters in gaussian distributions. Such constraints are only used in fitting models with general optimization routines such as `Rsolnp`; the EM algorithm automagically respects the sum constraint.

`lin`: Derivative of linear constraint.

`linup`: Upper bounds for linear constraints.

`linlow`: Lower bounds for linear constraints.

`parup`: Upper bounds on parameters.

`parlow`: Lower bounds on parameters.

Slots

`parameters`: A (named) list of parameters.

`fixed`: A logical vector indicating which parameters are fixed.

`y`: A matrix with the actual response; possibly multivariate.

`x`: A design matrix; possibly only an intercept term.

`npar`: The number of parameters.

`constr`: Information on constraints.

Accessor Functions

The following functions should be used for accessing the corresponding slots:

`npar`: The number of parameters of the model.

`getdf`: The number of non-fixed parameters.

Author(s)

Ingmar Visser & Maarten Speekenbrink

response-classes *Class "GLMresponse" and class "transInit"*

Description

Specific instances of response models for `depmix` models.

Details

The `GLMresponse`-class offers an interface to the `glm` functions that are subsequently used in fitting the `depmix` model of which the response is a part.

The `transInit` is an extension of `response` that is used to model the transition matrix and the initial state probabilities by the use of a multinomial logistic model, the difference being that in fact the response is missing as the transitions between states are not observed. This class has its own fit function which is an interface to the `multinom` function in `nnet`.

Slots

Both `GLMresponse` and `transInit` contain the `response`-class. In addition to the slots of that class, these classes have the following slots:

`formula`: A formula that specifies the model.

`family`: A family object specifying the link function. See the `GLMresponse` help page for possible options.

Accessor Functions

The following functions should be used for accessing the corresponding slots:

`npar`: The number of parameters of the model.

`getdf`: The number of non-fixed parameters.

Author(s)

Ingmar Visser & Maarten Speekenbrink

responses

Response models currently implemented in depmix.

Description

Depmix contains a number of default response models. We provide a brief description of these here.

BINOMresponse

BINOMresponse is a binomial response model. It derives from the basic [GLMresponse](#) class.

y: The dependent variable can be either a binary vector, a factor, or a 2-column matrix, with successes and misses.

x: The design matrix.

Parameters: A named list with a single element “coefficients”, which contains the GLM coefficients.

GAMMAresponse

GAMMAresponse is a model for a Gamma distributed response. It extends the basic [GLMresponse](#) class directly.

y: The dependent variable.

x: The design matrix.

Parameters: A named list with a single element “coefficients”, which contains the GLM coefficients.

MULTINOMresponse

MULTINOMresponse is a model for a multinomial distributed response. It extends the basic [GLMresponse](#) class, although the functionality is somewhat different from other models that do so.

y: The dependent variable. This is a binary matrix with N rows and Y columns, where Y is the total number of categories.

x: The design matrix.

Parameters: A named list with a single element “coefficients”, which is an $n \times Y$ matrix which contains the GLM coefficients.

MVNresponse

MVNresponse is a model for a multivariate normal distributed response. See [code](#) [makeDepmix](#) for an example of how to use this and other non-glm like distributions.

y: The dependent variable. This is a matrix.

x: The design matrix.

Parameters: A named list with a elements “coefficients”, which contains the GLM coefficients, and “Sigma”, which contains the covariance matrix.

NORMresponse

NORMresponse is a model for a normal (Gaussian) distributed response. It extends the basic [GLMresponse](#) class directly.

y: The dependent variable.

x: The design matrix.

Parameters: A named list with elements “coefficients”, which contains the GLM coefficients, and “sd”, which contains the standard deviation.

POISSONresponse

POISSONresponse is a model for a Poisson distributed response. It extends the basic [GLMresponse](#) class directly.

y: The dependent variable.

x: The design matrix.

Parameters: A named list with a single element “coefficients”, which contains the GLM coefficients.

Author(s)

Maarten Speekenbrink & Ingmar Visser

Examples

```
# binomial response model
x <- rnorm(1000)
p <- plogis(x)
ss <- rbinom(1000,1,p)
mod <- GLMresponse(cbind(ss,1-ss)~x,family=binomial())
fit(mod)
glm(cbind(ss,1-ss)~x, family=binomial)

# gamma response model
x=runif(1000,1,5)
res <- rgamma(1000,x)
## note that gamma needs proper starting values which are not
## provided by depmixS4 (even with them, this may produce warnings)
mod <- GLMresponse(res~x,family=Gamma(),pst=c(0.8,1/0.8))
fit(mod)
glm(res~x,family=Gamma)

# multinomial response model
x <- sample(0:1,1000,rep=TRUE)
mod <- GLMresponse(sample(1:3,1000,rep=TRUE)~x, family=multinomial(),pstart=c(0.33,0.33,0.33,0,0,1))
mod@y <- simulate(mod)
fit(mod)
colSums(mod@y[which(x==0),])/length(which(x==0))
colSums(mod@y[which(x==1),])/length(which(x==1))
```

```

# note that the response is treated as factor here, internal representation is in
# dummy coded format:
head(mod@y)
# similar to the binomial model, data may also be entered in multi-column format
# where the n for each row can be different
dt <- data.frame(y1=c(0,1,1,2,4,5),y2=c(1,0,1,0,1,0),y3=c(4,4,3,2,1,1))
m2 <- mix(cbind(y1,y2,y3)~1,data=dt,ns=2,family=multinomial("identity"))
fm2 <- fit(m2)
summary(fm2)

# multivariate normal response model
mn <- c(1,2,3)
sig <- matrix(c(1,.5,0,.5,1,0,0,0,2),3,3)
y <- mvrnorm(1000,mn,sig)
mod <- MVNresponse(y~1)
fit(mod)
colMeans(y)
var(y)

# normal (gaussian) response model
y <- rnorm(1000)
mod <- GLMresponse(y~1)
fm <- fit(mod)
cat("Test gaussian fit: ", all.equal(getpars(fm),c(mean(y),sd(y)),check.attributes=FALSE))

# poisson response model
x <- abs(rnorm(1000,2))
res <- rpois(1000,x)
mod <- GLMresponse(res~x,family=poisson())
fit(mod)
glm(res~x, family=poisson)

# this creates data with a single change point with Poisson distributed data
set.seed(3)
y1 <- rpois(50,1)
y2 <- rpois(50,2)
ydf <- data.frame(y=c(y1,y2))

# fit models with 1 to 3 states
m1 <- depmix(y~1,ns=1,family=poisson(),data=ydf)
fm1 <- fit(m1)
m2 <- depmix(y~1,ns=2,family=poisson(),data=ydf)
fm2 <- fit(m2)
m3 <- depmix(y~1,ns=3,family=poisson(),data=ydf)
fm3 <- fit(m3,em=em.control(maxit=500))

# plot the BICs to select the proper model
plot(1:3,c(BIC(fm1),BIC(fm2),BIC(fm3)),ty="b")

```

Description

Random draws from (dep-)mix objects.

Usage

```
## S4 method for signature 'depmix'
simulate(object, nsim=1, seed=NULL, ...)

## S4 method for signature 'mix'
simulate(object, nsim=1, seed=NULL, ...)

## S4 method for signature 'response'
simulate(object, nsim=1, seed=NULL, times, ...)

## S4 method for signature 'GLMresponse'
simulate(object, nsim=1, seed=NULL, times, ...)

## S4 method for signature 'transInit'
simulate(object, nsim=1, seed=NULL, times, is.prior=FALSE, ...)
```

Arguments

<code>object</code>	Object to generate random draws. An object of class <code>mix</code> , <code>depmix</code> , <code>response</code> or <code>transInit</code>
<code>nsim</code>	The number of draws (one draw simulates a data set of the size that is defined by <code>ntimes</code>); defaults to 1.
<code>seed</code>	Set the seed.
<code>times</code>	(optional) An indicator vector indicating for which times in the complete series to generate the data. For internal use.
<code>is.prior</code>	For <code>transInit</code> objects, indicates whether it is a prior (<code>init</code>) model, or not (i.e., it is a transition model)
<code>...</code>	Not used currently.

Details

For a `depmix` model, `simulate` generates `nsim` random state sequences, each of the same length as the observation sequence in the `depmix` model (i.e., `sum(ntimes(object))`). The state sequences are then used to generate `nsim` observation sequences of the same length.

For a `mix` model, `simulate` generates `nsim` random class assignments for each case. Those assignments are then used to generate observation/response values from the appropriate distributions.

Setting the `times` option selects the time points in the total state/observation sequence (i.e., counting is continued over `ntimes`). Direct calls of `simulate` with the `times` option are not recommended.

Value

For a depmix object, a new object of class depmix.sim.

For a transInit object, a state sequence.

For a response object, an observation sequence.

Author(s)

Maarten Speekenbrink

Examples

```
y <- rnorm(1000)
respst <- c(0,1,2,1)
trst <- c(0.9,0.1,0.1,0.9)

df <- data.frame(y=y)

mod <- depmix(y~1,data=df,respst=respst,trst=trst,inst=c(0.5,0.5),nti=1000,nst=2)

mod <- simulate(mod)
```

sp500

Standard & Poor's 500 index

Description

This data set consists of (monthly) values of the S&P 500 stock exchange index. The variable of interest is the logarithm of the return values, i.e., the logarithm of the ratio of indices, in this case the closing index is used.

Usage

```
data(speed)
```

Format

A data frame with 744 observations and 6 variables.

Open Index at the start of trading.

High Highest index.

Low Lowest index.

Close Index at the close of trading.

Volume The volume of trading.

logret The log return of the closing index.

Source

Yahoo Data.

Examples

```
data(sp500)

# the data can be made with the following code (eg to include a longer or
# shorter time span)

## Not run:

require(TTR)

# load SP500 returns
Sys.setenv(tz='UTC')

sp500 <- getYahooData('^GSPC', start=19500101, end=20120228, freq='daily')
ep <- endpoints(sp500, on="months", k=1)
sp500 <- sp500[ep[2:(length(ep)-1)]]
sp500$ret <- log(sp500$Close) - lag(log(sp500$Close))
sp500 <- na.exclude(sp500)

## End(Not run)
```

speed

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 from participant A in experiment 1a from Dutilh et al (2011).

Usage

```
data(speed)
```

Format

A data frame with 439 observations on the following 4 variables.

rt a numeric vector of response times (log ms)

corr a numeric vector of accuracy scores (0/1)

Pacc a numeric vector of the pay-off for accuracy

prev a numeric vector of accuracy scores (0/1) on the previous trial

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.

Corresponding author: g.dutilh@uva.nl

Examples

```
data(speed)
## maybe str(speed) ; plot(speed) ...
```

transInit

Methods for creating depmix transition and initial probability models

Description

Create transInit objects for `depmix` models using formulae and family objects.

Usage

```
transInit(formula, nstates, data=NULL, family=multinomial(),
pstart=NULL, fixed=NULL, prob=TRUE, ...)
```

```
## S4 method for signature 'transInit'
getdf(object)
```

Arguments

formula	A model formula .
data	An optional data.frame to interpret the variables from the formula argument in.
family	A family object; see details.
pstart	Starting values for the coefficients.
fixed	Logical vector indicating which parameters are to be fixed.

prob	Logical indicating whether the starting values for multinomial() family models are probabilities or logistic parameters (see details).
nstates	The number of states of the model.
object	An object of class transInit.
...	Not used currently.

Details

The transInit model provides functionality for the multinomial probabilities of the transitions between states, as well as for the prior or initial probabilities. These probabilities may depend on (time-varying) covariates. The model can be used with link functions `mlogit` and `identity`; the latter is the default when no covariates are. With the `mlogit` link function, the transition probabilities are modeled as baseline logistic multinomials (see Agresti, 2002, p. 272 ff.).

Start values for the parameters may be provided using the `pstart` argument; these can be provided as probabilities, the default option, or as baseline logistic parameters, use the `prob` argument to specify the chosen option. The default baseline category is set to 1, which can be modified through calling, say, `family=multinomial(base=2)`.

Note that the transInit model extends the [response-class](#), but that it actually lacks a response, i.e. the `y`-slot is empty, at the time of construction, as the transitions are not observed.

`getdf` returns the number of free parameters of a transInit model.

Value

transInit return objects of class transInit; this class extends the [response-class](#).

Author(s)

Ingmar Visser & Maarten Speekenbrink

References

Agresti, A. (2002). *Categorical Data Analysis*. Wiley series in probability and mathematical statistics. Wiley-Interscience, Hoboken, NJ, 2 edition.

Index

*Topic **classes**

- depmix-class, 8
- depmix.fitted-class, 12
- depmix.sim-class, 13
- mix-class, 31
- mix.fitted-class, 32
- mix.sim-class, 33
- response-class, 35

*Topic **datasets**

- balance, 3
- sp500, 41
- speed, 42

*Topic **methods**

- depmix, 4
- depmix-methods, 9
- em.control, 14
- fit, 16
- forwardbackward, 20
- GLMresponse, 22
- llratio, 23
- makeDepmix, 24
- mix, 29
- posterior, 34
- response-classes, 36
- simulate, 40
- transInit, 43

*Topic **models**

- responses, 37

*Topic **package**

- depmixS4-package, 2

balance, 3

BINOMresponse (responses), 37

depmix, 2, 3, 4, 8, 12, 13, 16, 22, 24, 25, 35, 36, 43

depmix, ANY-method (depmix), 4

depmix-class, 8

depmix-methods, 9

depmix.fit (fit), 16

depmix.fitted, 17

depmix.fitted (depmix.fitted-class), 12

depmix.fitted-class, 12

depmix.fitted.classLik
(depmix.fitted-class), 12

depmix.fitted.classLik-class
(depmix.fitted-class), 12

depmix.sim (depmix.sim-class), 13

depmix.sim-class, 13

depmixS4, 4, 29

depmixS4 (depmixS4-package), 2

depmixS4-package, 2

em.control, 14, 16, 17

fit, 2, 3, 5, 6, 16, 24, 25, 30, 31

fit, depmix-method (fit), 16

fit, mix-method (fit), 16

formula, 22, 43

forwardbackward, 10, 20

forwardbackward, depmix-method
(forwardbackward), 20

forwardbackward, mix-method
(forwardbackward), 20

freepars, 17

freepars (depmix-methods), 9

freepars, depmix-method
(depmix-methods), 9

freepars, depmix.fitted-method
(depmix-methods), 9

freepars, mix-method (depmix-methods), 9

freepars, mix.fitted-method
(depmix-methods), 9

GAMMAresponse (responses), 37

getdf (GLMresponse), 22

getdf, MULTINOMresponse-method
(GLMresponse), 22

getdf, response-method (GLMresponse), 22

getdf, transInit-method (transInit), 43

- getmodel (depmix-methods), 9
- getmodel, depmix-method (depmix-methods), 9
- getmodel, mix-method (depmix-methods), 9
- getpars, 5, 17, 30
- getpars (depmix-methods), 9
- getpars, depmix-method (depmix-methods), 9
- getpars, mix-method (depmix-methods), 9
- glm, 2, 22, 36
- GLMresponse, 5, 6, 22, 24, 25, 30, 31, 36–38
- GLMresponse, formula-method (GLMresponse), 22
- GLMresponse-class (response-classes), 36
- llratio, 17, 23
- logLik (depmix-methods), 9
- logLik, depmix-method (depmix-methods), 9
- logLik, depmix.fitted.classLik-method (depmix-methods), 9
- logLik, mix-method (depmix-methods), 9
- logLik, mix.fitted.classLik-method (depmix-methods), 9
- loglikelihoodratio (llratio), 23
- makeDepmix, 3, 5, 6, 23, 24, 30, 37
- makeMix (makeDepmix), 24
- mix, 2, 16, 29, 31–33
- mix, ANY-method (mix), 29
- mix-class, 31
- mix.fit (fit), 16
- mix.fitted-class, 32
- mix.fitted.classLik-class (mix.fitted-class), 32
- mix.sim (mix.sim-class), 33
- mix.sim-class, 33
- MULTINOMresponse (responses), 37
- MVNresponse (responses), 37
- MVNresponse-class (response-classes), 36
- nnet, 36
- nobs (depmix-methods), 9
- nobs, depmix-method (depmix-methods), 9
- nobs, mix-method (depmix-methods), 9
- NORMresponse (responses), 37
- npar (depmix-methods), 9
- npar, depmix-method (depmix-methods), 9
- npar, mix-method (depmix-methods), 9
- nresp (depmix-class), 8
- nresp, depmix-method (depmix-class), 8
- nresp, depmix.sim-method (depmix.sim-class), 13
- nresp, mix-method (mix-class), 31
- nresp, mix.sim-method (mix.sim-class), 33
- nstates (depmix-class), 8
- nstates, depmix-method (depmix-class), 8
- nstates, depmix.sim-method (depmix.sim-class), 13
- nstates, mix-method (mix-class), 31
- nstates, mix.sim-method (mix.sim-class), 33
- ntimes (depmix-class), 8
- ntimes, depmix-method (depmix-class), 8
- ntimes, depmix.sim-method (depmix.sim-class), 13
- ntimes, mix-method (mix-class), 31
- ntimes, mix.sim-method (mix.sim-class), 33
- POISSONresponse (responses), 37
- posterior, 17, 34
- posterior, depmix.fitted-method (posterior), 34
- posterior, mix.fitted-method (posterior), 34
- response-class, 35
- response-classes, 36
- responses, 22, 37
- setpars, 5, 17, 30
- setpars (depmix-methods), 9
- setpars, depmix-method (depmix-methods), 9
- setpars, mix-method (depmix-methods), 9
- show (depmix), 4
- show, depmix-method (depmix), 4
- show, depmix.fitted-method (fit), 16
- show, GLMresponse-method (GLMresponse), 22
- show, llratio-method (llratio), 23
- show, mix-method (mix), 29
- show, mix.fitted-method (fit), 16
- show, MVNresponse-method (responses), 37
- simulate, 39
- simulate, BINOMresponse-method (simulate), 40
- simulate, depmix-method (simulate), 40

- simulate, GAMMAresponse-method
 - (simulate), [40](#)
- simulate, GLMresponse-method (simulate), [40](#)
- simulate, mix-method (simulate), [40](#)
- simulate, MULTINOMresponse-method
 - (simulate), [40](#)
- simulate, MVNresponse-method (simulate), [40](#)
- simulate, NORMresponse-method
 - (simulate), [40](#)
- simulate, POISSONresponse-method
 - (simulate), [40](#)
- simulate, response-method (simulate), [40](#)
- simulate, transInit-method (simulate), [40](#)
- solnp, [16](#)
- sp500, [41](#)
- speed, [42](#)
- summary (depmix), [4](#)
- summary, depmix-method (depmix), [4](#)
- summary, depmix.fitted-method (fit), [16](#)
- summary, mix-method (mix), [29](#)
- summary, mix.fitted-method (fit), [16](#)

- transInit, [5](#), [6](#), [8](#), [13](#), [24](#), [25](#), [30](#), [31](#), [33](#), [43](#)
- transInit, formula-method (transInit), [43](#)
- transInit-class (response-classes), [36](#)

- viterbi, [10](#)