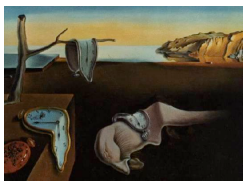

Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

depmixS4 : An R-package for hidden Markov models

Ingmar Visser
University of Amsterdam

Maarten Speekenbrink
University College London

Abstract

depmixS4 implements a general framework for defining and estimating dependent mixture models in the R programming language (R Development Core Team 2009). This includes standard Markov models, latent/hidden Markov models, and latent class and finite mixture distribution models. The models can be fitted on mixed multivariate data with distributions from the `glm` family, the logistic multinomial, or the multivariate normal distribution. Other distributions can be added easily, and an example is provided with the exgaus distribution. Parameters are estimated by the EM algorithm or, when (linear) constraints are imposed on the parameters, by direct numerical optimization with the **Rdonlp2** routine.

Keywords: hidden Markov model, dependent mixture model, mixture model.

1. Introduction

Markov and latent Markov models are frequently used in the social sciences, in different areas and applications. In psychology, they are used for modelling learning processes (see Wickens 1982, for an overview, and e.g. Schmittmann, Visser, and Raijmakers 2006, for a recent application). In economics, latent Markov models are so-called regime switching models (see e.g., Kim 1994 and Ghysels 1994). Further applications include speech recognition (Rabiner 1989), EEG analysis (Rainer and Miller 2000), and genetics (Krogh 1998). In these latter areas of application, latent Markov models are usually referred to as hidden Markov models. See for example Frühwirth-Schnatter (2006) for an overview of hidden Markov models with extensions. Further examples of applications can be found in e.g. Cappe, Moulines, and Ryden (2005, chapter 1).

The **depmixS4** package was motivated by the fact that while Markov models are used com-

monly in the social sciences, no comprehensive package was available for fitting such models. Existing software for estimating Markovian models include Panmark (Van de Pol, Langeheine, and Jong 1996), and for latent class models Latent Gold (Vermunt and Magidson 2003). These programs lack a number of important features, besides not being freely available. There are currently some packages in R that handle hidden Markov models but they lack a number of features that we needed in our research. In particular, **depmixS4** was designed to meet the following goals:

1. to be able to estimate parameters subject to general linear (in)equality constraints
2. to be able to fit transition models with covariates, i.e., to have time-dependent transition matrices
3. to be able to include covariates in the prior or initial state probabilities
4. to be easily extensible, in particular, to allow users to easily add new uni- or multivariate response distributions and new transition models, e.g., continuous time observation models

Although **depmixS4** was designed to deal with longitudinal or time series data, for say $T > 100$, it can also handle the limit case when $T = 1$. In this case, there are no time dependencies between observed data and the model reduces to a finite mixture or latent class model. While there are specialized packages to deal with mixture data, as far as we know these don't allow the inclusion of covariates on the prior probabilities of class membership. The possibility to estimate the effects of covariates on prior and transition probabilities is a distinguishing feature of **depmixS4**. In the next section, we provide an outline of the model and likelihood equations.

2. The dependent mixture model

The data considered here have the general form $\mathbf{O}_{1:T} = (O_1^1, \ldots, O_1^m, O_2^1, \ldots, O_2^m, \ldots, O_T^1, \ldots, O_T^m)$ for an m-variate time series of length T. As an example, consider a time series of responses generated by a single participant in a psychological response time experiment. The data consists of three variables, response time, response accuracy, and a covariate which is a pay-off variable reflecting the relative reward for speeded and/or accurate responding. These variables are measured on 168, 134 and 137 occasions respectively (the first part of this series is plotted in Figure 1). These data are more fully described in Dutilh, Visser, Wagenmakers, and Van der Maas (2009).

The latent Markov model is usually associated with data of this type, in particular for multinomially distributed responses. However, commonly employed estimation procedures (e.g., Van de Pol *et al.* 1996), are not suitable for long time series due to underflow problems. In contrast, the hidden Markov model is typically only used for 'long' univariate time series (Cappe *et al.* 2005, chapter 1). We use the term "dependent mixture model" because one of the authors (Ingmar Visser) thought it was time for a new name to relate these models[1].

[1]Only later did I find out that Leroux and Puterman (1992) already coined the term dependent mixture models in an application with hidden Markov mixtures of Poisson count data.

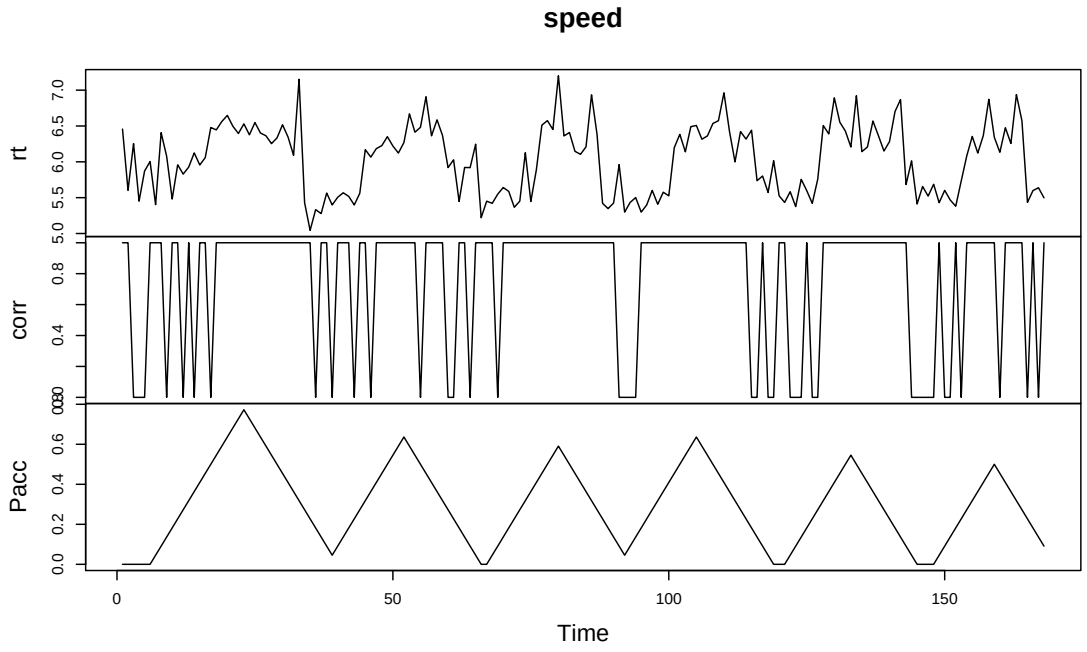

Figure 1: Response times (rt), accuracy (corr) and pay-off values (Pacc) for the first series of responses in dataset `speed`.

The fundamental assumption of a dependent mixture model is that at any time point, the observations are distributed as a mixture with n components (or states), and that time-dependencies between the observations are due to time-dependencies between the mixture components (i.e., transition probabilities between the components). These latter dependencies are assumed to follow a first-order Markov process. In the models we are considering here, the mixture distributions, the initial mixture probabilities and transition probabilities can all depend on covariates $\mathbf{z}_t$.

In a dependent mixture model, the joint likelihood of observations $\mathbf{O}_{1:T}$ and latent states $\mathbf{S}_{1:T} = (S_1, \ldots, S_T)$, given model parameters $\boldsymbol{\theta}$ and covariates $\mathbf{z}_{1:T} = (\mathbf{z}_1, \ldots, \mathbf{z}_T)$, can be written as:

$$\mathsf{P}(\mathbf{O}_{1:T}, \mathbf{S}_{1:T}|\boldsymbol{\theta}, \mathbf{z}_{1:T}) = \pi_i(\mathbf{z}_1)\mathbf{b}_{S_t}(\mathbf{O}_1|\mathbf{z}_1) \prod_{t=1}^{T-1} a_{ij}(\mathbf{z}_t)\mathbf{b}_{S_t}(\mathbf{O}_{t+1}|\mathbf{z}_{t+1}), \tag{1}$$

where we have the following elements:

1. S_t is an element of $\mathcal{S} = \{1 \ldots n\}$, a set of n latent classes or states.
2. $\pi_i(\mathbf{z}_1) = \mathsf{P}(S_1 = i|\mathbf{z}_1)$, giving the probability of class/state i at time $t = 1$ with covariate $\mathbf{z}_1$.
3. $a_{ij}(\mathbf{z}_t) = \mathsf{P}(S_{t+1} = j|S_t = i, \mathbf{z}_t)$, provides the probability of a transition from state i to state j with covariate $\mathbf{z}_t$,

4. $\mathbf{b}_{S_t}$ is a vector of observation densities $b_j^k(\mathbf{z}_t) = \mathsf{P}(O_t^k|S_t = j, \mathbf{z}_t)$ that provide the conditional densities of observations O_t^k associated with latent class/state j and covariate $\mathbf{z}_t$, $j = 1, \ldots, n$, $k = 1, \ldots, m$.

For the example data above, b_j^k could be a Gaussian distribution function for the response time variable, and a Bernoulli distribution for the accuracy variable. In the models we are considering here, both the transition probability functions a_{ij} and the initial state probability functions $\boldsymbol{\pi}$ may depend on covariates as well as the response distributions b_j^k.

2.1. Likelihood

To obtain maximum likelihood estimates of the model parameters, we need the marginal likelihood of the observations. For hidden Markov models, this marginal (log-)likelihood is usually computed by the so-called forward-backward algorithm (Baum and Petrie 1966; Rabiner 1989), or rather by the forward part of this algorithm. Lystig and Hughes (2002) changed the forward algorithm in such a way as to allow computing the gradients of the log-likelihood at the same time. They start by rewriting the likelihood as follows (for ease of exposition the dependence on the model parameters and covariates is dropped here):

$$L_T = \mathsf{P}(\mathbf{O}_{1:T}) = \prod_{t=1}^{T} \mathsf{P}(\mathbf{O}_t|\mathbf{O}_{1:(t-1)}), \tag{2}$$

where $\mathsf{P}(\mathbf{O}_1|\mathbf{O}_0) := \mathsf{P}(\mathbf{O}_1)$. Note that for a simple, i.e. observed, Markov chain these probabilities reduce to $\mathsf{P}(\mathbf{O}_t|\mathbf{O}_1, \ldots, \mathbf{O}_{t-1}) = \mathsf{P}(\mathbf{O}_t|\mathbf{O}_{t-1})$. The log-likelihood can now be expressed as:

$$l_T = \sum_{t=1}^{T} \log[\mathsf{P}(\mathbf{O}_t|\mathbf{O}_{1:(t-1)}]. \tag{3}$$

To compute the log-likelihood, Lystig and Hughes (2002) define the following (forward) recursion:

$$\phi_1(j) := \mathsf{P}(\mathbf{O}_1, S_1 = j) = \pi_j b_j(\mathbf{O}_1) \tag{4}$$

$$\phi_t(j) := \mathsf{P}(\mathbf{O}_t, S_t = j|\mathbf{O}_{1:(t-1)}) = \sum_{i=1}^{N} [\phi_{t-1}(i) a_{ij} b_j(\mathbf{O}_t)] \times (\Phi_{t-1})^{-1}, \tag{5}$$

where $\Phi_t = \sum_{i=1}^{N} \phi_t(i)$. Combining $\Phi_t = \mathsf{P}(\mathbf{O}_t|\mathbf{O}_{1:(t-1)})$, and equation (3) gives the following expression for the log-likelihood:

$$l_T = \sum_{t=1}^{T} \log \Phi_t. \tag{6}$$

2.2. Parameter estimation

Parameters are estimated in **depmixS4** using the EM algorithm or through the use of a general Newton-Raphson optimizer. In the EM algorithm, parameters are estimated by iteratively maximising the expected joint likelihood of the parameters given the observations and states. Let $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3)$ be the general parameter vector consisting of three subvectors with

parameters for the prior model, transition model, and response model respectively. The joint log-likelihood can be written as

$$\log \mathsf{P}(\mathbf{O}_{1:T}, \mathbf{S}_{1:T}|\mathbf{z}_{1:T}, \boldsymbol{\theta}) = \log \mathsf{P}(S_1|\mathbf{z}_1, \boldsymbol{\theta}_1) + \sum_{t=2}^{T} \log \mathsf{P}(S_t|S_{t-1}, \mathbf{z}_{t-1}, \boldsymbol{\theta}_2) + \sum_{t=1}^{T} \log \mathsf{P}(\mathbf{O}_t|S_t, \mathbf{z}_t, \boldsymbol{\theta}_3) \tag{7}$$

This likelihood depends on the unobserved states $\mathbf{S}_{1:T}$. In the Expectation step, we replace these with their expected values given a set of (initial) parameters $\boldsymbol{\theta}' = (\boldsymbol{\theta}'_1, \boldsymbol{\theta}'_2, \boldsymbol{\theta}'_3)$ and observations $\mathbf{O}_{1:T}$. The expected log likelihood

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}') = E_{\boldsymbol{\theta}'}(\log \mathsf{P}(\mathbf{O}_{1:T}, \mathbf{S}_{1:T}|\mathbf{O}_{1:T}, \mathbf{z}_{1:T}, \boldsymbol{\theta})) \tag{8}$$

can be written as

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}') = & \sum_{j=1}^{n} \gamma_1(j) \log \mathsf{P}(S_1 = j|\mathbf{z}_1, \boldsymbol{\theta}_1) \\ & + \sum_{t=2}^{T} \sum_{j=1}^{n} \sum_{k=1}^{n} \xi_t(j,k) \log \mathsf{P}(S_t = k|S_{t-1} = j, \mathbf{z}_{t-1}, \boldsymbol{\theta}_2) \\ & + \sum_{t=1}^{T} \sum_{j=1}^{n} \sum_{k=1}^{m} \gamma_t(j) \ln \mathsf{P}(O_t^k|S_t = j, \mathbf{z}_t, \boldsymbol{\theta}_3), \end{aligned} \tag{9}$$

where the expected values $\xi_t(j,k) = P(S_t = k, S_{t-1} = j|\mathbf{O}_{1:T}, \mathbf{z}_{1:T}, \boldsymbol{\theta}')$ and $\gamma_t(j) = P(S_t = j|\mathbf{O}_{1:T}, \mathbf{z}_{1:T}, \boldsymbol{\theta}')$ can be computed effectively by the forward-backward algorithm (see e.g., Rabiner 1989). The Maximisation step consists of the maximisation of (9) for $\boldsymbol{\theta}$. As the right hand side of (9) consists of three separate parts, we can maximise separately for $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$ and $\boldsymbol{\theta}_3$. In common models, maximisation for $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ is performed by the `nnet.default` routine in the **nnet** package (Venables and Ripley 2002), and maximisation for $\boldsymbol{\theta}_3$ by the standard `glm` routine. Note that for the latter maximisation, the expected values $\gamma_t(j)$ are used as prior weights of the observations O_t^k.

The EM algorithm however has some drawbacks. First, it can be slow to converge towards the end of optimization. Second, applying constraints to parameters can be problematic; in particular, EM can lead to wrong parameter estimates when applying constraints. Hence, in **depmixS4**, EM is used by default in unconstrained models, but otherwise, direct optimization is done using **Rdonlp2** (Tamura 2009; Spellucci 2002), because it handles general linear (in)equality constraints, and optionally also non-linear constraints.

3. Using depmixS4

Two steps are involved in using **depmixS4** which are illustrated below with examples:

1. model specification with function `depmix` (or with `mix` for latent class and finite mixture models, see example below on adding covariates to prior probabilities)

2. model fitting with function `fit`

We have separated the stages of model specification and model fitting because fitting large models can be fairly time-consuming and it is hence useful to be able to check the model specification before actually fitting the model.

3.1. Example data: speed

Throughout this article a data set called `speed` is used. As already indicated in the Introduction, it consists of three time series with three variables: response time, accuracy, and a covariate Pacc which defines the relative pay-off for speeded versus accurate responding. The participant in this experiment switches between fast responding at chance level and relatively slower responding at a high level of accuracy. Interesting hypotheses to test are: is the switching regime symmetric? Is there evidence for two states or does one state suffice? Is the guessing state actually a guessing state, i.e., is the probability of a correct response at chance level (0.5)?

3.2. A simple model

A dependent mixture model is defined by the number of states and the initial state, state transition, and response distribution functions. A dependent mixture model can be created with the `depmix`-function as follows:

```
> set.seed(1)
> mod <- depmix(rt~1, data=speed, nstates=2, trstart=runif(4))
```

The `depmix` function returns an object of class `depmix` which contains the model specification (and not a fitted model!). Note also that start values for the transition parameters are provided in this call using the `trstart` argument. At this time, the package does not provide automatic starting values.

The so-defined model needs to be `fit`ted with the following line of code:

```
> fm <- fit(mod)
```

The `fit`-function returns an object of class `depmix.fitted` which extends the `depmix` class, adding convergence information (and information about constraints if these were applied, see below). The `print` method provides summary information on convergence, the log-likelihood and the AIC and BIC values:

```
> fm

Convergence info: Log likelihood converged to within tol.
'log Lik.' -84.34 (df=7)
AIC:  182.68
BIC:  211.28
```

These statistics can also be extracted using `logLik`, `AIC` and `BIC`, respectively. By comparison, a 1-state model for these data, i.e. assuming there is no mixture, has a log-likelihood of -305.33, and 614.66, and 622.83 for the AIC and BIC respectively. Hence, the 2-state model

fits the data much better than the 1-state model. Note that the 1-state model can be specified using `mod <- depmix(rt~1, data=speed, nstates=1)`, although this model is trivial as it will simply return the mean and the sd of the rt variable.

The `summary` method of `fit`ted models provides the parameter estimates, first for the prior probabilities model, second for the transition model, and third for the response models.

```
> summary(fm)

Initial state probabilties model
Model of type multinomial, formula: ~1
Coefficients:
     [,1]      [,2]
[1,]    0 -11.25
Probalities at zero values of the covariates.
0.999 1.292e-05

Transition model for state (component) 1
Model of type multinomial, formula: ~1
Coefficients:
[1]  0.000 -2.392
Probalities at zero values of the covariates.
0.9163 0.0837

Transition model for state (component) 2
Model of type multinomial, formula: ~1
Coefficients:
[1] 0.000 2.139
Probalities at zero values of the covariates.
0.105 0.895

Response model(s) for state 1

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 6.385
sd  0.244

Response model(s) for state 2

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 5.511
sd  0.193
```

Note that, since no further arguments were specified, the initial state, state transition and

response distributions were set to their defaults (multinomial distributions for the first two, and Gaussian distributions for the response distributions).

3.3. Covariates on transition parameters

By default, the transition probabilities and the initial state probabilities are parameterized using the multinomial logistic model. More precisely, each row of the transition matrix is parameterized by a baseline category logistic multinomial, meaning that the parameter for the base category is fixed at zero (see Agresti 2002, p. 267 ff., for multinomial logistic models and various parameterizations). The default baseline category is the first state. Hence, for example, for a 3-state model, the initial state probability model would have three parameters of which the first is fixed at zero and the other two are freely estimated.

The multinomial logistic model allows us to include covariates on the initial state and transition probabilities. Chung, Walls, and Park (2007) discuss a related latent transition model for repeated measurement data ($T = 2$) using logistic regression on the transition parameters; they rely on Bayesian methods of estimation. Covariates on the transition probabilities can be specified using a one-sided formula as in the following example:

```
> set.seed(1)
> mod <- depmix(rt~1, data=speed, nstates=2, family=gaussian(),
    transition=~scale(Pacc), instart=runif(2))
> fm <- fit(mod)
```

Applying `summary` to the `fit`ted object gives (only transition models printed here):

```
...
Transition model for state (component) 1
Model of type multinomial, formula: ~scale(Pacc)
Coefficients:
     [,1]       [,2]
[1,]    0 -0.9215
[2,]    0  1.865
Probalities at zero values of the covariates.
0.7154 0.2846

Transition model for state (component) 2
Model of type multinomial, formula: ~scale(Pacc)
Coefficients:
     [,1]     [,2]
[1,]    0 2.471
[2,]    0 3.571
Probalities at zero values of the covariates.
0.0779 0.9221
...
```

The summary provides all parameters of the model, also the (redundant) zeroes for the baseline category in the multinomial model. The summary also prints the transition probabilities

at the zero value of the covariate. Note that scaling of the covariate is useful in this regard as it makes interpretation of these intercept probabilities easier.

3.4. Multivariate data

Multivariate data can be modelled by providing a list of formulae as well as a list of family objects for the distributions of the various responses. In above examples we have only used the response times which were modelled with the Gaussian distribution. The accuracy variable in the `speed` data can be modelled with a multinomial by specifying the following:

```
> set.seed(1)
> mod <- depmix(list(rt~1,corr~1), data=speed, nstates=2,
    family=list(gaussian(),multinomial()),
    transition=~scale(Pacc),instart=runif(2))
> fm <- fit(mod)
```

This provides the following fitted model parameters (only the response parameters are given here):

```
> summary(fm)


...
Response model(s) for state 1

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 5.522
sd  0.2029

Response model for response 2
Model of type multinomial, formula: corr ~ 1
Coefficients:
     [,1]      [,2]
[1,]    0 0.1031
Probalities at zero values of the covariates.
0.4743 0.5257

Response model(s) for state 2

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 6.394
sd  0.2374

Response model for response 2
```

```
Model of type multinomial, formula: corr ~ 1
Coefficients:
           [,1]     [,2]
[1,]    0 2.2455
Probalities at zero values of the covariates.
0.0957 0.9043
```

As can be seen, state 1 has fast response times and accuracy is approximately at chance level (.474), whereas state 2 corresponds with slower responding at higher accuracy levels (.904).

Note that by specifying multivariate observations in terms of a list, the variables are considered conditionally independent (given the states). Conditionally *dependent* variables must be handled as a single element in the list. Effectively, this means specifying a multivariate response model. The only multivariate response model currently implemented in **depmixS4** is for multivariate normal variables.

3.5. Adding covariates on the prior probabilities

To illustrate the use of covariates on the prior probabilities we have included another data set with **depmixS4**. The `balance` data consists of 4 binary items (correct-incorrect) on a balance scale task Siegler (1981). The data form a subset of the data published in Jansen and van der Maas (2002).

Similarly to the transition matrix, covariates on the prior probabilities of the latent states (or classes in this case), are defined by using a one-sided formula:

```
> balance$age <- balance$age-5
> set.seed(1)
> mod <- mix(list(d1~1,d2~1,d3~1,d4~1), data=balance, nstates=2,
    family=list(multinomial(), multinomial(), multinomial(),
    multinomial()), respstart=c(rep(c(0.9,0.1),4),rep(c(0.1,0.9),4)),
    prior=~age, initdata=balance)
> fm <- fit(mod)
```

Note here that we define a `mix` model instead of a `depmix` models as these data form independent observations. More formally, `depmix` models extend the class of `mix` models by adding the transition models. As for fitting `mix` models: as can be seen in equation 9, the EM algorithm can be applied by simply dropping the second summand containing the transition parameters, and this is implemented as such in the EM algorithms in **depmixS4**.

The summary of the `fit`ted model gives the following (only the prior model is shown here):

```
> summary(fm)

Mixture probabilities model
Model of type multinomial, formula: ~age
     [,1]       [,2]
[1,]    0 -2.518
[2,]    0  0.551
Probalities at zero values of the covariates.
```

```
0.9254 0.0746
...
```

Hence at young ages, children have a high probability of incorrect responding in class 1, whereas the prior probability for class 2 increases with age.

3.6. Fixing and constraining parameters

Using package **Rdonlp2** by Tamura (2009), parameters may be fitted subject to general linear (in-)equality constraints. Constraining and fixing parameters is done using the `conpat` argument to the `fit`-function, which specifies for each parameter in the model whether it's fixed (0) or free (1 or higher). Equality constraints can be imposed by having two parameters have the same number in the `conpat` vector. When only fixed values are required, the `fixed` argument can be used instead of `conpat`, with zeroes for fixed parameters and other values (ones e.g.) for non-fixed parameters. Fitting the models subject to these constraints is handled by the optimization routine `donlp2`. To be able to construct the `conpat` and/or `fixed` vectors one needs the correct ordering of parameters which is briefly discussed next before proceeding with an example.

Parameter numbering When using the `conpat` and `fixed` arguments, complete parameter vectors should be supplied, i.e., these vectors should have length of the number of parameters of the model, which can be obtained by calling `npar(object)`. Note that this is not the same as the degrees of freedom used e.g. in the `logLik` function because `npar` also counts the baseline category zeroes from the multinomial logistic models. Parameters are numbered in the following order:

1. the prior model parameters
2. the parameters for the transition models
3. the response model parameters per state (and subsequently per response in the case of multivariate time series)

To see the ordering of parameters use the following:

```
> setpars(mod, value=1:npar(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):

```
> setpars(mod, getpars(mod,which="fixed"))
```

When fitting constraints it is useful to have good starting values for the parameters and hence we first fit the following model without constraints:

```
> trst <- c(0.9,0.1,0,0,0.1,0.9,0,0)
> mod <- depmix(list(rt~1,corr~1),data=speed,transition=~Pacc,
          nstates=2,family=list(gaussian(),multinomial()),
          trstart=trst,inst=c(.999,0.001))
> fm1 <- fit(mod)
```

After this, we use the fitted values from this model to constrain the regression coefficients on the transition matrix (parameters numbers 6 and 10):

```
# start with fixed and free parameters
> conpat <- c(0,1,rep(c(0,1),4),1,1,0,1,1,1,0,1)
# constrain the beta's on the transition parameters to be equal
> conpat[6] <- conpat[10] <- 2
> fm2 <- fit(fm,equal=conpat)
```

Using `summary` on the fitted model shows that the regression coefficients are now estimated at the same value of 12.66. The function `llratio` computes the likelihood ratio χ^2-statistic and the associated p-value with appropriate degrees of freedom for testing the tenability of constraints (Dannemann and Holzmann 2007). Note that these arguments provide the possibility for arbitrary constraints, also between, e.g., a multinomial regression coefficient for the transition matrix and the mean of a Gaussian response model. Whether such constraints make sense is hence the responsibility of the user.

4. Extending depmixS4

The **depmixS4** package was designed with the aim of making it relatively easy to add new response distributions (as well as possibly new prior and transition models). To make this possible, the EM routine simply calls the `fit` methods of the separate response models without needing access to the internal workings of these routines. Referring to equation 9, the EM algorithm calls separate fit functions for each part of the model, the prior probability model, the transition models, and the response models. As a consequence, adding user-specified response models is straightforward.

User-defined distributions should extend the `response`-class and have the following slots:

1. y: the response variable
2. x: the design matrix, possibly only an intercept
3. paramaters: a named list with the coefficients and possibly other parameters, e.g., the standard deviation in the Gaussian response model
4. fixed: a vector of logicals indicating whether parameters are fixed
5. npar: numerical indicating the number of parameters of the model

In the `speed` data example, it may be more appropriate to model the response times with an exgaus rather than a Gaussian distribution. To do so, we first define an exgaus-class extending the response-class:

```
setClass("exgaus", contains="response")
```

The so-defined class now needs a number of methods:

1. constructor: function to create instances of the class with starting values

2. show method: to print the model to the terminal
3. dens: the function that computes the density of the responses
4. getpars and setpars: to get and set parameters
5. predict: to generate predict'ed values
6. fit: function to fit the model using posterior weights (used by the EM algorithm)

Only the constructor and the fit methods are provided here; the complete code can be found in the help file of the `makeDepmix` function. The `exgaus` example uses the **gamlss** and **gamlss.distr** packages (Stasinopoulos and Rigby 2009a,b) for computing the `dens`ity and for `fit`ting the parameters.

The constructor method return an object of class `exgaus`, and is defined as follows:

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

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
  }
)
```

The `fit` method is defined as follows:

```
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
  }
)
```

The `fit` method defines a `gamlss` model with only an intercept to be estimated and then sets the fitted parameters back into their respective slots in the 'exgaus' object. See the help for `gamlss.distr` for interpretation of these parameters.

After defining all the necessary methods for the new response model, we can now define the dependent mixture model using this reponse model. The function `makeDepmix` is included in **depmixS4** to have full control over model specification, and we need it here.

We first create all the response models that we need as a double list:

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

Next, we define the transition and prior probability models using the `transInit` function (which produces a transInit model, which also extends the response class):

```
trstart=c(0.9,0.1,0.1,0.9)
transition <- list()
transition[[1]] <- transInit(~Pacc,nst=2,data=speed,pstart=c(0.9,0.1,0,0))
transition[[2]] <- transInit(~Pacc,nst=2,data=speed,pstart=c(0.1,0.9,0,0))
inMod <- transInit(~1,ns=2,pstart=c(0.1,0.9),data=data.frame(1))
```

Finally, we put everything together using `makeDepmix` and fit the model:

```
> mod <- makeDepmix(response=rModels,transition=transition,
    prior=inMod,stat=FALSE)
> fm <- fit(mod)
```

Using `summary` will print the fitted parameters. Note that the use of `makeDepmix` allows the possibility of, say, fitting a Gaussian in one state and an exgaus distribution in another state. Note also that according to the AIC and BIC, the model with the exgaus describes the data much better than the same model in which the response times are modeled as gaussian.

5. Conclusions & future work

depmixS4 provides a flexible framework for fitting dependent mixture models for a large variety of response distributions. It can also fit latent class regression and finite mixture

models, although it should be noted that more specialized packages are available for this such as **FlexMix** (Leisch 2004). The package is intended for modeling of (individual) time series data with the aim of characterizing the transition processes underlying the data. The possibility to use covariates on the transition matrix greatly enhances the flexibility of the model. The EM algorithm uses a very general interface that allows easy addition of new response models.

We are currently working on implementing the gradients for response and transition models with two goals in mind. First, to speed up (constrained) parameter optimization using **Rdonlp2**. Second, analytic gradients are useful in computing the Hessian of the estimated parameters so as to arrive at standard errors for those. We are also planning to implement goodness-of-fit statistics (Titman and Sharples 2008), and automatic generation of starting values.

Acknowledgements

Ingmar Visser was supported by an EC Framework 6 grant, project 516542 (NEST). Maarten Speekenbrink was supported by the ESRC Centre for Economic Learning and Social Evolution (ELSE). Han van der Maas provided the speed-accuracy data Dutilh *et al.* (2009) and thereby neccessitated implementing models with time-dependent covariates. Brenda Jansen provided the balance scale data set (Jansen and van der Maas 2002) which was the perfect opportunity to test the covariates on the prior model parameters. The examples in the help files use both of these data sets.

References

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

Baum LE, Petrie T (1966). "Statistical inference for probabilistic functions of finite state Markov Chains." *Annals of Mathematical Statistics*, **67**, 1554–40.

Cappe O, Moulines E, Ryden T (2005). *Inference in Hidden Markov Models*. Springer Series in Statistics. Springer, New York.

Chung H, Walls T, Park Y (2007). "A Latent Transition Model With Logistic Regression." *Psychometrika*, **72**(3), 413–435. URL http://ideas.repec.org/a/spr/psycho/v72y2007i3p413-435.html.

Dannemann JRN, Holzmann H (2007). "Likelihood Ratio Testing for Hidden Markov Models Under Non-standard Conditions." *Scandinavian Journal of Statistics*, **25**(2), 309–321.

Dutilh G, Visser I, Wagenmakers EJ, Van der Maas HLJ (2009). "A Phase Transition Model for the Speed-Accuracy Trade-Off in Response Time Experiments." *Manuscript in preparation*.

Frühwirth-Schnatter S (2006). *Finite Mixture and Markov Switching Models*. Springer Series in Statistics. Springer.

Ghysels E (1994). "On the Periodic Structure of the Business Cycle." *Journal of Business and Economic Statistics*, **12**(3), 289–298.

Jansen BRJ, van der Maas HLJ (2002). "The development of children's rule use on the balance scale task." *Journal of Experimental Child Psychology*, **81**(4), 383–416.

Kim CJ (1994). "Dynamic linear models with Markov-switching." *Journal of Econometrics*, **60**, 1–22.

Krogh A (1998). "An introduction to hidden Markov models for biological sequences." In SL Salzberg, DB Searls, S Kasif (eds.), "Computational methods in molecular biology," chapter 4, pp. 45–63. Elsevier, Amsterdam.

Leisch F (2004). "FlexMix: A general framework for finite mixture models and latent class regression in R." *Journal of Statistical Software*, **11**(8), 1–18. URL http://www.jstatsoft.org/v11/i08/.

Leroux BG, Puterman ML (1992). "Maximum-Penalized-Likelihood Estimation for Independent and Markov-Dependent Mixture Models." *Biometrics*, **48**, 545–548.

Lystig TC, Hughes JP (2002). "Exact computation of the observed information matrix for hidden Markov models." *Journal of Computational and Graphical Statistics*.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Rabiner LR (1989). "A tutorial on hidden Markov models and selected applications in speech recognition." *Proceedings of IEEE*, **77**(2), 267–295.

Rainer G, Miller EK (2000). "Neural ensemble states in prefrontal cortex identified using a hidden Markov model with a modified EM algorithm." *Neurocomputing*, **32–33**, 961–966.

Schmittmann VD, Visser I, Raijmakers MEJ (2006). "Multiple learning modes in the development of rule-based category-learning task performance." *Neuropsychologia*, **44**(11), 2079–2091.

Siegler RS (1981). *Developmental sequences within and between concepts*. Number 46 in Monographs of the Society for Research in Child Development. SRCD.

Spellucci P (2002). "DONLP2." URL http://www.netlib.org/ampl/solvers/donlp2/.

Stasinopoulos M, Rigby B (2009a). *gamlss: Generalized Additive Models for Location Scale and Shape*. R package version 2.0-0, with contributions from Calliope Akantziliotou., URL http://CRAN.R-project.org/package=gamlss.

Stasinopoulos M, Rigby B (2009b). *gamlss.dist: Extra distributions to be used for GAMLSS modelling*. R package version 2.0-0, with contributions from Calliope Akantziliotou and Raydonal Ospina and Nicoletta Motpan., URL http://CRAN.R-project.org/package=gamlss.dist.

Tamura R (2009). *Rdonlp2: an R extension library to use Peter Spelluci's DONLP2 from R*. R package version 0.4, URL http://arumat.net/Rdonlp2/.

Titman AC, Sharples LD (2008). "A general goodness-of-fit test for Markov and hidden Markov models." *STATISTICS IN MEDICINE*, **27**, 2177–2195.

Van de Pol F, Langeheine R, Jong WD (1996). *PANMARK 3. Panel analysis using Markov chains. A latent class analysis program [User manual].* Voorburg: The Netherlands.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S.* Springer, New York, fourth edition. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Vermunt JK, Magidson J (2003). *Latent Gold 3.0 [Computer program and User's Guide].* Belmont (MA), USA.

Wickens TD (1982). *Models for Behavior: Stochastic processes in psychology.* W. H. Freeman and Company, San Francisco.

Affiliation:

Ingmar Visser
Department of Psychology
University of Amsterdam
Roetersstraat 15
1018 WB, Amsterdam
The Netherlands
E-mail: i.visser@uva.nl
URL: http://www.ingmar.org/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd