depmixS4: An R Package for Hidden Markov Models

Ingmar Visser
University of Amsterdam

Maarten Speekenbrink
University College London

Abstract

This introduction to the R package **depmixS4** is a (slightly) modified version of **?**, published in the *Journal of Statistical Software*. Please refer to that article when using **depmixS4**. The current version is 1.1-1; the version history and changes can be found in the NEWS file of the package. Below, the major versions are listed along with the most noteworthy changes.

depmixS4 implements a general framework for defining and estimating dependent mixture models in the R programming language. 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 expectation-maximization (EM) algorithm or, when (linear) constraints are imposed on the parameters, by direct numerical optimization with the **Rsolnp** or **Rdonlp2** routines.

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

Version history

1.1-0 Speed improvements due to writing the main loop in C code.

1.0-0 First release with this vignette, a modified version of the paper in the Journal of Statistical Software.

0.1-0 First version on CRAN.

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 **?**, for an overview, and e.g., **?**, for a recent application. In economics, latent Markov models are so-called regime switching models (see e.g., **?** and **?**). Further applications include speech recognition (**?**), EEG analysis (**?**), and genetics (**?**). In these latter areas of application, latent Markov models are usually referred to as hidden Markov models. See for example **?** for an overview of hidden Markov models with extensions. Further examples of applications can be

found in e.g., **?**, Chapter 1. A more gentle introduction into hidden Markov models with applications is the book by **?**.

The **depmixS4** package was motivated by the fact that while Markov models are used commonly in the social sciences, no comprehensive package was available for fitting such models. Existing software for estimating Markovian models include **Panmark** (**?**), and for latent class models **Latent Gold** (**?**). 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 Section 2, we provide an outline of the model and likelihood equations.

The **depmixS4** package is implemented using the R system for statistical computing (**?**) and is available from the Comprehensive R Archive Network at http://CRAN.R-project.org/package=depmixS4.

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. In the following, we use $\mathbf{O}_t$ as shorthand for $O_t^1, \ldots, O_t^m$. 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 series of 168 trials is plotted in Figure 1). These data are more fully described in **?**, and in the next section a number of example models for these data is described.

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., **?**), are not suitable for long time series due to underflow problems. In contrast, the hidden

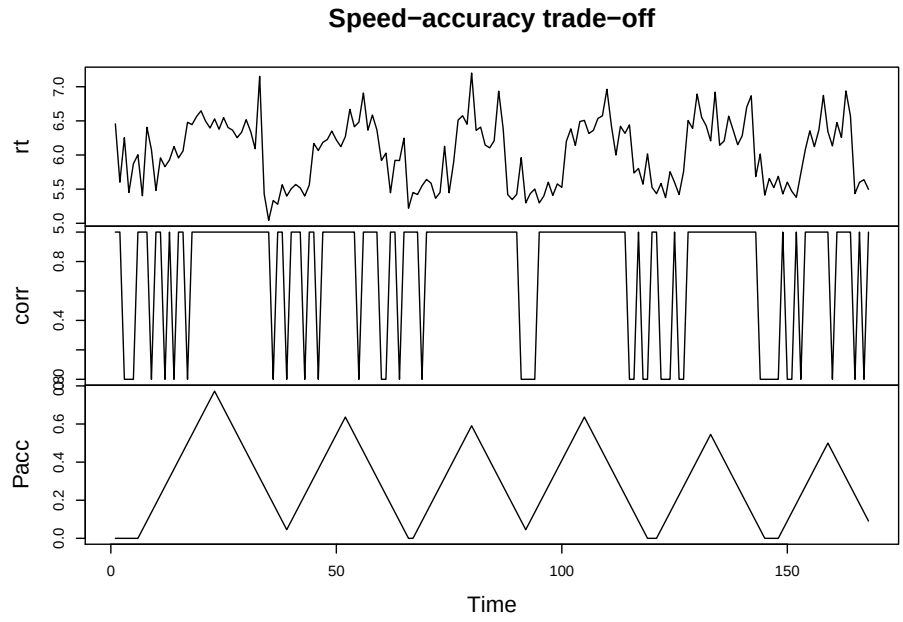

Figure 1: Response times (rt), accuracy (corr) and pay-off values (Pacc) for the first series of responses in dataset `speed`.

Markov model is typically only used for 'long' univariate time series (**?**, 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].

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:

$$\mathsf{P}(\mathbf{O}_{1:T}, \mathbf{S}_{1:T} | \boldsymbol{\theta}, \mathbf{z}_{1:T}) = \pi_i(\mathbf{z}_1)\mathbf{b}_{S_1}(\mathbf{O}_1|\mathbf{z}_1) \prod_{t=1}^{T-1} a_{ij}(\mathbf{z}_t)\mathbf{b}_{S_t}(\mathbf{O}_{t+1}|\mathbf{z}_{t+1}), \tag{1}$$

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.

[1] Only later he found out that **?** already coined the term dependent mixture models in an application with hidden Markov mixtures of Poisson count data.

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 considered 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 (**??**), or rather by the forward part of this algorithm. **?** 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, **?** define the following (forward) recursion:

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

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

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 expectation-maximization (EM) algorithm or through the use of a general Newton-Raphson optimizer. In the EM algorithm, parameters

are estimated by iteratively maximizing the expected joint log-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 models respectively. The joint log-likelihood can be written as:

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

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:

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

can be written as:

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

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., **?**). The Maximization step consists of the maximization of (9) for $\boldsymbol{\theta}$. As the right hand side of (9) consists of three separate parts, we can maximize separately for $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$ and $\boldsymbol{\theta}_3$. In common models, maximization for $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ is performed by the `nnet.default` routine in the **nnet** package (**?**), and maximization for $\boldsymbol{\theta}_3$ by the standard `glm` routine. Note that for the latter maximization, 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 used. Two options are available for direct optimization using package **Rsolnp** (**??**), or **Rdonlp2** (**??**). Both packages can handle general linear (in)equality constraints (and optionally also non-linear constraints).

2.3. Missing data

Missing data can be dealt with straightforwardly in computing the likelihood using the forward recursion in Equations (4–5). Assume we have observed $\mathbf{O}_{1:(t-1)}$ but that observation $\mathbf{O}_t$ is

missing. The key idea that, in this case, the filtering distribution, the probabilities ϕ_t, should be identical to the state prediction distribution, as there is no additional information to estimate the current state. Thus, the forward variables ϕ_t are now computed as:

$$\phi_t(i) = \mathsf{P}(S_t = i|\mathbf{O}_{1:(t-1)}) \tag{10}$$

$$= \sum_{j=1}^{N} \phi_{t-1}(j)\mathsf{P}(S_t = i|S_{t-1} = j). \tag{11}$$

For later observations, we can then use this latter equation again, realizing that the filtering distribution is technically e.g. $\mathsf{P}(S_{t+1}|\mathbf{O}_{1:(t-1),t+1})$. Computationally, the easiest way to implement this is to simply set $\mathbf{b}(\mathbf{O}_t|S_t) = 1$ if $\mathbf{O}_t$ is missing.

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 `rt`, accuracy `corr`, and a covariate, `Pacc`, which defines the relative pay-off for speeded versus accurate responding. Before describing some of the models that are fitted to these data, we provide a brief sketch of the reasons for gathering these data in the first place.

Response times are a very common dependent variable in psychological experiments and hence form the basis for inference about many psychological processes. A potential threat to such inference based on response times is formed by the speed-accuracy trade-off: different participants in an experiment may respond differently to typical instructions to 'respond as fast and accurate as possible'. A popular model which takes the speed-accuracy trade-off into account is the diffusion model (**?**), which has proven to provide accurate descriptions of response times in many different settings.

One potential problem with the diffusion model is that it predicts a continuous trade-off between speed and accuracy of responding, i.e., when participants are pressed to respond faster and faster, the diffusion model predicts that this would lead to a gradual decrease in accuracy. The `speed` data set that we analyze below was gathered to test this hypothesis versus the alternative hypothesis stating that there is a sudden transition from slow and accurate responding to fast responding at chance level. At each trial of the experiment, the participant is shown the current setting of the relative reward for speed versus accuracy. The

bottom panel of Figure 1 shows the values of this variable. The experiment was designed to investigate what would happen when this reward variable changes from reward for accuracy only to reward for speed only. The `speed` data that we analyse here are from participant A in Experiment 1 in **?**, who provide a complete description of the experiment and the relevant theoretical background.

The central question regarding this data is whether it is indeed best described by two modes of responding rather than a single mode of responding with a continuous trade-off between speed and accuracy. The hallmark of a discontinuity between slow versus speeded responding is that switching between the two modes is asymmetric (see e.g. **?**, for a theoretical underpinning of this claim). The `fit` help page of **depmixS4** provides a number of examples in which the asymmetry of the switching process is tested; those examples and other candidate models are discussed at length in **?**.

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:

```
R> library("depmixS4")
R> data("speed")
R> set.seed(1)
R> mod <- depmix(response = rt ~ 1, data = speed, nstates = 2,
+    trstart = runif(4))
```

The first line of code loads the **depmixS4** package and `data(speed)` loads the `speed` data set. The line `set.seed(1)` is necessary to get starting values that will result in the right model, see more on starting values below.

The call to `depmix` specifies the model with a number of arguments. The `response` argument is used to specify the response variable, possibly with covariates, in the familiar format using formulae such as in `lm` or `glm` models. The second argument, `data = speed`, provides the `data.frame` in which the variables from `response` are interpreted. Third, the number of states of the model is given by `nstates = 2`.

Starting values. Note also that start values for the transition parameters are provided in this call by the `trstart` argument. Starting values for parameters can be provided using three arguments: `instart` for the parameters relating to the prior probabilities, `trstart`, relating the transition models, and `respstart` for the parameters of the response models. Note that the starting values for the initial and transition models as well as multinomial logit response models are interpreted as *probabilities*, and internally converted to multinomial logit parameters (if a logit link function is used). Start values can also be generated randomly within the EM algorithm by generating random uniform values for the values of γ_t in (9) at iteration 0. Automatic generation of starting values is not available for constrained models (see below). In the call to `fit` below, the argument `emc=em.control(rand=FALSE)` controls the EM algorithm and specifies that random start values should not be generated[2].

[2] As of version 1.0-1, the default is have random parameter initialization when using the EM algorithm.

Fitting the model and printing results. The `depmix` function returns an object of class '`depmix`' which contains the model specification, and not a fitted model. Hence, the model needs to be fitted by calling `fit`:

```
R> fm <- fit(mod, emc=em.control(rand=FALSE))

iteration 0 logLik: -305.3
iteration 5 logLik: -305.3
iteration 10 logLik: -305.3
iteration 15 logLik: -305.3
iteration 20 logLik: -305.3
iteration 25 logLik: -305.3
iteration 30 logLik: -305.3
iteration 35 logLik: -305.3
iteration 40 logLik: -305.1
iteration 45 logLik: -304.5
iteration 50 logLik: -276.7
iteration 55 logLik: -89.83
iteration 60 logLik: -88.73
iteration 65 logLik: -88.73
iteration 68 logLik: -88.73
```

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 class has `print` and `summary` methods to see the results. The `print` method provides information on convergence, the log-likelihood and the AIC and BIC values:

```
R> fm

Convergence info: Log likelihood converged to within tol. (relative change)
'log Lik.' -88.73 (df=7)
AIC:  191.5
BIC:  220.1
```

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 standard deviation 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 models, and third for the response models.

```
R> summary(fm)

Initial state probabilties model
Model of type multinomial (identity), formula: ~1
```

```
<environment: 0x103782268>
Coefficients:
     [,1]      [,2]
[1,]    1 3.257e-61

Transition model for state (component) 1
Model of type multinomial (identity), formula: ~1
<environment: 0x104292390>
Coefficients:
[1] 0.91567 0.08433

Transition model for state (component) 2
Model of type multinomial (identity), formula: ~1
<environment: 0x104292390>
Coefficients:
[1] 0.1165 0.8835


Response model(s) for state 1

Response model for response 1
Model of type gaussian (identity), formula: rt ~ 1
Coefficients:
[1] 6.385
sd  0.2442


Response model(s) for state 2

Response model for response 1
Model of type gaussian (identity), formula: rt ~ 1
Coefficients:
[1] 5.51
sd  0.1918
```

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 a Gaussian distribution for the response). The resulting model indicates two well-separated states, one with slow and the second with fast responses. The transition probabilities indicate rather stable states, i.e., the probability of remaining in either of the states is around 0.9. The initial state probability estimates indicate that state 1 is the starting state for the process, with a negligible probability of starting in state 2.

3.3. Covariates on transition parameters

By default, the transition probabilities and the initial state probabilities are parameterized using a multinomial model with an identity link function. Using a multinomial logistic model

allows one to include covariates on the initial state and transition probabilities. In this case, 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 **?**, 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. **?** 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:

```
R> set.seed(1)
R> mod <- depmix(rt ~ 1, data = speed, nstates = 2, family = gaussian(),
+    transition = ~ scale(Pacc), instart = runif(2))
R> fm <- fit(mod, verbose = FALSE, emc=em.control(rand=FALSE))

iteration 42 logLik: -44.2
```

Note the use of `verbose = FALSE` to suppress printing of information on the iterations of the fitting process. Applying `summary` to the `fit`ted object gives (only transition models printed here by using argument `which`):

```
R> summary(fm, which = "transition")

Transition model for state (component) 1
Model of type multinomial (mlogit), formula: ~scale(Pacc)
Coefficients:
     [,1]    [,2]
[1,]    0 -0.9518
[2,]    0  1.3924
Probalities at zero values of the covariates.
0.7215 0.2785

Transition model for state (component) 2
Model of type multinomial (mlogit), formula: ~scale(Pacc)
Coefficients:
     [,1]  [,2]
[1,]    0 2.472
[2,]    0 3.581
Probalities at zero values of the covariates.
0.07788 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 as a Gaussian distribution. The accuracy variable in the `speed` data can be modelled with a multinomial by specifying the following:

```
R> set.seed(1)
R> mod <- depmix(list(rt ~ 1,corr ~ 1), data = speed, nstates = 2,
+    family = list(gaussian(), multinomial("identity")),
+    transition = ~ scale(Pacc), instart = runif(2))
R> fm <- fit(mod, verbose = FALSE, emc=em.control(rand=FALSE))

iteration 31 logLik: -255.5
```

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

```
R> summary(fm, which = "response")

Response model(s) for state 1

Response model for response 1
Model of type gaussian (identity), formula: rt ~ 1
Coefficients:
[1] 5.517
sd  0.1974

Response model for response 2
Model of type multinomial (identity), formula: corr ~ 1
Coefficients:
       [,1]   [,2]
[1,] 0.4747 0.5253


Response model(s) for state 2

Response model for response 1
Model of type gaussian (identity), formula: rt ~ 1
Coefficients:
[1] 6.391
sd  0.2386

Response model for response 2
Model of type multinomial (identity), formula: corr ~ 1
Coefficients:
       [,1]   [,2]
[1,] 0.0979 0.9021
```

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. Currently, **depmixS4** has one multivariate response model which is for multivariate normal variables.

3.5. Fixing and constraining parameters

Using package **Rsolnp** (**?**) or **Rdonlp2** (**?**), 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 giving two parameters 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 (e.g., ones) for non-fixed parameters. Fitting the models subject to these constraints is handled by the optimization routine `solnp` or, optionally, by `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 equal to 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:

```
R> 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):

```
R> 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:

```
R> trst <- c(0.9, 0.1, 0, 0, 0.1, 0.9, 0, 0)
R> mod <- depmix(list(rt ~ 1,corr ~ 1), data = speed, transition = ~ Pacc,
+    nstates = 2, family = list(gaussian(), multinomial("identity")),
+    trstart = trst, instart = c(0.99, 0.01))
R> fm1 <- fit(mod,verbose = FALSE, emc=em.control(rand=FALSE))

iteration 23 logLik: -255.5
```

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

```
R> pars <- c(unlist(getpars(fm1)))
R> pars[6] <- pars[10] <- 11
R> pars[1] <- 0
R> pars[2] <- 1
R> pars[13] <- pars[14] <- 0.5
R> fm1 <- setpars(mod, pars)
R> conpat <- c(0, 0, rep(c(0, 1), 4), 1, 1, 0, 0, 1, 1, 1, 1)
R> conpat[6] <- conpat[10] <- 2
R> fm2 <- fit(fm1, equal = conpat)
```

Using `summary` on the fitted model shows that the regression coefficients are now estimated at the same value of 12.66. Setting elements 13 and 14 of `conpat` to zero resulted in fixing those parameters at their starting values of 0.5. This means that state 1 can now be interpreted as a 'guessing' state which is associated with comparatively fast responses. Similarly for elements 1 and 2, resulting in fixed initial probabilities. 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 (**?**). Note that these arguments (i.e., `conpat` and `conrows`) 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.

3.6. 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 (**?**). The data form a subset of the data published in **?**. Before specifying specifying a model for these data, we briefly describe them.

The balance scale task is a famous task for testing cognitive strategies developed by Jean Piaget (see **?**). Figure 2 provides an example of a balance scale item. Participants' task is to say to which side the balance will tip when released, or alternatively, whether it will stay in balance. The item shown in Figure 2 is a so-called distance item: the number of weights placed on each side is equal, and only the distance of the weights to the fulcrum differs between each side.

Children in the lower grades of primary school are known to ignore the distance dimension, and base their answer only on the number of weights on each side. Hence, they would typically provide the wrong answer to these distance items. Slightly older children do take distance

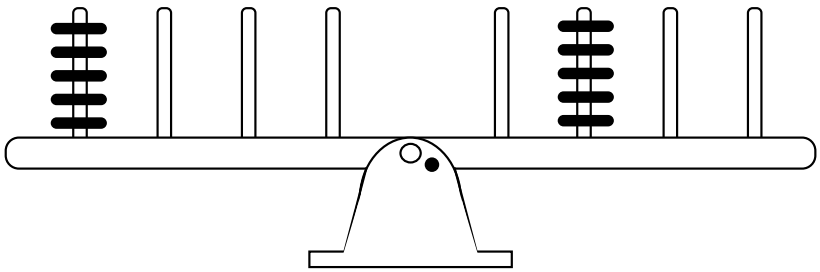

Figure 2: Balance scale item; this is a distance item (see the text for details).

into account when responding to balance scale items, but they only do so when the number of weights is equal on each side. These two strategies that children employ are known as Rule I and Rule II. Other strategies can be teased apart by administering different items. The `balance` data set that we analyse here consists of 4 distance items on a balance scale task administered to 779 participants ranging from 5 to 19 years of age. The full set of items consisted of 25 items; other items in the test are used to detect other strategies that children and young adults employ in solving balance scale items (see **?**, for details).

In the following model, age is included as covariate on class membership to test whether, with age, children apply more complex rules in solving balance scale items. 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 `prior = ~ age`:

```
R> data("balance")
R> set.seed(1)
R> mod <- mix(list(d1 ~ 1, d2 ~ 1, d3 ~ 1, d4 ~ 1), data = balance,
+    nstates = 3, family = list(multinomial("identity"),
+    multinomial("identity"), multinomial("identity"),
+    multinomial("identity")), respstart = runif(24), prior = ~ age,
+    initdata = balance)
R> fm <- fit(mod, verbose = FALSE, emc=em.control(rand=FALSE))

iteration 77 logLik: -917.5

R> fm

Convergence info: Log likelihood converged to within tol. (relative change)
'log Lik.' -917.5 (df=16)
AIC:  1867
BIC:  1942
```

Note here that we define a `mix` model instead of a `depmix` model as these data form independent observations. More formally, `depmix` models extend the class of '`mix`' models by adding 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**.

As can be seen from the print of the fitted model above, the BIC for this model equals 1941.6. The similarly defined 2-class model for these data has a BIC of 1969.2, and the 4-class model has BIC equal to 1950.4. Hence, the 3-class seems to be adequate for describing these data.

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

```
R> summary(fm, which = "prior")

Mixture probabilities model
Model of type multinomial (mlogit), formula: ~age
Coefficients:
     [,1]    [,2]    [,3]
[1,]    0  6.3957  1.7548
[2,]    0 -0.6763 -0.2905
Probalities at zero values of the covariates.
0.00165 0.9888 0.009541
```

The intercept values of the multinomial logistic parameters provide the prior probabilities of each class when the covariate has value zero (note that in this case the value zero does not have much significance, scaling and/or centering of covariates may be useful in such cases). The summary function prints these values. As can be seen from those values, at age zero, the prior probability is overwhelmingly at the second class. Inspection of the response parameters reveals that class 2 is associated with incorrect responding, whereas class 1 is associated with correct responding; class 3 is an intermediate class with guessing behavior. Figure 3 depicts the prior class probabilities as function of age based on the fitted parameters.

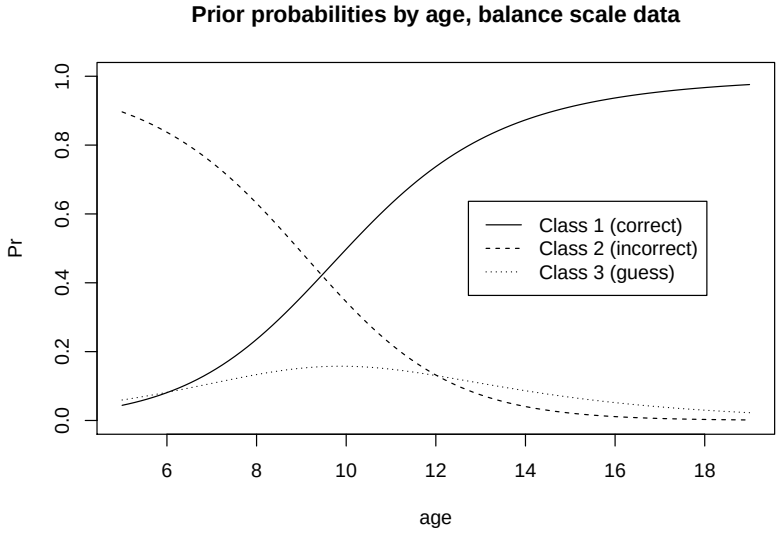

Figure 3: Class probabilities as a function of age.

As can be seen from Figure 3, at younger ages children predominantly apply Rule I, the wrong strategy for these items. According to the model, approximately 90 % of children at age 5

package	family	link
stats	binomial	logit, probit, cauchit, log, cloglog
stats	gaussian	identity, log, inverse
stats	Gamma	inverse, identity, log
stats	poisson	log, identity, sqrt
depmixS4	multinomial	logit, identity (no covariates allowed)
depmixS4	multivariate normal	identity (only available through makeDepmix)
depmixS4	ex-gauss	identity (only available through makeDepmix as example)

Table 1: Response distribution available in **depmixS4**.

apply Rule I. The remaining 10 % are evenly split among the 2 other classes. At age 19, almost all participants belong to class 1. Interestingly, prior probability of the 'guess' class 2, first increases with age, and then decreases again. This suggests that children pass through a phase in which they are uncertain or possibly switch between applying different strategies.

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. The currently implemented distributions are listed in Table 1.

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. `parameters`: A named list with the coefficients and possibly other parameters (e.g., the standard deviation in the normal 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:

```
R> 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`: 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 predicted 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 example with the exgaus distribution uses the **gamlss** and **gamlss.dist** packages (**????**) 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:

```
R> library("gamlss")
R> library("gamlss.dist")
R> setGeneric("exgaus", function(y, pstart = NULL, fixed = NULL, ...)
+    standardGeneric("exgaus"))
R> 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:

```
R> 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
+    }
+  )

[1] "fit"
```

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 response 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:

```
R> rModels <- list()
R> rModels[[1]] <- list()
R> rModels[[1]][[1]] <- exgaus(speed$rt, pstart = c(5, 0.1, 0.1))
R> rModels[[1]][[2]] <- GLMresponse(formula = corr ~ 1, data = speed,
+    family = multinomial(), pstart = c(0.5, 0.5))
R> rModels[[2]] <- list()
R> rModels[[2]][[1]] <- exgaus(speed$rt, pstart = c(6, 0.1, 0.1))
R> rModels[[2]][[2]] <- GLMresponse(formula = corr ~ 1, data = speed,
+    family = multinomial(), pstart = c(0.1, 0.9))
```

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

```
R> trstart <- c(0.9, 0.1, 0.1, 0.9)
R> transition <- list()
R> transition[[1]] <- transInit(~ Pacc, nst = 2, data = speed,
+    pstart = c(0.9, 0.1, 0, 0))
R> transition[[2]] <- transInit(~ Pacc, nst = 2, data = speed,
+    pstart = c(0.1, 0.9, 0, 0))
R> 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:

```
R> mod <- makeDepmix(response = rModels, transition = transition,
+    prior = inMod, stat = FALSE)
R> fm <- fit(mod, verbose = FALSE, emc=em.control(rand=FALSE))
```

```
iteration 43 logLik: -232.3
```

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 modelled as gaussian.

5. Conclusions and 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** (**??**). The package is intended for modelling (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** or **Rsolnp**. 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 (**?**).

Acknowledgments

Ingmar Visser was supported by an EC Framework 6 grant, project 516542 (NEST). Maarten Speekenbrink was supported by ESRC grant RES-062-23-1511 and the ESRC Centre for Economic Learning and Social Evolution (ELSE). Han van der Maas provided the speed-accuracy data (**?**) and thereby necessitated implementing models with time-dependent covariates. Brenda Jansen provided the balance scale data set (**?**) 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.

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/