depmixS4: An R-package for fitting mixture models on mixed multivariate data with Markov dependencies

Ingmar Visser & Maarten Speekenbrink*
Department of Psychology, University of Amsterdam
i.visser@uva.nl
Department of Psychology, University College London
m.speekenbrink@ucl.ac.uk

July 2, 2009

Abstract

`depmixS4` implements a general framework for defininining and fitting hidden Markov mixture model 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 multinomial and/or gaussian distributions. Parameters can be estimated subject to general linear constraints. Parameter estimation is done through an EM algorithm or by a direct optimization approach with gradients using the `Rdonlp2` optimization routine when contraints are imposed on the parameters. A number of illustrative examples are included.

*Correspondence concerning this manual should be adressed to: Ingmar Visser, Department of Psychology, University of Amsterdam, Roetersstraat 15, 1018 WB, Amsterdam, The Netherlands

Contents

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 **?** for a recent application. In economics, latent Markov models are commonly used as regime switching models, see e.g. **?** and **?**. Further applications include speech recognition (**?**), EEG analysis (**?**), and genetics (**?**). In those latter areas of application, latent Markov models are usually referred to as hidden Markov models.

The `depmixS4` package was motivated by the fact that Markov models are used commonly in the social sciences, but no comprehensive package was available for fitting such models. Common programs for Markovian models include Panmark (**?**), and for latent class models Latent Gold (**?**). Those programs are lacking 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 handle parameter estimates 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 of models
4. to allow for easy extensibility, in particular, to be able to add new response distributions, both univariate and multivariate, and similarly to be able to allow for the addition of other transition models, e.g., continuous time observation models

Although `depmixS4` is designed to deal with longitudinal or time series data, for say $T > 100$, it can also handle the limit case with $T = 1$. In those cases, there are no time dependencies between observed data, and the model reduces to a finite mixture model, or a latent class model. Although there are other specialized packages to deal with mixture data, one specific feature that we needed ourselves which is to the best of our knowledge not available in other packages is the possibility to include covariates on the prior probabilities of class membership. In the next section, an outline is provided of the model and the likelihood equations.

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 **?** and thereby neccessitated 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.

2 Dependent mixture models

The data considered here, has the general form $O_1^1, \ldots, O_1^m$, $O_2^1, \ldots, O_2^m$, $\ldots$, $O_T^1, \ldots, O_T^m$ for an m-variate time series of length T. As an example, consider a time series of responses generated by a single subject in a reaction time experiment. The data consists of three variables, reaction time, accuracy and a covariate which is a pay-off factor which determines the reward for speed and accuracy. These variables are measured on 168, 134 and 137 occasions respectively (in Figure **??** the first part of this series is plotted).

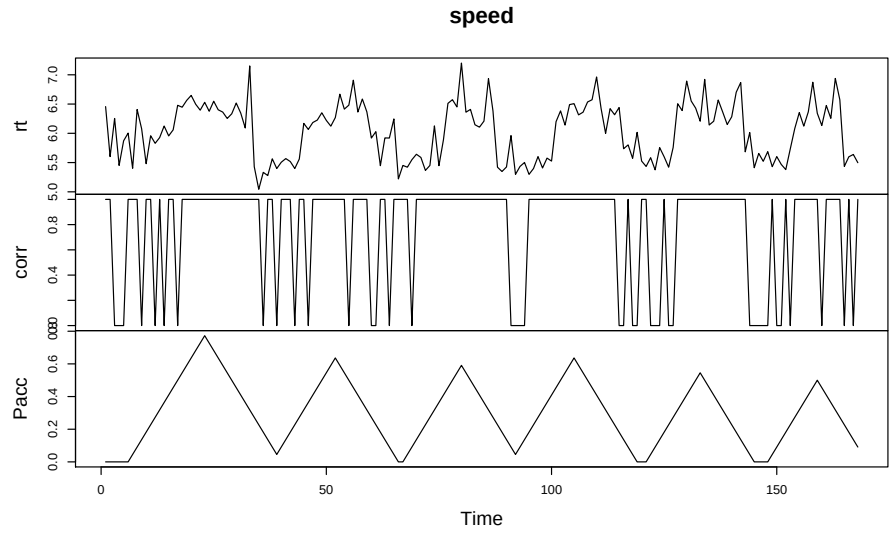

Figure 1: Reaction times, accuracy and pay-off values for the first series of responses in dataset `speed`.

The latent Markov model is commonly associated with data of this type, albeit usually only multinomial variables are considered. However, common estimation procedures, such as those implemented in **?** 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. In the next section, the likelihood and estimation procedure for the hidden Markov model is described, given data of the above form. These models are called dependent mixture models because one of the authors (Ingmar Visser) thought it was time for a new name for these models.

The dependent mixture model is defined by the following elements:

1. a set $\mathbf{S}$ of latent classes or states S_i, $i = 1, \ldots, n$,
2. matrices $\mathbf{A}_t$ of transition probabilities $a_{ij,t}$ for the transition from state S_i to state S_j at time t,
3. a set $\mathbf{B}_t$ of observation functions $b_j^k(\cdot)$ that provide the conditional probabilities of observations O_t^k associated with latent state S_j,

4. a vector $\boldsymbol{\pi}$ of latent state initial probabilities π_i

When transitions are added to the latent class model, it is more appropriate to refer to the classes as states. The word class is rather more associated with a stable trait-like attribute whereas a state can change over time.

2.1 Likelihood

The log-likelihood of hidden Markov models 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 is dropped here):

$$L_T = Pr(\mathbf{O}_1, \ldots, \mathbf{O}_T) = \prod_{t=1}^{T} Pr(\mathbf{O}_t | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1}), \tag{1}$$

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

$$l_T = \sum_{t=1}^{T} \log[Pr(\mathbf{O}_t | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1})]. \tag{2}$$

To compute the log-likelihood, **?** define the following (forward) recursion:

$$\phi_1(j) := Pr(\mathbf{O}_1, S_1 = j) = \pi_j b_j(\mathbf{O}_1) \tag{3}$$

$$\begin{aligned} \phi_t(j) &:= Pr(\mathbf{O}_t, S_t = j | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1}) \\ &= \sum_{i=1}^{N} [\phi_{t-1}(i) a_{ij} b_j(\mathbf{O}_t)] \times (\Phi_{t-1})^{-1}, \end{aligned} \tag{4}$$

where $\Phi_t = \sum_{i=1}^{N} \phi_t(i)$. Combining $\Phi_t = Pr(\mathbf{O}_t|\mathbf{O}_1, \ldots, \mathbf{O}_{t-1})$, and equation (**??**) gives the following expression for the log-likelihood:

$$l_T = \sum_{t=1}^{T} \log \Phi_t. \tag{5}$$

2.2 Computational considerations

From equations (**??**–**??**), it can be seen that computing the forward variables, and hence the log-likelihood, takes $O(Tn^2)$ computations, for an n-state model and a time series of length T.

2.3 Parameter estimation

Parameters are estimated in `depmixS4` using the EM algorithm or through the use of a general Newton-Raphson optimizer. The EM algorithm however has

some drawbacks. First, it can be slow to converge towards the end of optimization (although it is usually faster than direct optimization at the start, so possibly a combination of EM and direct optimization is fastest). 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` **??**, 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`
2. model fitting with function `fit`

3.1 Example data: speed

Throughout this manual a data set called `speed` is used. It consists of three time series with three variables: reaction time, accuracy, and a covariate Pacc which defines the relative pay-off for speeded and 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 Defining models

A dependent mixture model is defined by the number of states, and by the response distribution functions, and can be created with the `depmix`-function as follows (see the help files for other options):

```
mod <- depmix(rt~1, data=speed, nstates=2)
```

Above code illustrates the simplest case of a univariate time series without covariates; besides providing the data, the only other necessity is to specify the desired number of states. Note that the rt's are modelled here with a Gaussian distribution as that is the default family option in `depmixS4`. A multivariate model can be specified by providing a list of formulae rather than a single one as above:

```
mod <- depmix(list(rt~1,corr~1), data=speed, nstates=2,
family=list(gaussian(),multinomial())
```

Here it is also necessary to provide the family functions for each of the responses. Currently, the gaussian() and multinomial() are implemented.

The function `depmix` returns an object of class `depmix` which has its own summary function providing the parameter values of the model. The object consists of three main parts: the prior model, which specifies the initial state probabilities, the transition models, specifying the transition probabilities for each state, and the response models, specifying the densities for each response and each state. See the help files for further details.

Except in simple cases, starting values can be a problem in latent Markov models, and so in general it's best to provide them if you have a fairly good idea of what to expect. Providing starting values is done through three arguments: respstart, trstart, and instart, for response related parameters, transition parameters and prior parameters, respectively. The use of setpars reveals the ordering of parameters that should be used:

```
mod <- depmix(list(rt~1,corr~1), data=speed, nstates=2,
family=list(gaussian(),multinomial())
setpars(mod,1:npar(mod))
```

See the paragraph on parameter numbering below for further details.

3.3 Transition matrix and initial state probabilities

The transition matrix is parametrized as a list of multinomial logistic models. The initial state probabilities are similarly parametrized as a multinomial logistic model. Both models use a base category parametrization, meaning that the parameter for the base category is fixed at zero. The default base 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.

Covariates can be specified using a one-sided formula as in the following example:

```
mod <- depmix(list(rt~1,corr~1), data=speed, nstates=2,
transition=~Pacc)
```

Note that this can be done for the initial state probabilities by specifying prior= X1, where X1 is the desired covariate. The result of this is that the transition probabilities are now dependent on the covariate Pacc (which is an experimenter controlled variable to induce switching between guessing and accurate responding).

3.4 Fitting models

Fitting models is done using the function `fit`. The standard call only requires a model object of class `depmix`:

```
mod <- depmix(list(rt~1,corr~1), data=speed, nstates=2,
family=list(gaussian(),multinomial())
fmod <- fit(mod)
```

The function returns an object of class `depmix.fitted` which extends the `depmix` class, adding convergence information and possibly information about

constraints if these were applied. The function provides some online output on the progress of the optimization, the precise form of the output depends on the optimization method chosen.

Class `depmix` and `depmix.fitted` have `logLik`, `AIC` and `BIC` methods to provide fit statistics.

3.5 Fixing and constraining parameters

Constraining and fixing parameters is done using the `conpat` argument to the `depmix.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`.

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)`. 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:

```
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
```