Mixture Hidden Markov Models for Sequence Data: the seqHMM Package in R

Satu Helske
University of Jyväskylä, Finland

Jouni Helske
University of Jyväskylä, Finland

Abstract

Sequence analysis is being more and more widely used for the analysis of social sequences and other multivariate categorical time series data. However, it is often complex to describe, visualize, and compare large sequence data, especially when there are multiple parallel sequences per subject. Hidden (latent) Markov models (HMMs) are able to detect underlying latent structures and they can be used in various longitudinal settings: to account for measurement error, to detect unobservable states, or to compress information across several types of observations. Extending to mixture hidden Markov models (MHMMs) allows clustering data into homogeneous subsets, with or without external covariates.

The **seqHMM** package in R is designed for the efficient modeling of sequences and other categorical time series data containing one or multiple subjects with one or multiple interdependent sequences using HMMs and MHMMs. Also other restricted variants of the MHMM can be fitted, e.g. latent class models, Markov models, mixture Markov models, or even ordinary multinomial regression models with suitable parameterization of the HMM.

Good graphical presentations of data and models are useful during the whole analysis process from the first glimpse at the data to model fitting and presentation of results. The package provides easy options for plotting parallel sequence data, and proposes visualizing HMMs as directed graphs.

Keywords: multichannel sequences, categorical time series, visualizing sequence data, visualizing models, latent Markov models, latent class models, R.

CRAN-compliant modification of manuscript submitted to *Journal of Statistical Software*

1. Introduction

Sequence analysis is being more and more widely used for the analysis of categorical time series. These data consist of multiple independent subjects with one or multiple interdependent sequences (channels). Sequence analysis is used for computing the (dis)similarities of sequences, and often the goal is to find patterns in data using cluster analysis. However, describing, visualizing, and comparing large sequence data is often complex, especially in the case of multiple channels. Hidden (latent) Markov models (HMMs) can be used to compress and visualize information in such data. These models are able to detect underlying latent structures. Extending to mixture hidden Markov models (MHMMs) allows clustering via latent classes, possibly with additional covariate information. One of the major benefits of using hidden Markov modeling is that all stages of analysis are performed, evaluated, and

compared in a probabilistic framework; e.g. well-known model selection criteria are available for choosing the best clustering solution.

The **seqHMM** package for R (R Core Team 2015) is designed for modeling sequence data and other categorical time series with one or multiple subjects and one or multiple channels using HMMs and MHMMs. The package provides functions for the estimation and inference of models, as well as functions for the easy visualization of multichannel sequences and HMMs. Even though the package was originally developed for researchers familiar with social sequence analysis, knowledge on sequence analysis or social sciences is not necessary for the usage of **seqHMM**.

There are also other R packages in CRAN for HMM analysis of categorical data. The **HMM** package (Himmelmann 2010) is a compact package designed for fitting an HMM for a single observation sequence. The **hmm.discnp** package (Turner and Liu 2014) can handle multiple observation sequences with possibly varying lengths. For modeling continuous-time processes as hidden Markov models, the **msm** package (Jackson 2011) is available. Both **hmm.discnp** and **msm** support only single-channel observations. The **depmixS4** package (Visser and Speekenbrink 2010) is able to fit HMMs for multiple interdependent time series (with continuous or categorical values), but for one subject only. In the **msm** and **depmixS4** packages, covariates can be added for initial and transition probabilities. The **mhsmm** package (O'Connell and Højsgaard 2011) allows modeling of multiple sequences using hidden Markov and semi-Markov models. There are no ready-made options for modeling categorical data, but users can write their own extensions for arbitrary distributions. The **LMest** package (Bartolucci and Pandolfi 2015) is aimed to panel data with large number of subjects and small number of time points. It can be used for hidden Markov modeling of multivariate and multichannel categorical data, using covariates in emission and transition processes. **LMest** also support mixed latent Markov models, where the latent process is allowed to vary in different latent subpopulations. This differs from mixture hidden Markov models used in **seqHMM**, where also the emission probabilities vary between groups. The **seqHMM** package also supports covariates in explaining group memberships. A drawback in the **LMest** package is that the user cannot define initial values or zero constraints for model parameters, and thus important special cases such as left-to-right models cannot be used.

We start with describing data and methods: a short introduction to sequence data and sequence analysis, then the theory of hidden Markov models for such data, an expansion to mixture hidden Markov models and a glance at some special cases, and then some propositions on visualizing multichannel sequence data and hidden Markov models. After the theoretic part we take a look at features of the **seqHMM** package and at the end show an example on using the package for the analysis of life course data. Appendices include a list of notations and more thorough descriptions of some important algorithms.

2. Methods

2.1. Sequences and sequence analysis

By the term *sequence* we refer to an ordered set of categorical states. It can be a time series, such as a career trajectory or residental history, or any other series with ordered categorical observations, e.g. a DNA sequence or a structure of a story.

As an example we study the `biofam` data available in the **TraMineR** package (Gabadinho, Ritschard, Müller, and Studer 2011). It is a sample of 2000 individuals born in 1909–1972, constructed from the Swiss Household Panel survey in 2002 (Müller, Studer, and Ritschard 2007). The data set contains sequences of annual family life statuses from age 15 to 30. Eight observed states are defined from the combination of five basic states: living with parents, left home, married, having children, and divorced. To show a more complex example, we split the original data into three separate *channels* representing different life domains: marriage, parenthood, and residence. The data for each individual now includes three parallel sequences constituting of two or three *states* each: single/married/divorced, childless/parent, and living with parents / having left home.

Sequence analysis (SA) is statistical analysis of successions of states. It has roots in bioinformatics and computer science (see e.g. Durbin, Eddy, Krogh, and Mitchison 1998), but during the past few decades SA has also become more common in other disciplines for the analysis of longitudinal data. In social sciences SA has been used increasingly often and is now "central to the life-course perspective" (Blanchard, Bühlmann, and Gauthier 2014). SA is model-free data-driven approach, which is used for computing (dis)similarities of sequences. The most well-known method is optimal matching (McVicar and Anyadike-Danes 2002), but several alternatives exist (see e.g. Aisenbrey and Fasang 2010; Elzinga and Studer 2014; Gauthier, Widmer, Bucher, and Notredame 2009; Halpin 2010; Hollister 2009; Lesnard 2010). Also a method for analysing multichannel data has been developed (Gauthier, Widmer, Bucher, and Notredame 2010). Often the goal in SA is to find typical and atypical patterns in trajectories using cluster analysis, but any approach suitable for compressing information on the dissimilarities can be used. The data are usually presented also graphically in some way. So far the **TraMineR** package has been the most extensive and frequently used software for (social) sequence analysis.

2.2. Hidden Markov models

In the context of hidden Markov models, sequence data consists of *observed states*, which are regarded as probabilistic functions of *hidden states*. Hidden states cannot be observed directly, but only through the sequence(s) of observations, since they emit the observations on varying probabilities. A discrete first order hidden Markov model for a single sequence is characterized by the following:

- *Observed state sequence* $\mathbf{y} = (y_1, y_2, \ldots, y_T)$ with observed states $m \in \{1, \ldots, M\}$.
- *Hidden state sequence* $\mathbf{z} = (z_1, z_2, \ldots, z_T)$ with hidden states $s \in \{1, \ldots, S\}$.
- *Transition matrix* $A = \{a_{sr}\}$ of size $S \times S$, where a_{sr} is the probability of moving from the hidden state s at time $t-1$ to the hidden state r at time t:

  $$a_{sr} = P(z_t = r | z_{t-1} = s); \quad s, r \in \{1, \ldots, S\}.$$

  We only consider homogeneous HMMs, where the transition probabilities a_{sr} are constant over time.
- *Emission matrix* $B = \{b_s(m)\}$ of size $S \times M$, where $b_s(m)$ is the probability of the hidden state s emitting the observed state m:

  $$b_s(m) = P(y_t = m | z_t = s); \quad s \in \{1, \ldots, S\}, m \in \{1, \ldots, M\}.$$

- *Initial probability vector* $\pi = \{\pi_s\}$ of length S, where π_s is the probability of starting from the hidden state s:

$$\pi_s = P(z_1 = s); \quad s \in \{1, \ldots, S\}.$$

The (first order) Markov assumption states that the hidden state transition probability at time t only depends on the hidden state at the previous time point $t - 1$:

$$P(z_t|z_{t-1}, \ldots, z_1) = P(z_t|z_{t-1}). \tag{1}$$

Also, the observation at time t is only dependent on the current hidden state, not on previous hidden states or observations:

$$P(y_t|y_{t-1}, \ldots, y_1, z_t, \ldots, z_1) = P(y_t|z_t). \tag{2}$$

For a more detailed description of hidden Markov models, see e.g. Rabiner (1989), MacDonald and Zucchini (1997), and Durbin *et al.* (1998).

HMM for multiple sequences

We can also fit the same HMM for multiple subjects; instead of one observed sequence $\mathbf{y}$ we have N sequences as $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_N)^\top$, where the observations $\mathbf{y}_i = (y_{i1}, \ldots, y_{iT})$ of each subject i take values in the observed state space. Observed sequences are assumed to be mutually independent given the hidden states. The observations are assumed to be generated by the same model, but each subject has its own hidden state sequence.

HMM for multichannel sequences

In the case of multichannel sequence data, such as the example described in section 2.1, for each subject i there are C parallel sequences. Observations are now of the form y_{itc}, $i = 1, \ldots, N; \quad t = 1 \ldots, T; \quad c = 1 \ldots, C$, so that our complete data is $Y = \{Y^1, \ldots, Y^C\}$. In **seqHMM**, multichannel data are handled as a list of C data frames of size $N \times T$. We also define Y_i as all the observations corresponding to subject i.

We apply the same latent structure for all channels. In such a case the model has one transition matrix A but several emission matrices $B_1, \ldots, B_C$, one for each channel. We assume that the observed states in different channels at a given time point t are independent of each other given the hidden state at t, i.e., $P(\mathbf{y}_{it}|z_{it}) = P(y_{it1}|z_{it}) \cdots P(y_{itC}|z_{it})$.

Sometimes the independence assumption does not seem theoretically plausible. For example, even conditioning on a hidden state representing a general life stage, are marital status and parenthood truly independent? On the other hand, given a person's religious views, could their opinions on abortion and gay marriage be though as independent?

If the goal is to use hidden Markov models for prediction or simulating new sequence data, the analyst should carefully check the validity of independence assumptions. However, if the goal is merely to describe structures and compress information, it can be useful to accept the independence assumption even though it is not completely reasonable in a theoretical sense. When using multichannel sequences, the number of observed states is smaller, which leads to a more parsimonious representation of the model and easier inference of the phenomenon. Also due to the decreased number of observed states, the number of parameters of the model is decreased leading to the improved computational efficiency of model estimation.

The multichannel approach is particularly useful if some of the channels are only partially observed; combining missing and non-missing information into one observation is usually problematic. One would have to decide whether such observations are coded completely missing, which is simple but loses information, or whether all possible combinations of missing and non-missing states are included, which grows the state space larger and makes the interpretation of the model more difficult. In the multichannel approach the data can be used as it is.

Missing data

Missing observations are handled straightforwardly in the context of HMMs. When observation y_{itc} is missing, we gain no additional information regarding hidden states. In such a case, we set the emission probability $b_s(y_{itc}) = 1$ for all $s \in 1, \ldots, S$. Sequences with varying lengths are handled by setting missing values before and/or after the observed states.

Log-likelihood and parameter estimation

The unknown transition, emission and initial probabilities are commonly estimated via maximum likelihood. The log-likelihood for multiple multichannel sequences is written as

$$\log L = \sum_{i=1}^{N} \log P\left(Y_i|\mathcal{M}\right), \tag{3}$$

where Y_i are the observed sequences in channels $c = 1, \ldots, C$ for subject i and $\mathcal{M}$ describes the model and its parameters $\{\pi, A, B_1, \ldots, B_C\}$. The probability of the observation sequence of subject i given the model is

$$\begin{aligned} P(Y_i|\mathcal{M}) &= \sum_{\text{all } z} P\left(Y_i|z, \mathcal{M}\right) P\left(z|\mathcal{M}\right) \\ &= \sum_{\text{all } z} P(z_1|\mathcal{M}) P(\mathbf{y}_{i1}|z_1, \mathcal{M}) \prod_{t=2}^{T} P(z_t|z_{t-1}, \mathcal{M}) P(\mathbf{y}_{it}|z_t, \mathcal{M}) \\ &= \sum_{\text{all } z} \pi_{z_1} b_{z_1}(y_{i11}) \cdots b_{z_1}(y_{i1C}) \prod_{t=2}^{T} \left[a_{z_{t-1}z_t} b_{z_t}(y_{it1}) \cdots b_{z_t}(y_{itC})\right], \end{aligned} \tag{4}$$

where the hidden state sequences $z = (z_1, \ldots, z_T)$ take all possible combinations of values in the hidden state space $\{1, \ldots, S\}$ and where $\mathbf{y}_{it}$ are the observations of subject i at t in channels $1, \ldots, C$; π_{z_1} is the initial probability of the hidden state at time $t = 1$ in sequence z; $a_{z_{t-1}z_t}$ is the transition probability from the hidden state at time $t-1$ to the hidden state at t; and $b_{z_t}(y_{itc})$ is the probability that the hidden state of subject i at time t emits the observed state at t in channel c.

For direct numerical maximization (DNM) of the log-likelihood, any general-purpose optimization routines such as BFGS or Nelder–Mead can be used (with suitable reparameterizations). Another common estimation method is the expectation–maximization (EM) algorithm, also known as the Baum–Welch algorithm in the HMM context. The EM algorithm rapidly converges close to a local optimum, but compared to DNM, the converge speed is often slow near the optimum.

The probability (4) is efficiently calculated using the forward part of the *forward–backward algorithm* (Baum and Petrie 1966; Rabiner 1989, see appendix B). The backward part of the algorithm is needed for the EM algorithm, as well as for computation of analytical gradients for derivative based optimization routines.

The estimation process starts by giving initial values to the estimates. Good starting values are needed for finding the optimal solution in a reasonable time. In order to reduce the risk of being trapped in a poor local maximum, a large number of initial values should be tested.

Inference on hidden states

Given our model and observed sequences, we can make several interesting inferences regarding the hidden states. Forward probabilities $\alpha_{it}(s)$ (Rabiner 1989) are defined as the joint probability of hidden state s at time t and the observation sequences $\mathbf{y}_{i1}, \ldots, \mathbf{y}_{it}$ given the model $\mathcal{M}$, whereas backward probabilities $\beta_{it}(s)$ are defined as the joint probability of hidden state s at time t and the observation sequences $\mathbf{y}_{i(t+1)}, \ldots, \mathbf{y}_{iT}$ given the model $\mathcal{M}$.

From forward and backward probabilities we can compute the *posterior probabilities* of states, which give the probability of being in each hidden state at each time point, given the observed sequences of subject i. These are defined as

$$P(z_{it} = s|Y_i, \mathcal{M}) = \frac{\alpha_{it}\beta_{it}}{P\left(Y_i|\mathcal{M}\right)}. \tag{5}$$

Posterior probabilities can be used to find the locally most probable hidden state at each time point, but the resulting sequence is not necessarily globally optimal. To find the single best hidden state sequence $\hat{z}_i(Y_i) = \hat{z}_{i1}, \hat{z}_{i2}, \ldots, \hat{z}_{iT}$ for subject i, we maximize $P(z|Y_i, \mathcal{M})$ or, equivalently, $P(z, Y_i|\mathcal{M})$. A dynamic programming method, the *Viterbi algorithm* (Rabiner 1989, see appendix C), is used for solving the problem.

Model comparison

Models with the same number of parameters can be compared with the log-likelihood. For choosing between models with a different number of hidden states, we need to take account of the number of parameters. We define the Bayesian information criterion (BIC) as

$$BIC = -2\log(L_d) + p\log\left(\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{1}{C}\sum_{c=1}^{C}\mathrm{I}(y_{itc} \text{ observed})\right), \tag{6}$$

where L_d is computed using equation 3, p is the number of estimated parameters, I is the indicator function, and the summation in the logarithm is the size of the data. If data are completely observed, the summation is simplified to $N \times T$. Missing observations in multichannel data may lead to non-integer data size.

2.3. Clustering by mixture hidden Markov models

There are many approaches for finding and describing clusters or latent classes when working with HMMs. A simple option is to group sequences beforehand (e.g. using sequence analysis and some clustering method), after which one HMM is fitted for each cluster. This approach is simple in terms of HMMs. Models with a different number of hidden states and initial

values are explored and compared one cluster at a time. HMMs are used for compressing information and comparing different clustering solutions, e.g. finding the best number of clusters. The problem with this solution is that it is, of course, very sensitive to the original clustering and the estimated HMMs might not be well suited for borderline cases.

Instead of fixing sequences into clusters, it is possible to fit one model for the whole data and determine clustering during modeling. Now sequences are not in fixed clusters but get assigned to clusters with certain probabilities during the modeling process. In this section we expand the idea of HMMs to mixture hidden Markov models (MHMMs). This approach was formulated by van de Pol and Langeheine (1990) as a mixed Markov latent class model and later generalized to include time-constant and time-varying covariates by Menard (2008) (who named the resulting model as mixture latent Markov model, MLMM). The MHMM presented here is a variant of MLMM where only time-constant covariates are allowed. Time-constant covariates deal with unobserved heterogeneity and they are used for predicting cluster memberships of subjects.

Mixture hidden Markov model

Assume that we have a set of models $\mathcal{M} = \{\mathcal{M}^1, \ldots, \mathcal{M}^K\}$, where $\mathcal{M}^k = \{\pi^k, A^k, B_1^k, \ldots, B_C^k\}$ for $k = 1, \ldots, K$. For each subject Y_i, denote $P(\mathcal{M}^k) = w_k$ as the prior probability that the observation sequences of subject i belongs to the submodel/cluster $\mathcal{M}^k$. Now the log-likelihood is extended from equation (3) as

$$\begin{aligned}
\log L &= \sum_{i=1}^{N} \log P(Y_i|\mathcal{M}) \\
&= \sum_{i=1}^{N} \log \left[\sum_{k=1}^{K} P(\mathcal{M}^k) \sum_{\text{all } z} P\left(Y_i|z, \mathcal{M}^k\right) P\left(z|\mathcal{M}^k\right) \right] \\
&= \sum_{i=1}^{N} \log \left[\sum_{k=1}^{K} w_k \sum_{\text{all } z} \pi_{z_1}^k b_{z_1}^k(y_{i11}) \cdots b_{z_1}^k(y_{i1C}) \prod_{t=2}^{T} \left[a_{z_{t-1}z_t}^k b_{z_t}^k(y_{it1}) \cdots b_{z_t}^k(y_{itC}) \right] \right].
\end{aligned} \tag{7}$$

Compared to the usual hidden Markov model, there is an additional summation over the clusters in equation (7), which seems to make the computations less straightforward than in the non-mixture case. Fortunately, by redefining MHMM as a special type HMM allows us to use standard HMM algorithms without major modifications. We combine the K submodels into one large hidden Markov model consisting of $\sum_{k=1}^{K} S_k$ states, where the initial state vector contains elements of the form $w_k \pi^k$. Now the transition matrix is block diagonal

$$A = \begin{pmatrix} A^1 & 0 & \cdots & 0 \\ 0 & A^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A^K \end{pmatrix}, \tag{8}$$

where the diagonal blocks $A^k, k = 1, \ldots, K$, are square matrices containing the transition probabilities of one cluster. The off-diagonal blocks are zero matrices, so transitions between clusters are not allowed. Similarly, the emission matrices for each channel contain stacked emission matrices B^k.

Covariates and cluster probabilities

Covariates can be added to MHMM to explain cluster memberships as in latent class analysis. The prior cluster probabilities now depend on the subject's covariate values $\mathbf{x}_i$ and are defined as multinomial distribution:

$$P(\mathcal{M}^k|\mathbf{x}_i) = w_{ik} = \frac{e^{\beta_k \mathbf{x}_i}}{1 + \sum_{j=2}^{K} e^{\beta_j \mathbf{x}_i}}. \tag{9}$$

The first cluster is set as the reference by fixing $\beta_1 = 0$. Note that by convention we use β when referring to regression coefficients. It is not to be mixed with backward probabilities, which are usually given the same notation.

As in MHMM without covariates, we can still use standard HMM algorithms with a slight modification; we now allow initial state probabilities to vary between subjects. Of course, we also need to estimate the coefficients β. For direct numerical maximization the modifications are straightforward. In the EM algorithm, regarding the M-step for β, `seqHMM` uses Newton's method with analytical gradients and Hessian which are straightforward to compute given all other model parameters. This Hessian can also be used for computing the conditional standard errors of coefficients. For unconditional standard errors, which take account of possible correlation between the estimates of β and other model parameters, the Hessian is computed using finite difference approximation of the Jacobian of the analytical gradients.

The posterior cluster probabilities $P(\mathcal{M}^k|Y_i, \mathbf{x}_i)$ are obtained as

$$\begin{aligned} P(\mathcal{M}^k|Y_i, \mathbf{x}_i) &= \frac{P(Y_i|\mathcal{M}^k, \mathbf{x}_i)P(\mathcal{M}^k|\mathbf{x}_i)}{P(Y_i|\mathbf{x}_i)} \\ &= \frac{P(Y_i|\mathcal{M}^k, \mathbf{x}_i)P(\mathcal{M}^k|\mathbf{x}_i)}{\sum_{j=1}^{K} P(Y_i|\mathcal{M}^j, \mathbf{x}_i)P(\mathcal{M}^j|\mathbf{x}_i)} = \frac{L_k^i}{L^i}, \end{aligned} \tag{10}$$

where L^i is the likelihood of the complete MHMM for subject i, and L_k^i is the likelihood of cluster k for subject i. These are straightforwardly computed from forward probabilities. Posterior cluster probabilities are used e.g. for computing classification tables.

2.4. Important special cases

The hidden Markov model is not the only important special case of the mixture hidden Markov model. Here we cover some of the most important special cases that are included in the **seqHMM** package.

Markov model

The Markov model (MM) is a special case of the HMM, where there is no hidden structure. It can be regarded as an HMM where the hidden states correspond to the observed states perfectly. Now the number of hidden states matches the number of the observed states. The emission probability $P(y_{it}) = 1$ if $z_t = y_{it}$ and 0 otherwise, i.e., the emission matrices are identity matrices. Note that for building Markov models the data must be in a single-channel format.

Mixture Markov model

Like MM, the mixture Markov model (MMM) is a special case of the MHMM, where there is

no hidden structure. The likelihood of the model is now of the form

$$\begin{aligned} \log L &= \sum_{i=1}^{N} \log P(\mathbf{y}_i|\mathbf{x}_i, \mathcal{M}^k) = \sum_{i=1}^{N} \log \sum_{k=1}^{K} P(\mathcal{M}^k|\mathbf{x}_i) P(\mathbf{y}_i|\mathbf{x}_i, \mathcal{M}^k) \\ &= \sum_{i=1}^{N} \log \sum_{k=1}^{K} P(\mathcal{M}^k|\mathbf{x}_i) P(y_{i1}|\mathbf{x}_i, \mathcal{M}^k) \prod_{t=2}^{T} P(y_{it}|y_{i(t-1)}, \mathbf{x}_i, \mathcal{M}^k). \end{aligned} \tag{11}$$

Again, the data must be in a single-channel format.

Latent class model

Latent class models (LCM) are another class of models that are often used for longitudinal research. Such models have been called, e.g., (latent) growth models, latent trajectory models, or longitudinal latent class models (Menard 2008; Collins and Wugalter 1992). These models assume that dependencies between observations can be captured by a latent class, i.e., a time-constant variable which we call cluster in this paper.

The **seqHMM** includes a function for fitting an LCM as a special case of MHMM where there is only one hidden state for each cluster. The transition matrix of each cluster is now reduced to a scalar 1 and the likelihood is of the form

$$\begin{aligned} \log L &= \sum_{i=1}^{N} \log P(Y_i|\mathbf{x}_i, \mathcal{M}^k) = \sum_{i=1}^{N} \log \sum_{k=1}^{K} P(\mathcal{M}^k|\mathbf{x}_i) P(Y_i|\mathbf{x}_i, \mathcal{M}^k) \\ &= \sum_{i=1}^{N} \log \sum_{k=1}^{K} P(\mathcal{M}^k|\mathbf{x}_i) \prod_{t=1}^{T} P(\mathbf{y}_{it}|\mathbf{x}_i, \mathcal{M}^k). \end{aligned} \tag{12}$$

For LCMs, the data can consist of multiple channels, i.e., the data for each subject consists of multiple parallel sequences. It is also possible to use **seqHMM** for estimating LCMs for non-longitudinal data with only one time point, e.g. to study multiple questions in a survey.

3. Package features

The purpose of the **seqHMM** package is to offer tools for the whole HMM analysis process from sequence data manipulation and description to model building, evaluation, and visualization. Naturally, **seqHMM** builds on other packages, especially the **TraMineR** package designed for sequence analysis. For constructing, summarizing, and visualizing sequence data, **TraMineR** provides many useful features. First of all, we use the **TraMineR**'s `stslist` class as the sequence data structure of **seqHMM**. These state sequence objects have attributes such as color palette and alphabet, and they have specific methods for plotting, summarizing, and printing. Many other **TraMineR**'s features for plotting or data manipulation are also used in **seqHMM**.

On the other hand, **seqHMM** extends the functionalities of **TraMineR**, e.g. by providing easy-to-use plotting functions for multichannel data and a simple function for converting such data into single-channel representation.

Other significant packages include the **igraph** package (Csardi and Nepusz 2006), which is used for drawing graphs of HMMs, and the **nloptr** package (Ypma, Borchers, and Eddelbuettel

Table 1: Functions and methods in the **seqHMM** package

Usage	Functions/methods
Model construction	`build_hmm`, `build_mhmm`, `build_mm`, `build_mmm`, `build_lcm`, `simulate_initial_probs`, `simulate_transition_probs`, `simulate_emission_probs`
Model estimation	`fit_model`
Model visualization	`plot`, `ssplot`, `mssplot`
Model inference	`logLik`, `BIC`, `summary`
State inference	`hidden_paths`, `posterior_probs`, `forward_backward`
Data visualization	`ssplot`, `ssp` + `plot`, `ssp` + `gridplot`
Data and model manipulation	`mc_to_sc`, `mc_to_sc_data`, `trim_hmm`, `separate_mhmm`
Data simulation	`simulate_hmm`, `simulate_mhmm`

2014; Johnson 2014), which is used in direct numerical optimization of model parameters. The computationally intensive parts of the package are written in C++ with the help of the **Rcpp** (Eddelbuettel and François 2011; Eddelbuettel 2013) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014) packages.

In addition of using C++ for major algorithms, **seqHMM** also supports parallel computation via OpenMP interface by dividing computations for subjects between threads. The user can choose the number of parallel threads (typically the number of cores) to use for the specific task using argument `threads` where available.

Table 3 shows the functions and methods available in the **seqHMM** package. The package includes functions for estimating and evaluating HMMs and MHMMs as well as visualizing data and models. There are some functions for manipulating data and models, and for simulating model parameters or sequence data given a model. In the next sections we will discuss the usage of these functions more thoroughly.

As the straightforward implementation of the forward–backward algorithm poses a great risk of under- and overflow, typically forward probabilities are scaled so that there should be no underflow. Although scaling is often sufficient for forward algorithm, it can still result in an overflow problem in the backward algorithm. This is especially true in case of global optimization algorithms which can search infeasible areas of parameter space. Thus, `seqHMM` also supports computation on the logarithmic scale in most of the algorithms, which further reduces the numerical unstabilities. On the other hand, as there is a need to back-transform to the natural scale during the algorithms, the log-space approach is somewhat slower than the scaling approach. Therefore, the default option is to use the scaling approach, which can be changed to the log-space approach by setting the `log_space` argument to `TRUE` e.g. in `fit_model`.

3.1. Building and fitting models

A model is first constructed using an appropriate build function. As the Table 3 illustrates, there are several such functions available: `build_hmm` for hidden Markov models, `build_mhmm`

for mixture hidden Markov models, `build_mm` for Markov models, `build_mmm` for mixture Markov models, and `build_lcm` for latent class models.

Build functions check that the data and matrices are of the right form and create an object of class `hmm` (for HMMs and MMs) or `mhmm` (for MHMMs, MMMs, and LCMs). For the latter, covariates can be omitted or added with the usual `formula` argument using symbolic formulas familiar from e.g. the `lm` function. Even though missing observations are allowed in sequence data, covariates must be completely observed.

After a model is constructed, model parameters are estimated with the `fit_model` function. MMs, MMMs, and LCMs are handled internally as their more general counterparts, except in the case of `print` methods where some reduntant parts of the model are not printed.

In all models, initial zero probabilities are regarded as structural zeroes and only positive probabilities are estimated. Thus it is easy to construct e.g. a left-to-right model by defining transition probability matrix as an upper triangular matrix.

The `fit_model` function provides three estimation steps: 1) EM algorithm, 2) global DNM, and 3) local DNM. The user can call for one method or any combination of these steps, but should note that they are performed in the above-mentioned order. At the first step starting values are based on the model object given to `fit_model`. Results from a former step are then used as starting values in a latter. Exception to this are some global optimization algorithms, which do not use initial values (because of this, performing just the local DNM step can lead to better solutions than global DNM with small number of iterations).

In order to reduce the risk of being trapped in a poor local optimum, a large number of initial values should be tested. The **seqHMM** package strives to automize this. One option is to run EM algorithm multiple times with more or less random starting values for transition or emission probabilities or both. These are called for in the `control_em` argument. Although not done by default, this method seems to perform very well as the EM algorithm is relatively fast compared to DNM.

Another option is to use multilevel single-linkage method (MLSL) (Rinnooy Kan and Timmer 1987a,b). It draws multiple random starting values and performs local optimization from each starting point. The LDS modification uses low-discrepancy sequences instead of random numbers as starting points and should improve the convergence rate (Kucherenko and Sytsko 2005).

By default, the `fit_model` function uses the EM algorithm with a maximum of 1000 iterations and skips the local and global DNM steps. For local step, the L-BFGS algorithm (Nocedal 1980; Liu and Nocedal 1989) is used by default. Setting `global_step = TRUE` the function performs MSLS-LDS with the L-BFGS as the local optimizer. In order to reduce the computation time spent on non-global optima, the convergence tolerance of the local optimizer is set relatively large, so again local optimization should be performed at the final step. For DNM steps (2 and 3), any optimization method available in the **nloptr** package can be used.

There are some theoretical guarantees that the MLSL method finds all local optima in a finite number of local optimizations. Of course, it might not always succeed in a reasonable time. Also, it requires setting boundaries for the parameter space, which is not always straightforward. In DNM steps the transition, emission, and initial probabilities are estimated using unconstrained reparameterization using softmax function (a generalization of the logistic function), but good boundaries are essential for efficient use of the MLSL algorithm. If the boundaries are too strict, the global optimum cannot be found; if too wide, the probability

of finding the global optimum is decreased. The `fit_model` function uses starting values or results from the preciding estimation step to adjust the boundaries. EM can help in setting good boundaries, but in some cases it can also lead to worse results. For finding the best solution, it is advisable to try a couple of different settings; e.g. randomized EM, EM followed by MLSL, a couple of EM iterations followed by MLSL, and only MLSL.

State and model inference

In `seqHMM`, forward and backward probabilities are computed using the `forward_backward` function, either on the logarithmic scale or in a form of scaled probabilities, depending on the argument `log_space`. Posterior probabilities are obtained from the `posterior_probs` function. In `seqHMM`, the most probable paths are computed with the `hidden_paths` function. For details of Viterbi and forward–backward algorithm, see e.g. Rabiner (1989).

The `seqHMM` package provides the `logLik` method for computing the log-likelihood of a model. The method returns an object of class `logLik` which is compatible with the generic information criterion functions `AIC` and `BIC` of R. When constructing the `hmm` and `mhmm` objects via model building functions, the number of observations and the number of parameters of the model are stored as attributes `nobs` and `df` which are extracted by `logLik` method for computation of information criterions. The number of model parameters are defined from the initial model by taking account the parameter redundancy constraints (stemming from sum-to-one constraints of transition, emission, and initial state probabilities) and by defining all zero probabilities as structural, fixed values.

The `summary` method automatically computes some features for a MHMM, MMM, and latent class model, e.g. standard errors for covariates and prior and posterior cluster probabilities for subjects. A `print` method for this summary shows an output of the summaries: estimates and standard errors for covariates, log-likelihood and BIC, and information on most probable clusters and prior probabilities.

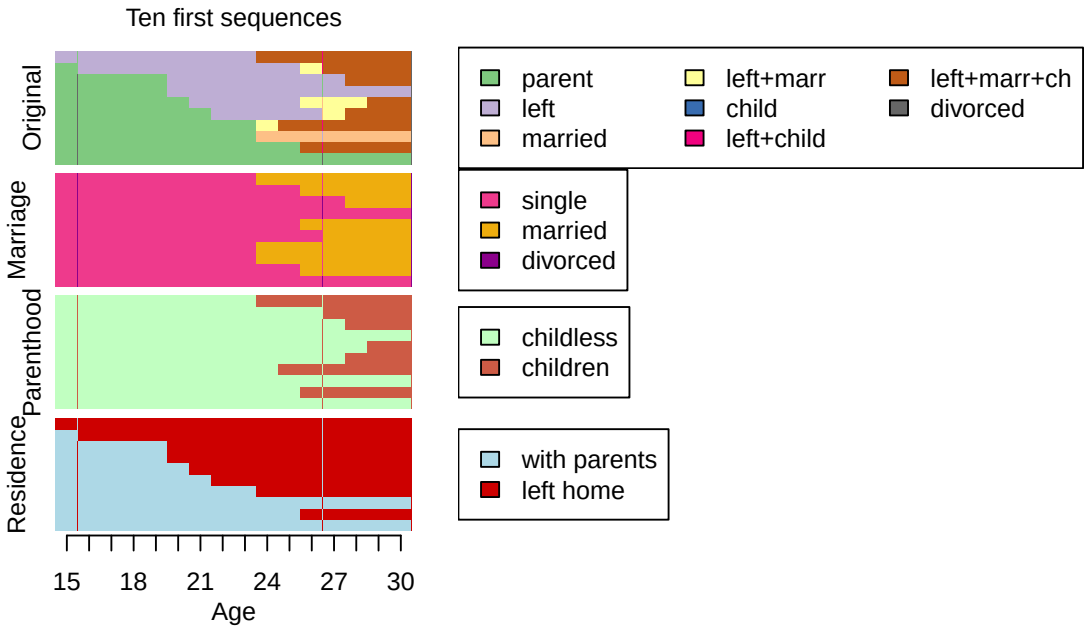

Figure 1: Stacked sequence plot of the first ten individuals in the `biofam` data plotted with the `ssplot` function. The top plot shows the original sequences, and the three bottom plots show the sequences in the separate channels for the same individuals. The sequences are in the same order in each plot, i.e., the same row always matches the same individual.

3.2. Visualizing sequence data

Good graphical presentations of data and models are useful during the whole analysis process from the first glimpse into data to model fitting and presentation of results. The **TraMineR** package provides nice plotting options and summaries for simple sequence data, but at the moment there is no easy way of plotting multichannel data. We propose to use a so called *stacked sequence plot* (ssp), where the channels are plotted on top of each other so that the same row in each figure matches the same subject. Figure 1 illustrates an example of a stacked sequence plot with the ten first sequences of the `biofam` data set. The code for creating the figure is shown in section 4.1.

The `ssplot` function is the simplest way of plotting multichannel sequence data in `seqHMM`. It can be used to illustrate state distributions or sequence index plots. The former is the default option, since index plots can take a lot of time and memory if data are large. Figure 2 illustrates a default plot which the user can modify in many ways (see the code in section 4.1). More examples are shown in the documentation pages of the `ssplot` function.

Another option is to define function arguments with the `ssp` function and then use previously saved arguments for plotting with a simple `plot` method. It is also possible to combine several `ssp` figures into one plot with the `gridplot` function. Figure 3 illustrates an example of such plot showing sequence index plots for women and men (see the code in section 4.1). Sequences are ordered in a more meaningful order using multidimensional scaling scores of observations (computed from sequence dissimilarities). After defining the plot for one group, a similar plot for others is easily defined using the `update` function.

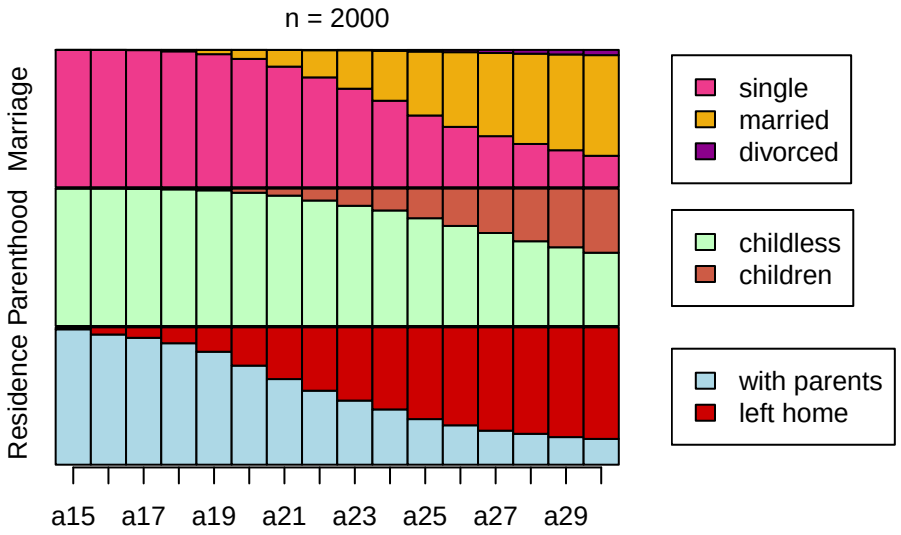

Figure 2: Stacked sequence plot of annual state distributions in the three-channel `biofam` data. This is the default output of the `ssplot` function.

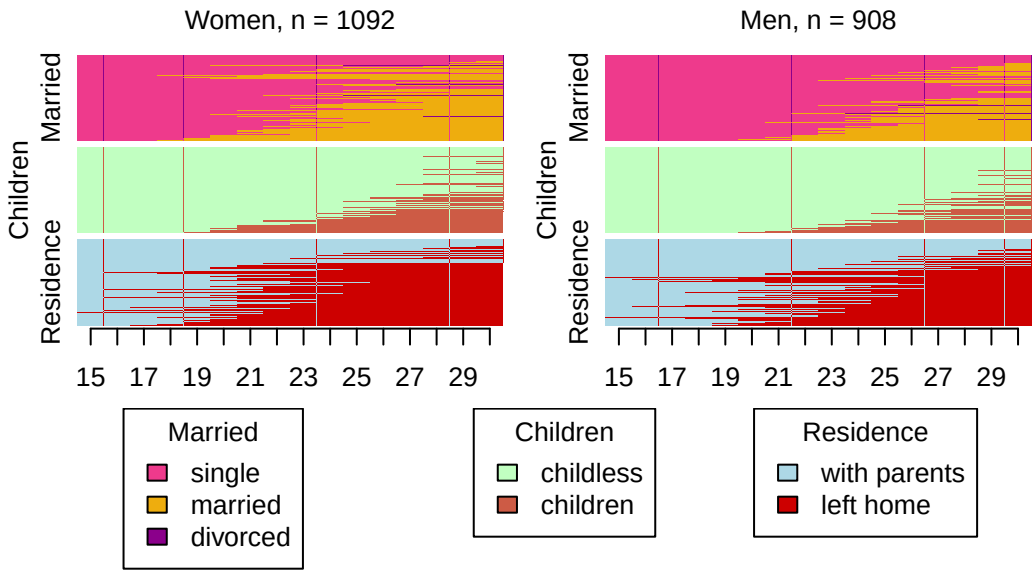

Figure 3: Showing state distribution plots for women and men in the `biofam` data. Two figures were defined with the `ssp` function and then combined into one figure with the `gridplot` function.

The `gridplot` function is useful for showing different features for the same subjects or same features for different groups. The user has a lot of control over the layout, e.g. dimensions of the grid, widths and heights of the cells, and positions of the legends.

We also provide a function `mc_to_sc_data` for easy conversion of multichannel sequence data into a single channel representation. Plotting combined data is often useful in addition to (or instead of) showing separate channels.

3.3. Visualizing hidden Markov models

For easy visualization of the model structure and parameters, we propose plotting HMMs as directed graphs. Such graphs are easily called with the `plot` method, with an object of class `hmm` as an argument. Figure 4 illustrates a five-state HMM. The code for producing the plot is shown in section 4.4.

Hidden states are presented with pie charts as vertices (or nodes), and transition probabilities are shown as edges (arrows, arcs). By default, the higher the transition probability, the thicker the stroke of the edge. Emitted observed states are shown as slices in the pies. For gaining a simpler view, observations with small emission probabilities (less than 0.05 by default) can be combined into one category. Initial state probabilities are given below or next to the respective vertices. In a case of multichannel sequences, the data and the model are converted into a single-channel representation with the `mc_to_sc` function.

A simple default plot is easy to call, but the user has a lot of control over the layout. Figure 5 illustrates another possible visualization of the same model. The code is shown in section 4.4.

For defining the colors, the plotting functions use `colorpalette` data, which is a list of ready-made color palettes with 1–200 distinct colors. It is provided in the package, so the user can easily modify colors in the plots. See also the `RColorBrewer` package (Neuwirth 2014) for more color palettes with distinct colors. The `plot_colors` function is provided for easy visualization of color palettes.

The `ssplot` function (see section 3.2) also accepts an object of class `hmm`. The user can easily choose to plot observations, most probable paths of hidden states, or both. The function automatically computes hidden paths if the user does not provide them.

Figure 6 shows observed sequences with the most probable paths of hidden states given the model. Sequences are sorted according to multidimensional scaling scores computed from hidden paths. The code for creating the plot is show in section 4.4.

The `plot` method works for `mhmm` objects as well. The user can choose between an interactive mode, where the model for each (chosen) cluster is plotted separately, and a combined plot with all models in one plot. The equivalent to the `ssplot` function for MHMMs is `mssplot`. It plots stacked sequence plots separately for each cluster. If the user asks to plot more than one cluster, the function is interactive by default.

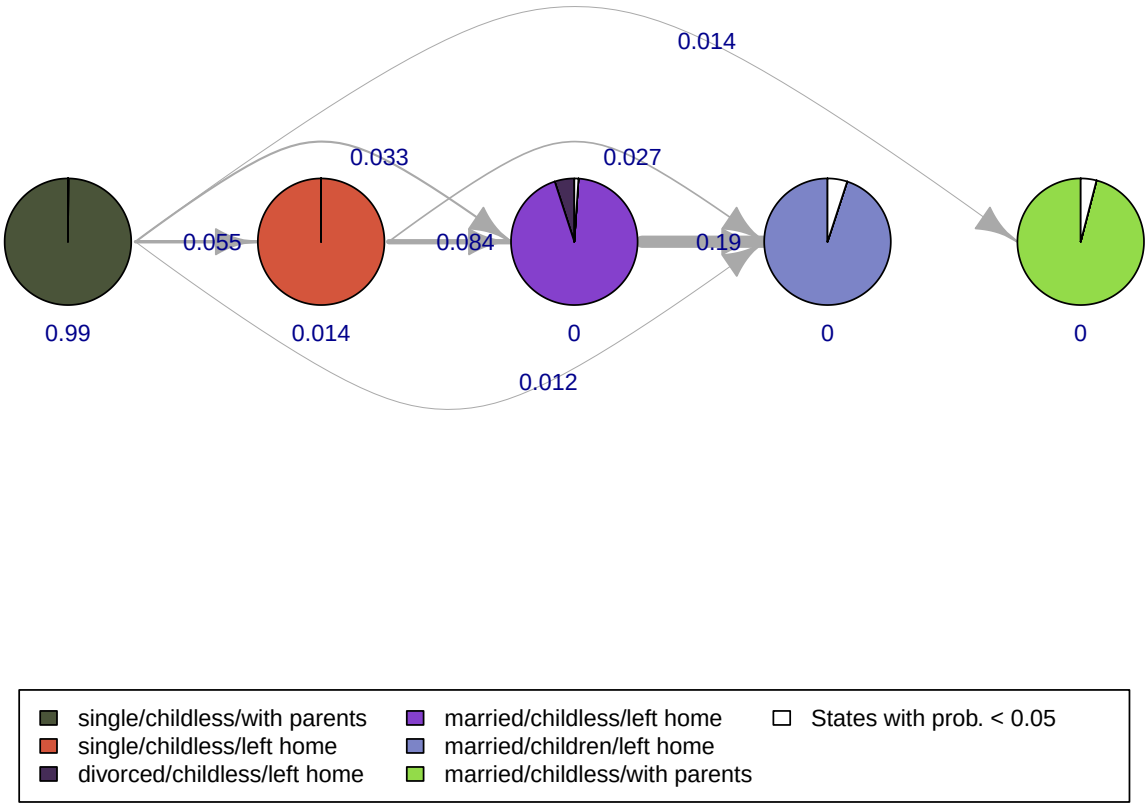

Figure 4: Illustrating a hidden Markov model as a directed graph. Pies represent five hidden states, with slices showing emission probabilities of combinations of observed states. States with emission probability less than 0.05 are combined into one slice. Edges show the transtion probabilities. Initial probabilities of hidden states are given below the pies.

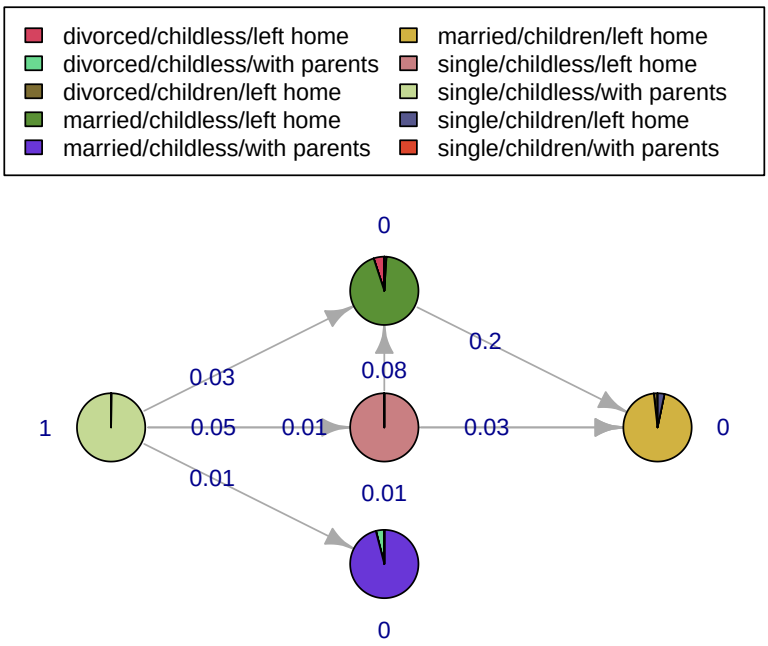

Figure 5: Another version of the hidden Markov model of Figure 4 with a different layout and modified labels, legends, and colors. All observed states are shown.

4. Examples with life course data

In this section we show examples of using the **seqHMM** package. We start by constructing and visualizing sequence data, then show how HMMs are built and fitted for single-channel and multichannel data, then move on to clustering with MHMMs, and finally illustrate how to plot HMMs.

Throughout the examples we use the same `biofam` data described in section 2.1. We use both the original single-channel data and the three-channel modification named `biofam3c`, which is included in the **seqHMM** package. See more information on the conversion from the documentation of the `biofam3c` data.

4.1. Sequence data

Before getting to estimation, it is good to get to know the data. We start by loading the original `biofam` data as well as the three-channel version of the same data, `biofam3c`. We convert the data into the `stslist` form with the `seqdef` function. We set the starting age at 15 and set the order of the states with the `alphabet` argument (for plotting). Colors of the states can be modified and stored as an attribute in the `stslist` object – this way the user only needs to define them once.

```
R> library("seqHMM")
R>
R> data("biofam", package = "TraMineR")
R> biofam_seq <- seqdef(
```

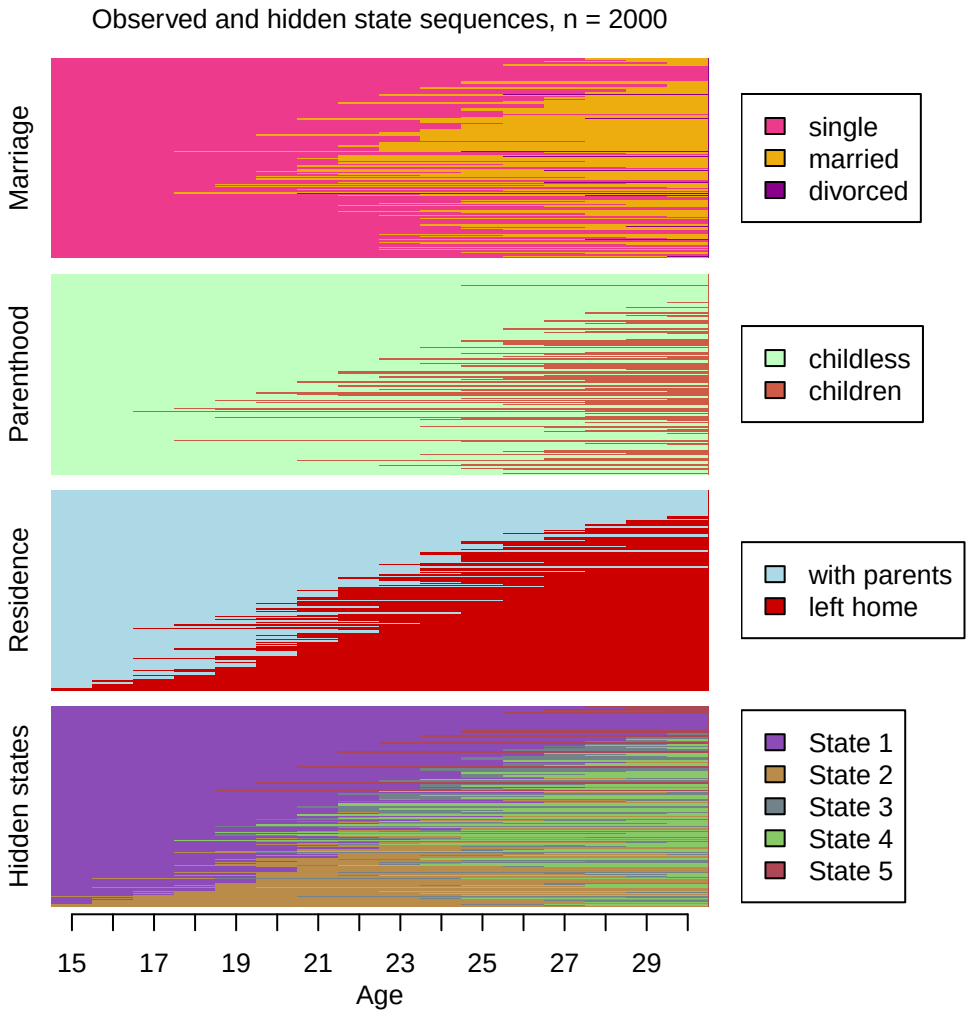

Figure 6: Using the `ssplot` function for a `hmm` object makes it easy to plot observed sequences together with the most probable paths of hidden states given the model.

```
+    biofam[, 10:25],
+    labels = c("parent", "left", "married", "left+marr", "child",
+               "left+child", "left+marr+ch", "divorced"),
+    start = 15)
R>
R> data("biofam3c")
R> marr_seq <- seqdef(biofam3c$married, start = 15,
+                     alphabet = c("single", "married", "divorced"))
R> child_seq <- seqdef(biofam3c$children, start = 15,
+                      alphabet = c("childless", "children"))
R> left_seq <- seqdef(biofam3c$left, start = 15,
+                     alphabet = c("with parents", "left home"))
R>
R> attr(marr_seq, "cpal") <- c("violetred2", "darkgoldenrod2", "darkmagenta")
R> attr(child_seq, "cpal") <- c("darkseagreen1", "coral3")
R> attr(left_seq, "cpal") <- c("lightblue", "red3")
```

Here we show codes for creating figures 2, 1, and 3. Such plots give a good glimpse on multichannel data.

Figure 2: Plotting state distributions

We start by showing how to call the simple default plot of Figure 2 in section 3.3. By default the function plots state distributions (`type = "d"`). Multichannel data are given as a list where each component is an `stslist` corresponding one channel. If names are given, those will be used as labels in plotting.

```
R> ssplot(list("Marriage" = marr_seq, "Parenthood" = child_seq,
+    "Residence" = left_seq))
```

Figure 1: Plotting sequences

Figure 1 with the whole sequences requires modifying more arguments. We call for sequence index plots (`type = "I"`) and sort sequences according to the first channel (the original sequences), starting from the beginning. We give labels to y and x axes and modify the positions of y labels. We give a title to the plot but omit the number of subjects, which by default is printed. We set the proportion of the plot given to legends and the number of columns in each legend.

```
R> ssplot(list(biofam_seq[1:10,], marr_seq[1:10,], child_seq[1:10,],
+    left_seq[1:10,]),
+    sortv = "from.start", sort.channel = 1, type = "I",
+    ylab = c("Original", "Marriage", "Parenthood", "Residence"),
+    xtlab = 15:30, xlab = "Age", title = "Ten first sequences",
+    title.n = FALSE, legend.prop = 0.63, ylab.pos = c(1, 1.5),
+    ncol.legend = c(3, 1, 1, 1))
```

Figure 3: Plotting sequence data in a grid

For using the `gridplot` function, we first need to specify the `ssp` objects of the separate plots. Here we start by defining the first plot for women with the `ssp` function. It stores the features of the plot, but does not draw anything. We want to sort sequences according to multidimensional scaling scores. These are computed from optimal matching dissimilarities for observed sequences. Any dissimilarity method available in `TraMineR` can be used instead of the default (see the documentation of the `seqdef` function for more information). We want to use the same legends for the both plots, so we remove legends from the `ssp` objects.

Since we are going to plot to two similar figures, one for women and one for men, we can pass the first `ssp` object to the `update` function. This way we only need to define the changesand omit everything that is similar.

These two `ssp` objects are then passed on to the `gridplot` function. Here we make a 2×2 grid, of which the bottom row is for the legends, but the function can also automatically determine the number of rows and columns and the positions of the legends.

```
R> ssp_f <- ssp(
+    list(marr_seq[biofam3c$covariates$sex == "woman",],
+      child_seq[biofam3c$covariates$sex == "woman",],
+      left_seq[biofam3c$covariates$sex == "woman",]),
+    type = "I", sortv = "mds.obs", withlegend = FALSE,
+    title = "Women", ylab.pos = c(1, 2, 1),
+    ylab = c("Married", "Children", "Residence"), xtlab = 15:30)
R>
R> ssp_m <- update(ssp_f, title = "Men",
+    x = list(marr_seq[biofam3c$covariates$sex == "man",],
+      child_seq[biofam3c$covariates$sex == "man",],
+      left_seq[biofam3c$covariates$sex == "man",]))
R>
R> gridplot(list(ssp_f, ssp_m), ncol = 2, nrow = 2, byrow = TRUE,
+    legend.pos = "bottom", legend.pos2 = "top", row.prop = c(0.65, 0.35))
```

Figure 7: Converting multichannel data to single-channel

When working with multiple channels, it is useful to look at the combined data as well. The `mc_to_sc_data` converts the data into a single-channel representation. At each time point of each subject, the states in each channel are combined into one. Note that here the number of combined observations (10 states) is larger than in the original data (8 states), because we have split the original divorced state into three.

Also single-channel data can be plotted with the `ssplot` function. Figure 7 illustrates the state distributions of the combined data. Here we asko to show y axis, which by default is omitted for gaining less cluttered output in stacked plots.

```
R> sc_data <- mc_to_sc_data(list(marr_seq, child_seq, left_seq))
R>
R> ssplot(sc_data, type = "d", ylab = "Proportion", yaxis = TRUE,
+    xtlab = 15:30, xlab = "Age", title = "Combined states", legend.prop = 0.4)
```

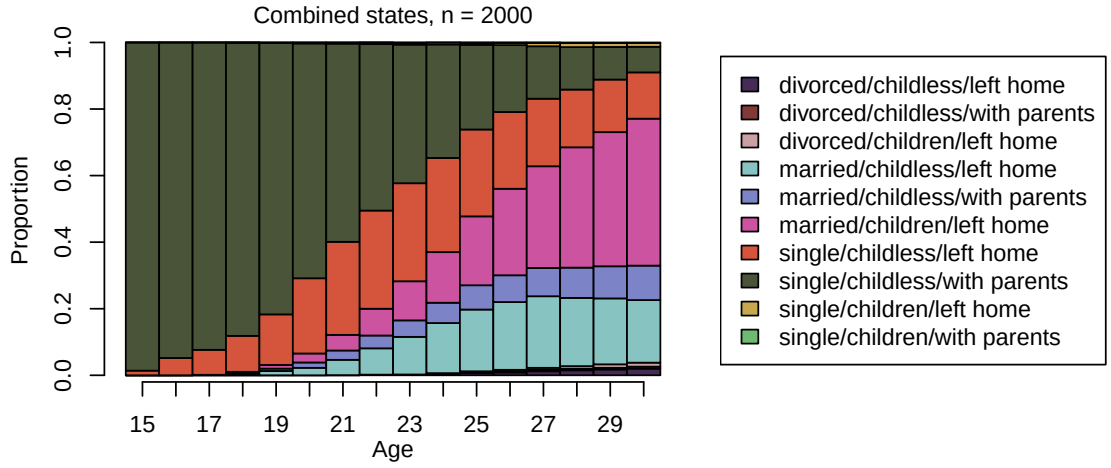

Figure 7: Three-channel `biofam3c` data converted into single-channel data.

4.2. Hidden Markov models

We start by showing how to fit a HMM for single-channel `biofam` data.

First we set starting values for initial, transition, and emission probabilities. Here the hidden states are regarded as more general life stages during which individuals are more likely to meet certain observable life events. We expect that the life stages are somehow related to age, so constructing starting values from observed state frequencies by age group seems like an option worth a try (these are easily computed using the `seqstatf` function in **TraMineR**). We construct a model with four hidden states using age groups 15–18, 19–21, 22–24, 25–27 and 28–30.

The `fit_model` function uses the probabilities given by the initial model as starting values when estimating the parameters. Only positive probabilities are estimated; zero values are fixed to zero. Thus amount of 0.1 is added to each value in a case of zero-frequencies in some categories (at this point we do not want to fix any parameters to zero). Each row is divided with its sum, so that the row sums equal to 1.

```
R> sc_init <- c(0.9, 0.06, 0.02, 0.01, 0.01)
R>
R> sc_trans <- matrix(
+    c(0.80, 0.10, 0.05, 0.03, 0.02,
+      0.02, 0.80, 0.10, 0.05, 0.03,
+      0.02, 0.03, 0.80, 0.10, 0.05,
+      0.02, 0.03, 0.05, 0.80, 0.10,
+      0.02, 0.03, 0.05, 0.05, 0.85),
+    nrow = 5, ncol = 5, byrow = TRUE)
R>
R> sc_emiss <- matrix(NA, nrow = 5, ncol = 8)
R> sc_emiss[1,] <- seqstatf(biofam_seq[, 1:4])[, 2] + 0.1
R> sc_emiss[2,] <- seqstatf(biofam_seq[, 5:7])[, 2] + 0.1
R> sc_emiss[3,] <- seqstatf(biofam_seq[, 8:10])[, 2] + 0.1
```

```
R> sc_emiss[4,] <- seqstatf(biofam_seq[, 11:13])[, 2] + 0.1
R> sc_emiss[5,] <- seqstatf(biofam_seq[, 14:16])[, 2] + 0.1
R> sc_emiss <- sc_emiss / rowSums(sc_emiss)
```

The model is initialized with the `build_hmm` function. It checks that the data and matrices are of the right form and creates an object of class `hmm`. Markov models are constructed in a similar way using the `build_mm` function, only emission probabilities are omitted.

```
R> sc_initmod <- build_hmm(observations = biofam_seq, initial_probs = sc_init,
+    transition_probs = sc_trans, emission_probs = sc_emiss)
```

We then use `fit_model` for parameter estimation. Here we estimate the model using the default options of the EM step.

```
R> sc_fit <- fit_model(sc_initmod)
```

The fitting function returns the estimated model, its log-likelihood, and information on the optimization steps.

```
R> sc_fit$logLik

[1] -16781.99

R> sc_fit$model

Initial probabilities :
State 1 State 2 State 3 State 4 State 5
  0.986   0.000   0.014   0.000   0.000

Transition probabilities :
         to
from      State 1 State 2 State 3 State 4 State 5
  State 1   0.786   0.175  0.0391 0.00000  0.0000
  State 2   0.000   0.786  0.0751 0.07567  0.0631
  State 3   0.000   0.000  0.8898 0.08342  0.0267
  State 4   0.000   0.000  0.0000 0.78738  0.2126
  State 5   0.000   0.000  0.0000 0.00136  0.9986

Emission probabilities :
           symbol_names
state_names 0 1       2     3       4      5     6      7
    State 1 1 0 0.00000 0.000 0.00000 0.0000 0.000 0.0000
    State 2 1 0 0.00000 0.000 0.00000 0.0000 0.000 0.0000
    State 3 0 1 0.00000 0.000 0.00000 0.0000 0.000 0.0000
    State 4 0 0 0.00195 0.992 0.00581 0.0000 0.000 0.0000
    State 5 0 0 0.21508 0.000 0.00000 0.0246 0.713 0.0474

R> BIC(sc_fit$model)
```

```
[1] 34176.02
```

As a multichannel example we fit a 5-state model for the 3-channel data. Emission probabilities are now given as a list of three emission matrices, one for each channel. The `alphabet` function from the **TraMineR** package can be used to check the order of the observed states – the same order is used in the `build` functions. Here we construct a left-to-right model where transitions to earlier states are not allowed, so the transition matrix is upper-triangular. This seems like a valid option from a life-course prespective. Also, in the previous single-channel model of the same data the transition matrix was estimated almost upper triangular. We also give names for channels – these are used when printing and plotting the model.

We estimate model parameters using the local step with the default L-BFGS algorithm using parallel computation with 4 threads.

```
R> mc_init <- c(0.9, 0.05, 0.02, 0.02, 0.01)
R>
R> mc_trans <- matrix(
+    c(0.80, 0.10, 0.05, 0.03, 0.02,
+      0,    0.90, 0.05, 0.03, 0.02,
+      0,       0, 0.90, 0.07, 0.03,
+      0,       0,    0, 0.90, 0.10,
+      0,       0,    0,    0,    1),
+    nrow = 5, ncol = 5, byrow = TRUE)
R>
R> mc_emiss_marr <- matrix(
+    c(0.90, 0.05, 0.05,
+      0.90, 0.05, 0.05,
+      0.05, 0.90, 0.05,
+      0.05, 0.90, 0.05,
+      0.30, 0.30, 0.40),
+    nrow = 5, ncol = 3, byrow = TRUE)
R>
R> mc_emiss_child <- matrix(
+    c(0.9, 0.1,
+      0.9, 0.1,
+      0.1, 0.9,
+      0.1, 0.9,
+      0.5, 0.5),
+    nrow = 5, ncol = 2, byrow = TRUE)
R>
R> mc_emiss_left <- matrix(
+    c(0.9, 0.1,
+      0.1, 0.9,
+      0.1, 0.9,
+      0.1, 0.9,
+      0.5, 0.5),
+    nrow = 5, ncol = 2, byrow = TRUE)
R>
```

```
R> mc_initmod <- build_hmm(
+    observations = list(marr_seq, child_seq, left_seq),
+    initial_probs = mc_init, transition_probs = mc_trans,
+    emission_probs = list(mc_emiss_marr, mc_emiss_child, mc_emiss_left),
+    channel_names = c("Marriage", "Parenthood", "Residence"))
R>
R> # For CRAN vignette: load the estimated model object for speed-up
R> data("hmm_biofam")
R> # mc_fit <- fit_model(mc_initmod, em_step = FALSE, local_step = TRUE,
R> #   threads = 4)
```

We store the model as a separate object for ease of use and then compute BIC.

```
R> # Vignette: already loaded hmm_biofam
R> #hmm_biofam <- mc_fit$model
R> BIC(hmm_biofam)

[1] 28842.7
```

4.3. Clustering and mixture hidden Markov models

When fitting mixture hidden Markov models, the starting values are given as lists, with one component per cluster. For multichannel data, emission probabilities are given as a list of lists. Here we fit a model for two clusters with 5 and 4 hidden states. For the cluster with five states we use the same starting values as for the multichannel HMM described earlier. Covariates are defined with the usual `formula` and `data` arguments.

Here we fit a model using 100 random restarts of the EM algorithm followed by the local L-BFGS method. Again we use parallel computation.

```
R> mc_init2 <- c(0.9, 0.05, 0.03, 0.02)
R>
R> mc_trans2 <- matrix(
+    c(0.85, 0.05, 0.05, 0.05,
+      0,    0.90, 0.05, 0.05,
+      0,       0, 0.95, 0.05,
+      0,       0,    0,    1),
+    nrow = 4, ncol = 4, byrow = TRUE)
R>
R> alphabet(marr_seq)

[1] "single"   "married"  "divorced"

R> mc_emiss_marr2 <- matrix(
+    c(0.90, 0.05, 0.05,
+      0.90, 0.05, 0.05,
```

```
+      0.05, 0.85, 0.10,
+      0.05, 0.80, 0.15),
+    nrow = 4, ncol = 3, byrow = TRUE)
R>
R> alphabet(child_seq)

[1] "childless" "children"

R> mc_emiss_child2 <- matrix(
+    c(0.9, 0.1,
+      0.5, 0.5,
+      0.5, 0.5,
+      0.5, 0.5),
+    nrow = 4, ncol = 2, byrow = TRUE)
R>
R> alphabet(left_seq)

[1] "with parents" "left home"

R> mc_emiss_left2 <- matrix(
+    c(0.9, 0.1,
+      0.5, 0.5,
+      0.5, 0.5,
+      0.5, 0.5),
+    nrow = 4, ncol = 2, byrow = TRUE)
R>
R> init_mhmm <- build_mhmm(
+    observations = list(marr_seq, child_seq, left_seq),
+    initial_probs = list(mc_init, mc_init2),
+    transition_probs = list(mc_trans, mc_trans2),
+    emission_probs = list(list(mc_emiss_marr, mc_emiss_child, mc_emiss_left),
+      list(mc_emiss_marr2, mc_emiss_child2, mc_emiss_left2)),
+    formula = ~sex + birthyr, data = biofam3c$covariates,
+    cluster_names = c("Cluster 1", "Cluster 2"),
+    channel_names = c("Marriage", "Parenthood", "Residence"))
R>
R> # Vignette: One thread and less restarts
R> set.seed(1001)
R> mhmm_fit <- fit_model(
+    init_mhmm, local_step = TRUE, threads = 1,
+    control_em = list(restart = list(times = 10)))
R> mhmm <- mhmm_fit$model
```

The `summary` method automatically computes some features for a MHMM, e.g. standard errors for covariates and prior and posterior cluster probabilities for subjects. A `print` method shows some summaries of these: estimates and standard errors for covariates (see section 2.3),

log-likelihood and BIC, and information on most probable clusters and prior probabilities. Parameter estimates for transitions, emissions, and initial probabilities are omitted by default. The classification table shows mean probabilities of belonging to each cluster by the most probable cluster (defined from posterior cluster probabilities). A good model shoud have values close to 1 on the diagonal.

```
R> summary(mhmm, conditional_se = FALSE)

Covariate effects :
Cluster 1 is the reference.

Cluster 2 :
             Estimate  Std. error
(Intercept)   99.3275    12.52342
sexwoman       0.1767     0.14137
birthyr       -0.0522     0.00646

Log-likelihood: -12965.93   BIC: 26575.01

Means of prior cluster probabilities :
Cluster 1 Cluster 2
    0.857     0.143

Most probable clusters :
            Cluster 1  Cluster 2
count            1748        252
proportion      0.874      0.126

Classification table :
Mean cluster probabilities (in columns) by the most probable cluster (rows)

          Cluster 1 Cluster 2
Cluster 1    0.9784    0.0216
Cluster 2    0.0125    0.9875
```

4.4. Visualizing hidden Markov models

The figures in section 3.3 illustrate the five-state multichannel HMM fitted in section 4.2. A basic HMM graph is easily called with the `plot` method.

```
R> plot(hmm_biofam)
```

A simple default plot is a convenient way of visualizing the models during analysis process, but for publishing it is often better to modify the plot to get an output that best illustrates

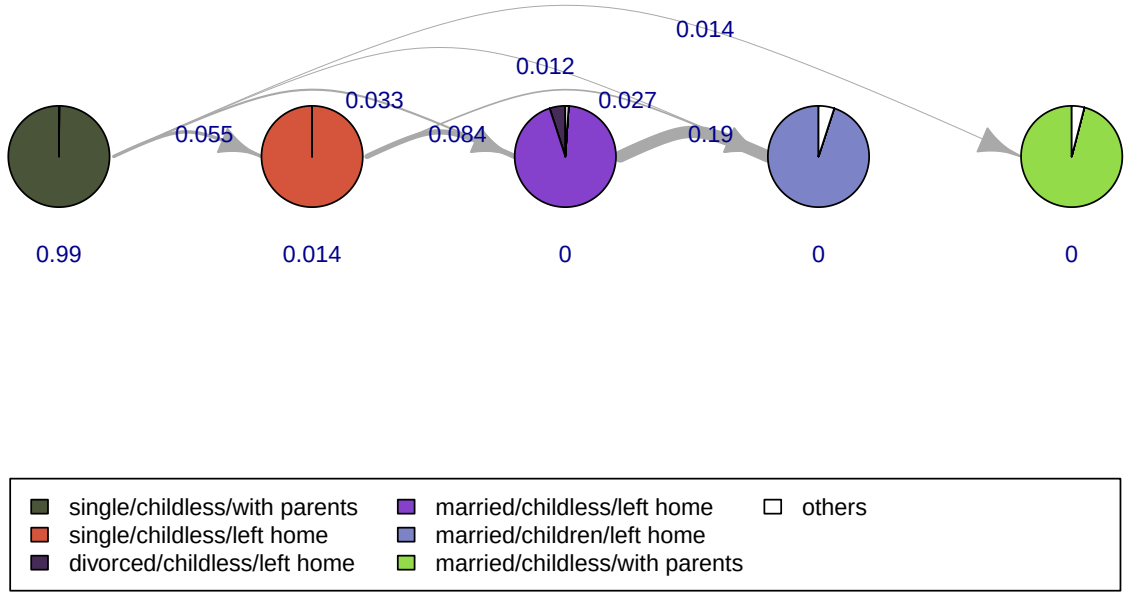

Figure 8: A default plot of a hidden Markov model.

the structure of the model in hand. Figure 4 and Figure 5 show two variations of the same model.

Figure 4: HMM plot with modifications

In Figure 4 we draw larger vertices, control the distances of initial probabilities (vertex labels), set the curvatures of the edges, give a more descriptive label for the combined slices and give less space for the legend.

```
R> plot(hmm_biofam, vertex.size = 50, vertex.label.dist = 1.5,
+    edge.curved = c(0, 0.8, -0.8, 0.8, 0, 0.8, 0),
+    legend.prop = 0.3, combined.slice.label = "States with prob. < 0.05")
```

Figure 5: HMM plot with a different layout

Here we position the vertices using given coordinates. Coordinates are given in a two-column matrix, with x coordinates in the first column and y coordinates in the second. Arguments `xlim` and `ylim` set the lengths of the axes and `rescale = FALSE` prevents rescaling the coordinates to the $[-1, 1] \times [-1, 1]$ interval (the default). We modify the positions of initial probabilities, fix edge widths to 1, reduce the size of the arrows in edges, position legend on top of the figure, and print labels in two columns in the legend. Parameter values are shown with one significant digit. All emission probabilities are shown regardless of their value (`combine.slices = 0`).

New colors are set from the ready-defined `colorpalette` data. The **seqHMM** package uses these palettes when determining colors automatically, e.g. in the `mc_to_sc` function. Since here there are 10 combined states, the default color palette is number 10. For getting different colors, we choose the ten first colors from palette number 14.

```
R> plot(hmm_biofam, layout = matrix(c(1, 3, 3, 5,  3,
+                                      0, 0, 1, 0, -1), ncol = 2),
+    xlim = c(0.5, 5.5), ylim = c(-1.5,1.5), rescale = FALSE,
+    vertex.size=50, edge.curved = FALSE, edge.width = 1, edge.arrow.size = 1,
+    vertex.label.pos = c(pi, pi/2, -pi/2, 0, pi/2),
+    withlegend = "top", legend.prop = 0.3, ncol.legend = 2,
+    label.signif = 1, combine.slices = 0, cpal = colorpalette[[30]][c(14:5)])
```

Figure 6: `ssplot` *for HMM object*

Plotting observed and hidden state sequences is easy with the `ssplot` function: the function accepts an `hmm` object instead of (a list of) `stslist`s. If hidden state paths are not provided, the function automatically computes them when needed.

```
R> ssplot(hmm_biofam, plots = "both", type = "I", sortv = "mds.hidden",
+    xtlab = 15:30, xlab = "Age", title = "Observed and hidden state sequences")
```

4.5. Visualizing mixture hidden Markov models

Objects of class `mhmm` have similar plotting methods to `hmm` objects. The default way of visualizing a model is to plot in an interactive mode, where the model for each cluster is plotted separately. Another option is a combined plot with all models in one plot, although it can be difficult to fit several graphs and legends in one figure.

Figure 9 illustrates the MHMM fitted in section 4.3. By setting `interactive = FALSE` and `nrow = 2` we tell to plot graphs in a grid with two rows. The rest of the arguments are similar to basic HMM plotting and apply for all the graphs.

```
R> plot(mhmm, interactive = FALSE, nrow = 2, legend.prop = 0.35,
+    cex.legend = 1.3, edge.label.cex = 1.3, vertex.label.cex = 1.3)
```

The equivalent of the `ssplot` function for `mhmm` objects is `mssplot`. It shows data and/or hidden paths one cluster at a time. The function is interactive if more than one cluster is plotted (thus omitted here). Subjects are allocated to clusters according to the posterior cluster probabilities.

```
R> mssplot(mhmm, ask = TRUE)
```

If the user wants more control than the default `mhmm` plotting functions offer, they can use the `separate_mhmm` function to convert a `mhmm` object into a list of separate `hmm` objects. These can then be plotted as any `hmm` objects, e.g. use `ssp` and `gridplot` for plotting sequences and hidden paths of each cluster into the same figure.

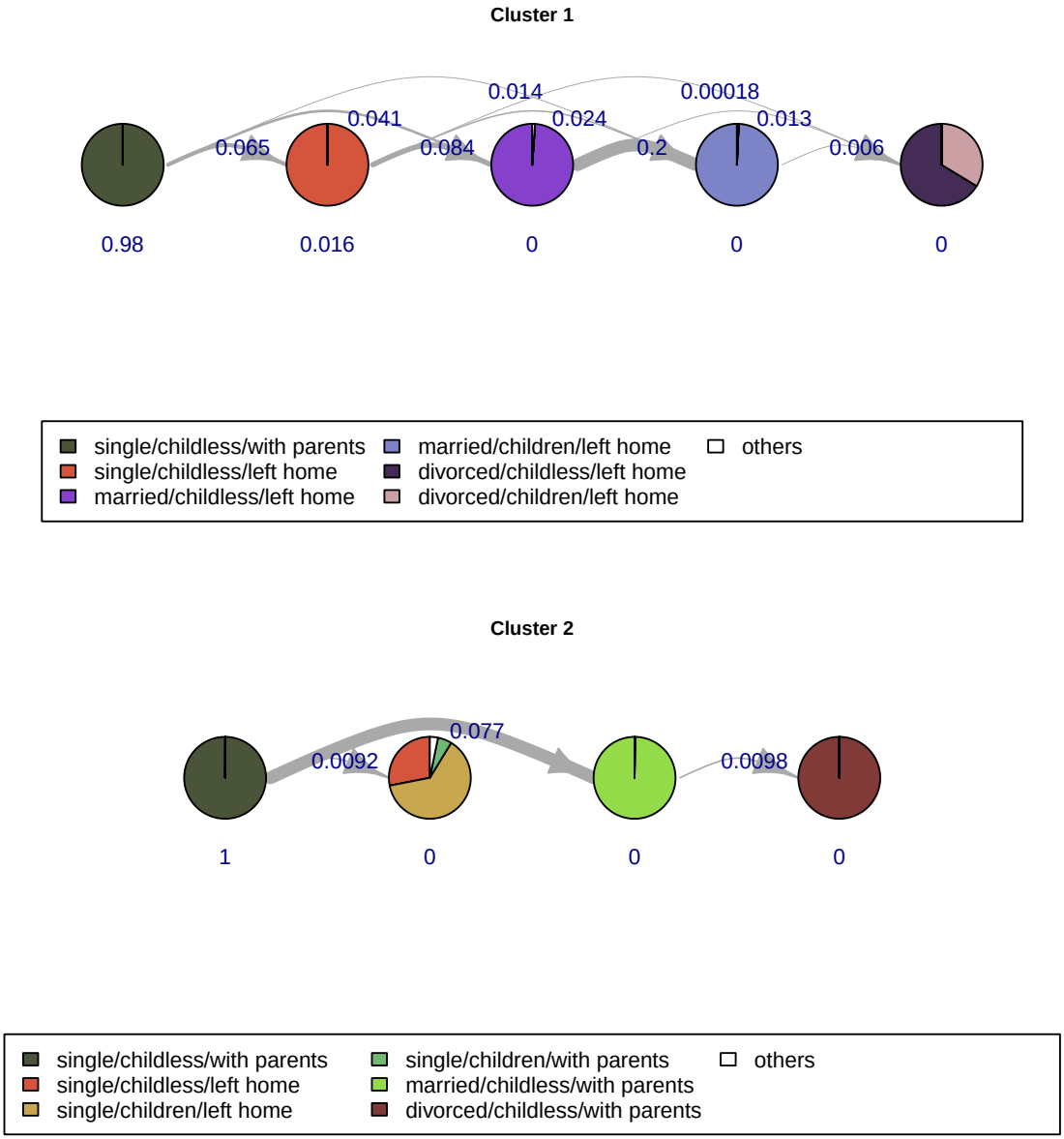

Figure 9: Plotting submodels of a MHMM with the `plot` method.

5. Conclusion

Hidden Markov models are useful in various longitudinal settings with categorical observations. They can be used for accounting measurement error in the observations (e.g. drug use as in Menard 2008), for detecting true unobservable states (e.g. different periods of the bipolar disorder as in Lopez 2008), and for compressing information accross several types of observations. The life course example of this paper serves as a simple illustration of such problem, where hidden states are regarded as general life stages during which individuals are more likely to encounter certain life events.

The **seqHMM** package is designed for analyzing categorical sequences with hidden Markov models and mixture hidden Markov models, as well as their restricted variants Markov models, mixture Markov models, and latent class models. It can handle many types of data from a single sequence to multiple multichannel sequences. Covariates can be included in MHMMs to explain cluster membership. The package also offers versatile plotting options for sequence data and HMMs, and can easily convert multichannel sequence data and models into single-channel representations.

Parameter estimation in (M)HMMs is often very sensitive to starting values. To deal with that, **seqHMM** offers several fitting options with global and local optimization using direct numerical estimation and the EM algorithm.

Almost all intensive computations are done in C++. The package also supports parallel computation.

Especially combined with the **TraMineR** package, **seqHMM** is designed to offer tools for the whole analysis process from data preparation and description to model fitting, evaluation, and visualization. In future we could develop MHMMs to deal with time-varying covariates and add an option to incorporate sampling weights for model estimation. Also the computational efficiency of the restricted variants of (M)HMMs such as latent class models could be improved by taking account of the restricted structure of those models in EM and log-likelihood computations.

A. Notations

Symbol	Meaning
Y_i	Observation sequences of subject $i, i = 1 \ldots, N$
$\mathbf{y}_{it}$	Observations of subject i at time $t, t = 1, \ldots, T$
y_{itc}	Observation of subject i at time t in channel $c, c = 1, \ldots, C$
$m_c \in \{1, \ldots, M_c\}$	Observed state space for channel c
z_{it}	Hidden state at time t for subject i
$s \in \{1, \ldots, S\}$	Hidden state space
$A = \{a_{sr}\}$	Transition matrix of size $S \times S$
$a_{sr} = P(z_t = r \vert z_{t-1} = s)$	Transition probability between hidden states s and r
$B_c = \{b_s(m_c)\}$	Emission matrix of size $S \times M_c$ for channel c
$b_s(m_c) = P(y_{itc} = m_c \vert z_{it} = s)$	Emission probability of observed state m_c in channel c given hidden state s
$b_s(\mathbf{y}_{it}) = b_s(y_{it1}) \cdots b_s(y_{itC})$	Joint emission probability of observations at time t in channels $1, \ldots, C$ given hidden state s
$\pi = (\pi_1, \ldots, \pi_S)$	Vector of initial probabilities
$\pi_s = P(z_1 = s)$	Initial probability of hidden state s
$\hat{z}_i(Y_i)$	The most probable hidden state sequence for subject i
$\mathbf{x}_i$	Covariates of subject i
$\mathcal{M}_k, k = 1, \ldots, K$	Submodel for cluster k (latent class/cluster)
w_{ik}	Probability of cluster k for subject i
β_k	Regression coefficients for cluster k
$\{\pi^k, A^k, B_1^k, \ldots, B_C^k, \beta_k\}$	Model parameters for cluster k

B. Forward–Backward Algorithm

The *forward variable*

$$\alpha_{it}(s) = P(\mathbf{y}_{i1}, \ldots, \mathbf{y}_{it}, z_t = s \vert \mathcal{M})$$

is the joint probability of partial observation sequences for subject i until time t and the hidden state s at time t given the model M. Let us denote $b_s(\mathbf{y}_{it}) = b_s(y_{it1}) \cdots b_s(y_{itC})$, the joint emission probability of observations at time t in channels $1, \ldots, C$ given hidden state s. The forward variable can be solved inductively:

1. Initialization
   $\alpha_{i1}(s) = \pi_s b_s(\mathbf{y}_{i1}), i = 1, \ldots, N, s = 1, \ldots, S$
2. Induction
   $\alpha_{i(t+1)}(r) = \left[\sum_{s=1}^{S} \alpha_{it}(s) a_{sr}\right] b_r(\mathbf{y}_{i(t+1)}), t = 1, \ldots, T-1, r = 1, \ldots, S$
3. Termination
   $P(Y_i \vert \mathcal{M}) = \sum_{s=1}^{S} \alpha_{iT}(s)$

The *backward variable*

$$\beta_{it}(s) = P(\mathbf{y}_{i(t+1)}, \ldots, \mathbf{y}_{iT} \vert z_t = s, \mathcal{M})$$

is the joint probability of the partial observation sequence after time t and hidden state s at time t given the model parameters M. (By convention we use the notion β for the backward

variable. This is not to be confused with the regression coefficients in the mixture HMM.) Also this can be solved inductively:

1. Initialization
   $\beta_{iT}(s) = 1, i = 1, \ldots, N, s = 1, \ldots, S$
2. Induction
   $\beta_{i(t+1)}(s) = \left[\sum_{r=1}^{S} a_{sr}\right] b_s(\mathbf{y}_{i(t+1)})\beta_{i(t+1)}(r), t = T-1, \ldots, 1, s = 1, \ldots, S$

C. Viterbi Algorithm

We define the score

$$\delta_{it}(s) = \max_{z_{i1}z_{i2}\cdots z_{i(t-1)}} P(z_{i1}\cdots z_{it} = s, \mathbf{y}_{i1}\cdots\mathbf{y}_{it}|\mathcal{M}), \tag{13}$$

which is the highest probability of the hidden state sequence up to time t ending in state s. By induction we have

$$\delta_{i(t+1)}(r) = \left[\max_s \delta_{it}(s)a_{sr}\right] \cdot b_r(\mathbf{y}_{i(t+1)}). \tag{14}$$

We collect the arguments maximizing equation (14) in an array $\psi_{it}(r)$ to keep track of the best hidden state sequence. The full Viterbi algorithm can be stated as follows:

1. Initialization
   $\delta_{i1}(s) = \pi_s b_s(\mathbf{y}_{i1}), s = 1, \ldots, S$
   $\psi_{i1}(s) = 0$
2. Recursion
   $\delta_{it}(r) = \max_{s=1,\ldots,S}(\delta_{i(t-1)}(s)a_{sr})b_h(\mathbf{y}_{it})$,
   $\psi_{it}(s) = \arg\max_{s=1,\ldots,S}(\delta_{i(t-1)}(s)a_{sr}), s = 1, \ldots, S; t = 2, \ldots, T$
3. Termination
   $\hat{P} = \max_{s=1,\ldots,S}(\delta_{iT}(s))$
   $\hat{z}_{iT} = \arg\max_{s=1,\ldots,S}(\delta_{iT}(s))$
4. Sequence backtracking
   $\hat{z}_{it} = \psi_{i(t+1)}(\hat{s}_{i(t+1)}), t = T-1, \ldots, 1.$

To avoid underflow error due to multiplying many small probabilities, the Viterbi algorithm can be computed in log space, i.e., calculating $\log(\delta_{it}(s))$.

References

Aisenbrey S, Fasang A (2010). "New Life for Old Ideas: The "Second Wave" of Sequence Analysis – Bringing the "Course" Back Into the Life Course." *Sociological Methods & Research*, **38**(3), 420–462. doi:10.1177/0049124109357532.

Bartolucci F, Pandolfi S (2015). ***LMest**: Latent Markov Models with and without Covariates*. R package version 2.1, URL http://CRAN.R-project.org/package=LMest.

Baum LE, Petrie T (1966). "Statistical Inference for Probabilistic Functions of Finite State Markov Chains." *The annals of mathematical statistics*, **67**(6), 1554–1563. URL http://www.jstor.org/stable/2238772?seq=1#page_scan_tab_contents.

Blanchard P, Bühlmann F, Gauthier JA (eds.) (2014). *Advances in Sequence Analysis: Theory, Method, Applications*. Springer New York Heidelberg Dordrecht London. doi:10.1007/978-3-319-04969-4.

Collins LM, Wugalter SE (1992). "Latent Class Models for Stage-Sequential Dynamic Latent Variables." *Multivariate Behavioral Research*, **27**(1), 131–157.

Csardi G, Nepusz T (2006). "The **igraph** Software Package for Complex Network Research." *InterJournal*, **Complex Systems**(1695). URL http://igraph.org.

Durbin R, Eddy S, Krogh A, Mitchison G (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, UK.

Eddelbuettel D (2013). *Seamless R and C++ Integration with **Rcpp***. Springer, New York. ISBN 978-1-4614-6867-7.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel D, Sanderson C (2014). "RcppArmadillo: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics and Data Analysis*, **71**, 1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Elzinga CH, Studer M (2014). "Spell Sequences, State Proximities, and Distance Metrics." *Sociological Methods & Research*, pp. 3–47.

Gabadinho A, Ritschard G, Müller NS, Studer M (2011). "Analyzing and Visualizing State Sequences in R with **TraMineR**." *Journal of Statistical Software*, **40**(4), 1–37. URL http://www.jstatsoft.org/v40/i04/paper.

Gauthier JA, Widmer ED, Bucher P, Notredame C (2009). "How Much Does It Cost? Optimization of Costs in Sequence Analysis of Social Science Data." *Sociological Methods & Research*, **38**(1), 197–231. doi:10.1177/0049124109342065.

Gauthier JA, Widmer ED, Bucher P, Notredame C (2010). "Multichannel Sequence Analysis Applied to Social Science Data." *Sociological Methodology*, **40**(1), 1–38. doi:10.1111/j.1467-9531.2010.01227.x.

Halpin B (2010). "Optimal Matching Analysis and Life-Course Data: The Importance of Duration." *Sociological Methods & Research*, **38**(3), 365–388. URL http://smr.sagepub.com/cgi/reprint/38/3/365.

Himmelmann L (2010). *HMM – Hidden Markov Models*. R Package Version 1.0, URL http://CRAN.R-project.org/package=HMM.

Hollister M (2009). "Is Optimal Matching Suboptimal?" *Sociological Methods & Research*, **38**(2), 235–264. doi:10.1177/0049124109346164.

Jackson CH (2011). "Multi-State Models for Panel Data: The **msm** Package for R." *Journal of Statistical Software*, **38**(8), 1–29. URL http://www.jstatsoft.org/v38/i08/.

Johnson SG (2014). *The NLopt Nonlinear Optimization Package*. URL http://ab-initio.mit.edu/nlopt.

Kucherenko S, Sytsko Y (2005). "Application of Deterministic Low-Discrepancy Sequences in Global Optimization." *Computational Optimization and Applications*, **30**(3), 297–318.

Lesnard L (2010). "Setting Cost in Optimal Matching to Uncover Contemporaneous Socio-Temporal Patterns." *Sociological Methods & Research*, **38**(3), 389–419. doi:10.1177/0049124110362526.

Liu DC, Nocedal J (1989). "On the Limited Memory BFGS Method for Large Scale Optimization." *Mathematical programming*, **45**(1-3), 503–528.

Lopez A (2008). *Markov Models for Longitudinal Course of Youth Bipolar Disorder*. ProQuest, Ann Arbor, MI. URL http://d-scholarship.pitt.edu/6524/1/LopezAdrianaApril23.pdf.

MacDonald IL, Zucchini W (1997). *Hidden Markov and Other Models for Discrete-Valued Time Series*, volume 110. CRC Press, Boca Raton, FL.

McVicar D, Anyadike-Danes M (2002). "Predicting Successful and Unsuccessful Transitions from School to Work by Using Sequence Methods." *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, **165**(2), 317–334. doi:10.1111/1467-985X.00641.

Menard S (ed.) (2008). *Latent Class Models in Longitudinal Research*. Elsevier, Burlington, MA.

Müller NS, Studer M, Ritschard G (2007). "Classification de Parcours de vie à l'Aide de l'Optimal Matching." *XIVe Rencontre de la Société francophone de classification (SFC 2007)*, pp. 157–160.

Neuwirth E (2014). *RColorBrewer: ColorBrewer Palettes*. R Package Version 1.1-2, URL http://CRAN.R-project.org/package=RColorBrewer.

Nocedal J (1980). "Updating Quasi-Newton Matrices with Limited Storage." *Mathematics of computation*, **35**(151), 773–782.

O'Connell J, Højsgaard S (2011). "Hidden Semi Markov Models for Multiple Observation Sequences: The **mhsmm** Package for R." *Journal of Statistical Software*, **39**(4), 1–22. URL http://www.jstatsoft.org/v39/i04/.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Rabiner L (1989). "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *Proceedings of the IEEE*, **77**(2), 257–286. doi:10.1109/5.18626.

Rinnooy Kan A, Timmer G (1987a). "Stochastic Global Optimization Methods Part I: Clustering Methods." *Mathematical programming*, **39**(1), 27–56.

Rinnooy Kan A, Timmer G (1987b). "Stochastic Global Optimization Methods Part II: Multi-Level Methods." *Mathematical Programming*, **39**(1), 57–78.

Turner R, Liu L (2014). ***hmm.discnp**: Hidden Markov Models with Discrete Non-Parametric Observation Distributions*. R Package Version 0.2-3, URL http://CRAN.R-project.org/package=hmm.discnp.

van de Pol F, Langeheine R (1990). "Mixed Markov Latent Class Models." *Sociological methodology*, **20**, 213–247. URL http://www.jstor.org/stable/271087.

Visser I, Speekenbrink M (2010). "**depmixS4**: An R-package for hidden Markov models." *Journal of Statistical Software*, **36**(7), 1–21.

Ypma J, Borchers HW, Eddelbuettel D (2014). ***nloptr**: R interface to NLopt*. R Package Version 1.0.4, URL http://CRAN.R-project.org/package=nloptr.

Affiliation:

Satu Helske
Department of Mathematics and Statistics
P.O.Box 35 (MaD)
FI-40014 University of Jyvaskyla
Finland
E-mail: satu.helske@jyu.fi
URL: http://users.jyu.fi/~samahels/