Package ‘seqHMM’

May 10, 2018

Title Hidden Markov Models for Life Sequences and Other Multivariate, Multichannel Categorical Time Series

Version 1.0.8-1

Date 2018-05-08

Author Jouni Helske, Satu Helske

Maintainer Jouni Helske <jouni.helske@iki.fi>

Description Designed for fitting hidden (latent) Markov models and mixture hidden Markov models for social sequence data and other categorical time series. Also some more restricted versions of these type of models are available: Markov models, mixture Markov models, and latent class models. The package supports models for one or multiple subjects with one or multiple parallel sequences (channels). External covariates can be added to explain cluster membership in mixture models. The package provides functions for evaluating and comparing models, as well as functions for easy plotting of multichannel sequence data and hidden Markov models. Models are estimated using maximum likelihood via the EM algorithm and/or direct numerical maximization with analytical gradients. All main algorithms are written in C++ with support for parallel computation.

LazyData true

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 3.2.0)

Imports gridBase, igraph, Matrix, nloptr, numDeriv, Rcpp (>= 0.11.3), TraMineR (>= 1.8-8), graphics, grDevices, grid, methods, stats, utils

Suggests MASS, nnet, knitr

SystemRequirements C++11

License GPL (>= 2)

Encoding UTF-8

BugReports https://github.com/helske/seqHMM/issues

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-05-10 12:17:29 UTC

R topics documented:

biofam3c *Three-channel biofam data*

Description

Biofam data from the TraMineR package converted into three channels.

Format

A list including three sequence data sets for 2000 individuals with 16 state variables, and a separate data frame with 1 id variable, 8 covariates, and 2 weights variables.

Details

This data is constructed from the `biofam` data in the TraMineR package. Here the original state sequences are converted into three separate data sets: `children`, `married`, and `left`. These include the corresponding life states from age 15 to 30: `childless` or (having) `children`; `single`, `married`, or `divorced`; and (living) `with parents` or `left home`.

Note that the `divorced` state does not give information on parenthood or residence, so a guess is made based on preceeding states.

The fourth data frame `covariates` is a collection of additional variables from the original data:

`idhous`	id
`sex`	sex
`birthyr`	birth year
`nat_1_02`	first nationality
`plingu02`	language of questionnaire
`p02r01`	religion
`p02r04`	religious participation
`cspfaj`	father's social status
`cspmoj`	mother's social status
`wp00tbgp`	weights inflating to the Swiss population
`wp00tbgs`	weights respecting sample size

The data is loaded by calling `data(biofam3c)`. It was built using following code:

```
data("biofam" , package = "TraMineR")
biofam3c <- with(biofam, {

## Building one channel per type of event left, children or married
bf <- as.matrix(biofam[, 10:25])
children <- bf == 4 | bf == 5 | bf == 6
married <- bf == 2 | bf == 3 | bf == 6
left <- bf == 1 | bf == 3 | bf == 5 | bf == 6 | bf == 7

children[children == TRUE] <- "children"
```

```
children[children == FALSE] <- "childless"
# Divorced parents
div <- bf[(rowSums(bf == 7) > 0 & rowSums(bf == 5) > 0) |
            (rowSums(bf == 7) > 0 & rowSums(bf == 6) > 0),]
children[rownames(bf)

married[married == TRUE] <- "married"
married[married == FALSE] <- "single"
married[bf == 7] <- "divorced"

left[left == TRUE] <- "left home"
left[left == FALSE] <- "with parents"
# Divorced living with parents (before divorce)
wp <- bf[(rowSums(bf == 7) > 0 & rowSums(bf == 2) > 0 &
          rowSums(bf == 3) == 0 & rowSums(bf == 5) == 0 &
          rowSums(bf == 6) == 0) |
         (rowSums(bf == 7) > 0 & rowSums(bf == 4) > 0 &
          rowSums(bf == 3) == 0 & rowSums(bf == 5) == 0 &
          rowSums(bf == 6) == 0), ]
left[rownames(bf)

list("children" = children, "married" = married, "left" = left,
  "covariates" = biofam[, c(1:9, 26:27)])
})
```

Source

biofam data constructed from the Swiss Household Panel www.swisspanel.ch

References

Müller, N. S., M. Studer, G. Ritschard (2007). Classification de parcours de vie à l'aide de l'optimal matching. In *XIVe Rencontre de la Société francophone de classification (SFC 2007), Paris, 5 - 7 septembre 2007*, pp. 157–160.

build_hmm — *Build a Hidden Markov Model*

Description

Function build_hmm constructs a hidden Markov model object of class hmm.

Usage

```
build_hmm(observations, n_states, transition_probs, emission_probs,
  initial_probs, state_names = NULL, channel_names = NULL, ...)
```

Arguments

`observations`	An `stslist` object (see `seqdef`) containing the sequences, or a list of such objects (one for each channel).
`n_states`	A scalar giving the number of hidden states (not used if starting values for model parameters are given with `initial_probs`, `transition_probs`, and `emission_probs`).
`transition_probs`	A matrix of transition probabilities.
`emission_probs`	A matrix of emission probabilities or a list of such objects (one for each channel). Emission probabilities should follow the ordering of the alphabet of observations (`alphabet(observations)`, returned as `symbol_names`).
`initial_probs`	A vector of initial state probabilities.
`state_names`	A list of optional labels for the hidden states. If `NULL`, the state names are taken from the row names of the transition matrix. If this is also `NULL`, numbered states are used.
`channel_names`	A vector of optional names for the channels.
`...`	Additional arguments to `simulate_transition_probs`.

Details

The returned model contains some attributes such as `nobs` and `df`, which define the number of observations in the model and the number of estimable model parameters, used in computing BIC. When computing `nobs` for a multichannel model with C channels, each observed value in a single channel amounts to $1/C$ observation, i.e. a fully observed time point for a single sequence amounts to one observation. For the degrees of freedom `df`, zero probabilities of the initial model are defined as structural zeroes.

Value

Object of class `hmm` with the following elements:

`observations` State sequence object or a list of such objects containing the data.

`transition_probs` A matrix of transition probabilities.

`emission_probs` A matrix or a list of matrices of emission probabilities.

`initial_probs` A vector of initial probabilities.

`state_names` Names for hidden states.

`symbol_names` Names for observed states.

`channel_names` Names for channels of sequence data.

`length_of_sequences` (Maximum) length of sequences.

`n_sequences` Number of sequences.

`n_symbols` Number of observed states (in each channel).

`n_states` Number of hidden states.

`n_channels` Number of channels.

See Also

`fit_model` for estimating model parameters; and `plot.hmm` for plotting hmm objects.

Examples

```
# Single-channel data

data("mvad", package = "TraMineR")

mvad_alphabet <- c("employment", "FE", "HE", "joblessness", "school",
                   "training")
mvad_labels <- c("employment", "further education", "higher education",
                 "joblessness", "school", "training")
mvad_scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad_seq <- seqdef(mvad, 17:86, alphabet = mvad_alphabet, states = mvad_scodes,
                   labels = mvad_labels, xtstep = 6)

# Initializing an HMM with 4 hidden states, random starting values
init_hmm_mvad1 <- build_hmm(observations = mvad_seq, n_states = 4)

# Starting values for the emission matrix
emiss <- matrix(NA, nrow = 4, ncol = 6)
emiss[1,] <- seqstatf(mvad_seq[, 1:12])[, 2] + 1
emiss[2,] <- seqstatf(mvad_seq[, 13:24])[, 2] + 1
emiss[3,] <- seqstatf(mvad_seq[, 25:48])[, 2] + 1
emiss[4,] <- seqstatf(mvad_seq[, 49:70])[, 2] + 1
emiss <- emiss / rowSums(emiss)

# Starting values for the transition matrix

tr <- matrix(
  c(0.80, 0.10, 0.05, 0.05,
    0.05, 0.80, 0.10, 0.05,
    0.05, 0.05, 0.80, 0.10,
    0.05, 0.05, 0.10, 0.80),
  nrow=4, ncol=4, byrow=TRUE)

# Starting values for initial state probabilities
init <- c(0.3, 0.3, 0.2, 0.2)

# HMM with own starting values
init_hmm_mvad2 <- build_hmm(
  observations = mvad_seq, transition_probs = tr,
  emission_probs = emiss, initial_probs = init)

#########################################


# Multichannel data

# Three-state three-channel hidden Markov model
# See ?hmm_biofam for a five-state version

data("biofam3c")

# Building sequence objects
marr_seq <- seqdef(biofam3c$married, start = 15,
  alphabet = c("single", "married", "divorced"))
child_seq <- seqdef(biofam3c$children, start = 15,
  alphabet = c("childless", "children"))
left_seq <- seqdef(biofam3c$left, start = 15,
  alphabet = c("with parents", "left home"))

# Define colors
attr(marr_seq, "cpal") <- c("violetred2", "darkgoldenrod2", "darkmagenta")
attr(child_seq, "cpal") <- c("darkseagreen1", "coral3")
attr(left_seq, "cpal") <- c("lightblue", "red3")

# Left-to-right HMM with 3 hidden states and random starting values
set.seed(1010)
init_hmm_bf1 <- build_hmm(
  observations = list(marr_seq, child_seq, left_seq),
  n_states = 3, left_right = TRUE, diag_c = 2)


# Starting values for emission matrices

emiss_marr <- matrix(NA, nrow = 3, ncol = 3)
emiss_marr[1,] <- seqstatf(marr_seq[, 1:5])[, 2] + 1
emiss_marr[2,] <- seqstatf(marr_seq[, 6:10])[, 2] + 1
emiss_marr[3,] <- seqstatf(marr_seq[, 11:16])[, 2] + 1
emiss_marr <- emiss_marr / rowSums(emiss_marr)

emiss_child <- matrix(NA, nrow = 3, ncol = 2)
emiss_child[1,] <- seqstatf(child_seq[, 1:5])[, 2] + 1
emiss_child[2,] <- seqstatf(child_seq[, 6:10])[, 2] + 1
emiss_child[3,] <- seqstatf(child_seq[, 11:16])[, 2] + 1
emiss_child <- emiss_child / rowSums(emiss_child)

emiss_left <- matrix(NA, nrow = 3, ncol = 2)
emiss_left[1,] <- seqstatf(left_seq[, 1:5])[, 2] + 1
emiss_left[2,] <- seqstatf(left_seq[, 6:10])[, 2] + 1
emiss_left[3,] <- seqstatf(left_seq[, 11:16])[, 2] + 1
emiss_left <- emiss_left / rowSums(emiss_left)

# Starting values for transition matrix
trans <- matrix(c(0.9, 0.07, 0.03,
                  0,    0.9,  0.1,
                  0,      0,    1),
  nrow = 3, ncol = 3, byrow = TRUE)

# Starting values for initial state probabilities
inits <- c(0.9, 0.09, 0.01)
```

```
# HMM with own starting values
init_hmm_bf2 <- build_hmm(
  observations = list(marr_seq, child_seq, left_seq),
  transition_probs = trans,
  emission_probs = list(emiss_marr, emiss_child, emiss_left),
  initial_probs = inits)
```

build_lcm *Build a Latent Class Model*

Description

Function build_lcm is a shortcut for constructing a latent class model as a restricted case of an mhmm object.

Usage

```
build_lcm(observations, n_clusters, emission_probs, formula, data, coefficients,
  cluster_names = NULL, channel_names = NULL)
```

Arguments

observations	An stslist object (see seqdef) containing the sequences, or a list of such objects (one for each channel).
n_clusters	A scalar giving the number of clusters/submodels (not used if starting values for model parameters are given with emission_probs).
emission_probs	A matrix containing emission probabilities for each class by rows, or in case of multichannel data a list of such matrices. Note that the matrices must have dimensions k x s where k is the number of latent classes and s is the number of unique symbols (observed states) in the data. Emission probabilities should follow the ordering of the alphabet of observations (alphabet(observations), returned as symbol_names).
formula	Covariates as an object of class formula, left side omitted.
data	An optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(formula).
coefficients	An optional kxl matrix of regression coefficients for time-constant covariates for mixture probabilities, where l is the number of clusters and k is the number of covariates. A logit-link is used for mixture probabilities. The first column is set to zero.
cluster_names	A vector of optional names for the classes/clusters.
channel_names	A vector of optional names for the channels.

Value

Object of class mhmm with the following elements:

`observations` State sequence object or a list of such containing the data.

`transition_probs` A matrix of transition probabilities.

`emission_probs` A matrix or a list of matrices of emission probabilities.

`initial_probs` A vector of initial probabilities.

`coefficients` A matrix of parameter coefficients for covariates (covariates in rows, clusters in columns).

`X` Covariate values for each subject.

`cluster_names` Names for clusters.

`state_names` Names for hidden states.

`symbol_names` Names for observed states.

`channel_names` Names for channels of sequence data

`length_of_sequences` (Maximum) length of sequences.

`n_sequences` Number of sequences.

`n_symbols` Number of observed states (in each channel).

`n_states` Number of hidden states.

`n_channels` Number of channels.

`n_covariates` Number of covariates.

`n_clusters` Number of clusters.

See Also

`fit_model` for estimating model parameters; `summary.mhmm` for a summary of a mixture model; `separate_mhmm` for organizing an `mhmm` object into a list of separate `hmm` objects; and `plot.mhmm` for plotting mixture models.

Examples

```
# Simulate observations from two classes
set.seed(123)
obs <- seqdef(rbind(
  matrix(sample(letters[1:3], 5000, TRUE, prob = c(0.1, 0.6, 0.3)), 500, 10),
  matrix(sample(letters[1:3], 2000, TRUE, prob = c(0.4, 0.4, 0.2)), 200, 10)))

# Initialize the model
set.seed(9087)
model <- build_lcm(obs, n_clusters = 2)

# Estimate model parameters
fit <- fit_model(model)

# How many of the observations were correctly classified:
sum(summary(fit$model)$most_probable_cluster == rep(c("Class 2", "Class 1"), times = c(500, 200)))
```

```
############################################################
## Not run:
# LCM for longitudinal data

# Define sequence data
data("mvad", package = "TraMineR")
mvad_alphabet <- c("employment", "FE", "HE", "joblessness", "school",
  "training")
mvad_labels <- c("employment", "further education", "higher education",
  "joblessness", "school", "training")
mvad_scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad_seq <- seqdef(mvad, 17:86, alphabet = mvad_alphabet, states = mvad_scodes,
  labels = mvad_labels, xtstep = 6)

# Initialize the LCM with random starting values
set.seed(7654)
init_lcm_mvad1 <- build_lcm(observations = mvad_seq,
  n_clusters = 2, formula = ~male, data = mvad)

# Own starting values for emission probabilities
emiss <- rbind(rep(1/6, 6), rep(1/6, 6))

# LCM with own starting values
init_lcm_mvad2 <- build_lcm(observations = mvad_seq,
  emission_probs = emiss, formula = ~male, data = mvad)

# Estimate model parameters (EM algorithm with random restarts)
lcm_mvad <- fit_model(init_lcm_mvad1,
  control_em = list(restart = list(times = 5)))$model

# Plot the LCM
plot(lcm_mvad, interactive = FALSE, ncol = 2)

###################################################################

# Binomial regression (comparison to glm)

require("MASS")
data("birthwt")

model <- build_lcm(
  observations = seqdef(birthwt$low), emission_probs = diag(2),
  formula = ~age + lwt + smoke + ht, data = birthwt)
fit <- fit_model(model)
summary(fit$model)
summary(glm(low ~ age + lwt + smoke + ht, binomial, data = birthwt))


# Multinomial regression (comparison to multinom)

require("nnet")
```

```
set.seed(123)
n <- 100
X <- cbind(1, x1 = runif(n, 0, 1), x2 =  runif(n, 0, 1))
coefs <- cbind(0,c(-2, 5, -2), c(0, -2, 2))
pr <- exp(X %*% coefs)  + rnorm(n*3)
pr <- pr/rowSums(pr)
y <- apply(pr, 1, which.max)
table(y)

model <- build_lcm(
  observations = seqdef(y), emission_probs = diag(3),
  formula = ~x1 + x2,  data = data.frame(X[, -1]))
fit <- fit_model(model)
summary(fit$model)
summary(multinom(y ~ x1 + x2, data = data.frame(X[,-1])))

## End(Not run)
```

`build_mhmm` *Build a Mixture Hidden Markov Model*

Description

Function `build_mhmm` constructs a mixture hidden Markov model object of class `mhmm`.

Usage

```
build_mhmm(observations, n_states, transition_probs, emission_probs,
  initial_probs, formula, data, coefficients, cluster_names = NULL,
  state_names = NULL, channel_names = NULL, ...)
```

Arguments

`observations`	An `stslist` object (see `seqdef`) containing the sequences, or a list of such objects (one for each channel).
`n_states`	A numerical vector giving the number of hidden states in each submodel (not used if starting values for model parameters are given with `initial_probs`, `transition_probs`, and `emission_probs`).
`transition_probs`	A list of matrices of transition probabilities for the submodel of each cluster.
`emission_probs`	A list which contains matrices of emission probabilities or a list of such objects (one for each channel) for the submodel of each cluster. Note that the matrices must have dimensions mxs where m is the number of hidden states and s is the number of unique symbols (observed states) in the data. Emission probabilities should follow the ordering of the alphabet of observations (`alphabet(observations)`, returned as `symbol_names`).
`initial_probs`	A list which contains vectors of initial state probabilities for the submodel of each cluster.

`formula`	Covariates as an object of class `formula`, left side omitted.
`data`	An optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`.
`coefficients`	An optional kxl matrix of regression coefficients for time-constant covariates for mixture probabilities, where l is the number of clusters and k is the number of covariates. A logit-link is used for mixture probabilities. The first column is set to zero.
`cluster_names`	A vector of optional names for the clusters.
`state_names`	A list of optional labels for the hidden states. If `NULL`, the state names are taken as row names of transition matrices. If this is also `NULL`, numbered states are used.
`channel_names`	A vector of optional names for the channels.
`...`	Additional arguments to `simulate_transition_probs`.

Details

The returned model contains some attributes such as `nobs` and `df`, which define the number of observations in the model and the number of estimable model parameters, used in computing BIC. When computing `nobs` for a multichannel model with C channels, each observed value in a single channel amounts to $1/C$ observation, i.e. a fully observed time point for a single sequence amounts to one observation. For the degrees of freedom `df`, zero probabilities of the initial model are defined as structural zeroes.

Value

Object of class `mhmm` with following elements:

`observations` State sequence object or a list of such containing the data.

`transition_probs` A matrix of transition probabilities.

`emission_probs` A matrix or a list of matrices of emission probabilities.

`initial_probs` A vector of initial probabilities.

`coefficients` A matrix of parameter coefficients for covariates (covariates in rows, clusters in columns).

`X` Covariate values for each subject.

`cluster_names` Names for clusters.

`state_names` Names for hidden states.

`symbol_names` Names for observed states.

`channel_names` Names for channels of sequence data

`length_of_sequences` (Maximum) length of sequences.

`n_sequences` Number of sequences.

`n_symbols` Number of observed states (in each channel).

`n_states` Number of hidden states.

`n_channels` Number of channels.

`n_covariates` Number of covariates.

`n_clusters` Number of clusters.

See Also

fit_model for fitting mixture Hidden Markov models; summary.mhmm for a summary of a MHMM; separate_mhmm for reorganizing a MHMM into a list of separate hidden Markov models; and plot.mhmm for plotting mhmm objects.

Examples

```
data("biofam3c")

## Building sequence objects
marr_seq <- seqdef(biofam3c$married, start = 15,
  alphabet = c("single", "married", "divorced"))
child_seq <- seqdef(biofam3c$children, start = 15,
  alphabet = c("childless", "children"))
left_seq <- seqdef(biofam3c$left, start = 15,
  alphabet = c("with parents", "left home"))

## Choosing colors
attr(marr_seq, "cpal") <- c("#AB82FF", "#E6AB02", "#E7298A")
attr(child_seq, "cpal") <- c("#66C2A5", "#FC8D62")
attr(left_seq, "cpal") <- c("#A6CEE3", "#E31A1C")

## MHMM with random starting values, no covariates
set.seed(468)
init_mhmm_bf1 <- build_mhmm(
  observations = list(marr_seq, child_seq, left_seq),
  n_states = c(4, 4, 6),
  channel_names = c("Marriage", "Parenthood", "Residence"))


## Starting values for emission probabilities

# Cluster 1
B1_marr <- matrix(
  c(0.8, 0.1, 0.1, # High probability for single
    0.8, 0.1, 0.1,
    0.3, 0.6, 0.1, # High probability for married
    0.3, 0.3, 0.4), # High probability for divorced
  nrow = 4, ncol = 3, byrow = TRUE)

B1_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.9, 0.1,
    0.9, 0.1),
  nrow = 4, ncol = 2, byrow = TRUE)

B1_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9, # High probability for having left home
    0.1, 0.9,
    0.1, 0.9),
  nrow = 4, ncol = 2, byrow = TRUE)

# Cluster 2

B2_marr <- matrix(
  c(0.8, 0.1, 0.1, # High probability for single
    0.8, 0.1, 0.1,
    0.1, 0.8, 0.1, # High probability for married
    0.7, 0.2, 0.1),
  nrow = 4, ncol = 3, byrow = TRUE)

B2_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.9, 0.1,
    0.1, 0.9),
  nrow = 4, ncol = 2, byrow = TRUE)

B2_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9,
    0.1, 0.9,
    0.1, 0.9),
  nrow = 4, ncol = 2, byrow = TRUE)

# Cluster 3
B3_marr <- matrix(
  c(0.8, 0.1, 0.1, # High probability for single
    0.8, 0.1, 0.1,
    0.8, 0.1, 0.1,
    0.1, 0.8, 0.1, # High probability for married
    0.3, 0.4, 0.3,
    0.1, 0.1, 0.8), # High probability for divorced
  nrow = 6, ncol = 3, byrow = TRUE)

B3_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.5, 0.5,
    0.5, 0.5,
    0.5, 0.5,
    0.1, 0.9),
  nrow = 6, ncol = 2, byrow = TRUE)


B3_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9,
    0.5, 0.5,
    0.5, 0.5,
    0.1, 0.9,
    0.1, 0.9),
  nrow = 6, ncol = 2, byrow = TRUE)

# Starting values for transition matrices
A1 <- matrix(
  c(0.80, 0.16, 0.03, 0.01,
    0,    0.90, 0.07, 0.03,
    0,    0,    0.90, 0.10,
    0,    0,    0,       1),
  nrow = 4, ncol = 4, byrow = TRUE)

A2 <- matrix(
  c(0.80, 0.10, 0.05, 0.03, 0.01, 0.01,
    0,    0.70, 0.10, 0.10, 0.05, 0.05,
    0,    0,    0.85, 0.01, 0.10, 0.04,
    0,    0,    0,    0.90, 0.05, 0.05,
    0,    0,    0,    0,    0.90, 0.10,
    0,    0,    0,    0,    0,       1),
  nrow = 6, ncol = 6, byrow = TRUE)

# Starting values for initial state probabilities
initial_probs1 <- c(0.9, 0.07, 0.02, 0.01)
initial_probs2 <- c(0.9, 0.04, 0.03, 0.01, 0.01, 0.01)

# Birth cohort
biofam3c$covariates$cohort <- cut(biofam3c$covariates$birthyr, c(1908, 1935, 1945, 1957))
biofam3c$covariates$cohort <- factor(
  biofam3c$covariates$cohort, labels=c("1909-1935", "1936-1945", "1946-1957"))

## MHMM with own starting values and covariates
init_mhmm_bf2 <- build_mhmm(
  observations = list(marr_seq, child_seq, left_seq),
  initial_probs = list(initial_probs1, initial_probs1, initial_probs2),
  transition_probs = list(A1, A1, A2),
  emission_probs = list(list(B1_marr, B1_child, B1_left),
    list(B2_marr, B2_child, B2_left),
    list(B3_marr, B3_child, B3_left)),
  formula = ~sex + cohort, data = biofam3c$covariates,
  cluster_names = c("Cluster 1", "Cluster 2", "Cluster 3"),
  channel_names = c("Marriage", "Parenthood", "Residence"),
  state_names = list(paste("State", 1:4), paste("State", 1:4),
                     paste("State", 1:6)))
```

`build_mm` *Build a Markov Model*

Description

Function `build_mm` builds and automatically estimates a Markov model. It is also a shortcut for constructing a Markov model as a restricted case of an `hmm` object.

Usage

```
build_mm(observations)
```

Arguments

observations	An stslist object (see seqdef) containing the sequences.

Details

Unlike the other build functions in seqHMM, the build_mm function automatically estimates the model parameters. As initial and transition probabilities can be directly estimated from the observed initial state probabilities and transition counts, there is no need for starting values or further estimation with the fit_model function.

Value

Object of class hmm with following elements:

observations	State sequence object or a list of such containing the data.
transition_probs	A matrix of transition probabilities.
emission_probs	A matrix or a list of matrices of emission probabilities.
initial_probs	A vector of initial probabilities.
state_names	Names for hidden states.
symbol_names	Names for observed states.
channel_names	Names for channels of sequence data.
length_of_sequences	(Maximum) length of sequences.
n_sequences	Number of sequences.
n_symbols	Number of observed states (in each channel).
n_states	Number of hidden states.
n_channels	Number of channels.

See Also

plot.hmm for plotting the model.

Examples

```
# Construct sequence data
data("mvad", package = "TraMineR")

mvad_alphabet <-
  c("employment", "FE", "HE", "joblessness", "school", "training")
mvad_labels <- c("employment", "further education", "higher education",
  "joblessness", "school", "training")
mvad_scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad_seq <- seqdef(mvad, 17:86, alphabet = mvad_alphabet,
  states = mvad_scodes, labels = mvad_labels, xtstep = 6)
```

```
# Define a color palette for the sequence data
attr(mvad_seq, "cpal") <- colorpalette[[6]]

# Estimate the Markov model
mm_mvad <- build_mm(observations = mvad_seq)
```

`build_mmm` *Build a Mixture Markov Model*

Description

Function `build_mmm` is a shortcut for constructing a mixture Markov model as a restricted case of an `mhmm` object.

Usage

```
build_mmm(observations, n_clusters, transition_probs, initial_probs, formula,
  data, coefficients, cluster_names = NULL, ...)
```

Arguments

`observations`	An `stslist` object (see `seqdef`) containing the sequences.
`n_clusters`	A scalar giving the number of clusters/submodels (not used if starting values for model parameters are given with `initial_probs` and `transition_probs`).
`transition_probs`	A list of matrices of transition probabilities for submodels of each cluster.
`initial_probs`	A list which contains vectors of initial state probabilities for submodels of each cluster.
`formula`	Covariates as an object of class `formula`, left side omitted.
`data`	An optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`.
`coefficients`	An optional kxl matrix of regression coefficients for time-constant covariates for mixture probabilities, where l is the number of clusters and k is the number of covariates. A logit-link is used for mixture probabilities. The first column is set to zero.
`cluster_names`	A vector of optional names for the clusters.
`...`	Additional arguments to `simulate_transition_probs`.

Value

Object of class `mhmm` with following elements:

`observations` State sequence object or a list of such containing the data.

`transition_probs` A matrix of transition probabilities.

`emission_probs` A matrix or a list of matrices of emission probabilities.

`initial_probs` A vector of initial probabilities.

`coefficients` A matrix of parameter coefficients for covariates (covariates in rows, clusters in columns).

`X` Covariate values for each subject.

`cluster_names` Names for clusters.

`state_names` Names for hidden states.

`symbol_names` Names for observed states.

`channel_names` Names for channels of sequence data

`length_of_sequences` (Maximum) length of sequences.

`n_sequences` Number of sequences.

`n_symbols` Number of observed states (in each channel).

`n_states` Number of hidden states.

`n_channels` Number of channels.

`n_covariates` Number of covariates.

`n_clusters` Number of clusters.

See Also

`fit_model` for estimating model parameters; `summary.mhmm` for a summary of a mixture model; `separate_mhmm` for organizing an `mhmm` object into a list of separate `hmm` objects; and `plot.mhmm` for plotting mixture models.

Examples

```
# Define sequence data
data("mvad", package = "TraMineR")
mvad_alphabet <- c("employment", "FE", "HE", "joblessness", "school",
  "training")
mvad_labels <- c("employment", "further education", "higher education",
  "joblessness", "school", "training")
mvad_scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad_seq <- seqdef(mvad, 17:86, alphabet = mvad_alphabet, states = mvad_scodes,
  labels = mvad_labels, xtstep = 6)

# Initialize the MMM
set.seed(123)
mmm_mvad <- build_mmm(observations = mvad_seq,
  n_clusters = 2,
  formula = ~male, data = mvad)

## Not run:
# Estimate model parameters
mmm_mvad <- fit_model(mmm_mvad)$model

# Plot model (both clusters in the same plot)
require(igraph)
plot(mmm_mvad, interactive = FALSE,
  # Modify legend position and properties
  with.legend = "right", legend.prop = 0.3, cex.legend = 1.2,
  # Define vertex layout
  layout = layout_as_star,
  # Modify edge properties
  edge.label = NA, edge.arrow.size = 0.8, edge.curved = 0.2,
  # Modify vertex label positions (initial probabilities)
  vertex.label.pos = c("left", "right", "right", "left", "left", "right"))

# Summary of the MMM
summary(mmm_mvad)

## End(Not run)
```

colorpalette *Color palettes*

Description

A list containing ready defined color palettes with distinct colors using iWantHue. By default, seqHMM uses these palettes when assigning colors.

Format

A list with 200 color palettes.

Source

iWantHue web page http://tools.medialab.sciences-po.fr/iwanthue/

See Also

plot_colors for visualization of color palettes. Implementations of iWantHue for R:

- https://github.com/hoesler/rwantshue
- https://github.com/johnbaums/hues

Examples

```
data("colorpalette")
# Color palette with 9 colors
colorpalette[[9]]
# Color palette with 24 colors
colorpalette[[24]]
```

---

estimate_coef *Estimate Regression Coefficients of Mixture Hidden Markov Models*

Description

Function `estimate_coef` estimates the regression coefficients of mixture hidden Markov models and its restricted variants while keeping other parameters fixed.

Usage

```
estimate_coef(model, threads = 1)
```

Arguments

`model`	An object of class `hmm` or `mhmm`.
`threads`	Number of threads to use in parallel computing. The default is 1.

---

fit_model *Estimate Parameters of (Mixture) Hidden Markov Models and Their Restricted Variants*

Description

Function `fit_model` estimates the parameters of mixture hidden Markov models and its restricted variants using maximimum likelihood. Initial values for estimation are taken from the corresponding components of the model with preservation of original zero probabilities.

Usage

```
fit_model(model, em_step = TRUE, global_step = FALSE, local_step = FALSE,
  control_em = list(), control_global = list(), control_local = list(),
  lb, ub, threads = 1, log_space = FALSE, ...)
```

Arguments

`model`	An object of class `hmm` or `mhmm`.
`em_step`	Logical. Whether or not to use the EM algorithm at the start of the parameter estimation. The default is `TRUE`.
`global_step`	Logical. Whether or not to use global optimization via `nloptr` (possibly after the EM step). The default is `FALSE`.
`local_step`	Logical. Whether or not to use local optimization via `nloptr` (possibly after the EM and/or global steps). The default is `FALSE`.
`control_em`	Optional list of control parameters for the EM algorithm. Possible arguments are

maxeval The maximum number of iterations, the default is 1000.

print_level The level of printing. Possible values are 0 (prints nothing), 1 (prints information at the start and the end of the algorithm), 2 (prints at every iteration), and for mixture models 3 (print also during optimization of coefficients).

reltol Relative tolerance for convergence defined as $(logLik_new - logLik_old)/(abs(logLik_old) + 0.1)$. The default is 1e-10.

restart A list containing options for possible EM restarts with the following components:

- **times** Number of restarts of the EM algorithm using random initial values. The default is 0, i.e. no restarts.
- **transition** Logical. Should the original transition probabilities be varied? The default is `TRUE`.
- **emission** Logical. Should the original emission probabilities be varied? The default is `TRUE`.
- **sd** Standard deviation for `rnorm` used in randomization. The default is 0.25.
- **maxeval** Maximum number of iterations, the default is `control_em$maxeval`
- **print_level** Level of printing in restarted EM steps. The default is `control_em$print_level`.
- **reltol** Relative tolerance for convergence at restarted EM steps. The default is `control_em$reltol`. If the relative change of the final model of the restart phase is larger than the tolerance for the original EM phase, the final model is re-estimated with the original `reltol` and `maxeval` at the end of the EM step.
- **n_optimum** Save the log-likelihood values of the `n_optimum` best models (from all estimated models including the the first EM run.). The default is `min(times + 1, 25)`.
- **use_original** If `TRUE`. Use the initial values of the input model as starting points for the permutations. Otherwise permute the results of the first EM run.

`control_global`	Optional list of additional arguments for `nloptr` argument `opts`. The default values are

algorithm `"NLOPT_GD_MLSL_LDS"`

local_opts `list(algorithm = "NLOPT_LD_LBFGS", ftol_rel = 1e-6, xtol_rel = 1e-4)`

	maxeval `10000` (maximum number of iterations in global optimization algorithm.)
	maxtime `60` (maximum time for global optimization. Set to 0 for unlimited time.)
`control_local`	Optional list of additional arguments for `nloptr` argument `opts`. The default values are **algorithm** `"NLOPT_LD_LBFGS"` **ftol_rel** `1e-10` **xtol_rel** `1e-8` **maxeval** `10000` (maximum number of iterations)
`lb, ub`	Lower and upper bounds for parameters in Softmax parameterization. The default interval is $[pmin(-25, 2 * initialvalues), pmax(25, 2 * initialvalues)]$, except for gamma coefficients, where the scale of covariates is taken into account. Note that it might still be a good idea to scale covariates around unit scale. Bounds are used only in the global optimization step.
`threads`	Number of threads to use in parallel computing. The default is 1.
`log_space`	Make computations using log-space instead of scaling for greater numerical stability at a cost of decreased computational performance. The default is `FALSE`.
`...`	Additional arguments to `nloptr`.

Details

The fitting function provides three estimation steps: 1) EM algorithm, 2) global optimization, and 3) local optimization. The user can call for one method or any combination of these steps, but should note that they are preformed in the above-mentioned order. The results from a former step are used as starting values in a latter, except for some of global optimization algorithms (such as MLSL and StoGO) which only use initial values for setting up the boundaries for the optimization.

It is possible to rerun the EM algorithm automatically using random starting values based on the first run of EM. Number of restarts is defined by the `restart` argument in `control_em`. As the EM algorithm is relatively fast, this method might be preferred option compared to the proper global optimization strategy of step 2.

The default global optimization method (triggered via `global_step = TRUE`) is the multilevel single-linkage method (MLSL) with the LDS modification (`NLOPT_GD_MLSL_LDS` as `algorithm` in `control_global`), with L-BFGS as the local optimizer. The MLSL method draws random starting points and performs a local optimization from each. The LDS modification uses low-discrepancy sequences instead of pseudo-random numbers as starting points and should improve the convergence rate. In order to reduce the computation time spent on non-global optima, the convergence tolerance of the local optimizer is set relatively large. At step 3, a local optimization (L-BFGS by default) is run with a lower tolerance to find the optimum with high precision.

There are some theoretical guarantees that the MLSL method used as the default optimizer in step 2 shoud find all local optima in a finite number of local optimizations. Of course, it might not always succeed in a reasonable time. The EM algorithm can help in finding good boundaries for the search, especially with good starting values, but in some cases it can mislead. A good strategy is to try a couple of different fitting options with different combinations of the methods: e.g. all steps, only global and local steps, and a few evaluations of EM followed by global and local optimization.

By default, the estimation time is limited to 60 seconds in global optimization step, so it is advisable to change the default settings for the proper global optimization.

Any algorithm available in the `nloptr` function can be used for the global and local steps.

Value

logLik Log-likelihood of the estimated model.

em_results Results after the EM step: log-likelihood (`logLik`), number of iterations (`iterations`), relative change in log-likelihoods between the last two iterations (`change`), and the log-likelihoods of the `n_optimum` best models after the EM step (`best_opt_restart`).

global_results Results after the global step.

local_results Results after the local step.

call The matched function call.

See Also

`build_hmm`, `build_mhmm`, `build_mm`, `build_mmm`, and `build_lcm` for constructing different types of models; `summary.mhmm` for a summary of a MHMM; `separate_mhmm` for reorganizing a MHMM into a list of separate hidden Markov models; `plot.hmm` and `plot.mhmm` for plotting model objects; and `ssplot` and `mssplot` for plotting stacked sequence plots of `hmm` and `mhmm` objects.

Examples

```
# Hidden Markov model

data("mvad", package = "TraMineR")

mvad_alphabet <-
  c("employment", "FE", "HE", "joblessness", "school", "training")
mvad_labels <- c("employment", "further education", "higher education",
  "joblessness", "school", "training")
mvad_scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad_seq <- seqdef(mvad, 17:86, alphabet = mvad_alphabet,
  states = mvad_scodes, labels = mvad_labels, xtstep = 6)

attr(mvad_seq, "cpal") <- colorpalette[[6]]

# Starting values for the emission matrix
emiss <- matrix(
  c(0.05, 0.05, 0.05, 0.05, 0.75, 0.05, # SC
    0.05, 0.75, 0.05, 0.05, 0.05, 0.05, # FE
    0.05, 0.05, 0.05, 0.4,  0.05, 0.4,  # JL, TR
    0.05, 0.05, 0.75, 0.05, 0.05, 0.05, # HE
    0.75, 0.05, 0.05, 0.05, 0.05, 0.05),# EM
  nrow = 5, ncol = 6, byrow = TRUE)

# Starting values for the transition matrix
trans <- matrix(0.025, 5, 5)
diag(trans) <- 0.9
```

```
# Starting values for initial state probabilities
initial_probs <- c(0.2, 0.2, 0.2, 0.2, 0.2)

# Building a hidden Markov model
init_hmm_mvad <- build_hmm(observations = mvad_seq,
  transition_probs = trans, emission_probs = emiss,
  initial_probs = initial_probs)

## Not run:
set.seed(21)
fit_hmm_mvad <- fit_model(init_hmm_mvad, control_em = list(restart = list(times = 50)))
hmm_mvad <- fit_hmm_mvad$model

## End(Not run)

# save time, load the previously estimated model
data("hmm_mvad")

# Markov model
# Note: build_mm estimates model parameters from observations,
# no need for estimating with fit_model

mm_mvad <- build_mm(observations = mvad_seq)

# Comparing likelihoods, MM fits better
logLik(hmm_mvad)
logLik(mm_mvad)

## Not run:
require("igraph") #for layout_in_circle

plot(mm_mvad, layout = layout_in_circle, legend.prop = 0.3,
  edge.curved = 0.3, edge.label = NA,
  vertex.label.pos = c(0, 0, pi, pi, pi, 0))

##############################################################


#' # Three-state three-channel hidden Markov model
# See ?hmm_biofam for five-state version

data("biofam3c")

# Building sequence objects
marr_seq <- seqdef(biofam3c$married, start = 15,
  alphabet = c("single", "married", "divorced"))
child_seq <- seqdef(biofam3c$children, start = 15,
  alphabet = c("childless", "children"))
left_seq <- seqdef(biofam3c$left, start = 15,
  alphabet = c("with parents", "left home"))

# Define colors
```

```
attr(marr_seq, "cpal") <- c("violetred2", "darkgoldenrod2", "darkmagenta")
attr(child_seq, "cpal") <- c("darkseagreen1", "coral3")
attr(left_seq, "cpal") <- c("lightblue", "red3")

# Starting values for emission matrices

emiss_marr <- matrix(NA, nrow = 3, ncol = 3)
emiss_marr[1,] <- seqstatf(marr_seq[, 1:5])[, 2] + 1
emiss_marr[2,] <- seqstatf(marr_seq[, 6:10])[, 2] + 1
emiss_marr[3,] <- seqstatf(marr_seq[, 11:16])[, 2] + 1
emiss_marr <- emiss_marr / rowSums(emiss_marr)

emiss_child <- matrix(NA, nrow = 3, ncol = 2)
emiss_child[1,] <- seqstatf(child_seq[, 1:5])[, 2] + 1
emiss_child[2,] <- seqstatf(child_seq[, 6:10])[, 2] + 1
emiss_child[3,] <- seqstatf(child_seq[, 11:16])[, 2] + 1
emiss_child <- emiss_child / rowSums(emiss_child)

emiss_left <- matrix(NA, nrow = 3, ncol = 2)
emiss_left[1,] <- seqstatf(left_seq[, 1:5])[, 2] + 1
emiss_left[2,] <- seqstatf(left_seq[, 6:10])[, 2] + 1
emiss_left[3,] <- seqstatf(left_seq[, 11:16])[, 2] + 1
emiss_left <- emiss_left / rowSums(emiss_left)

# Starting values for transition matrix
trans <- matrix(c(0.9, 0.07, 0.03,
                0,  0.9,  0.1,
                0,    0,    1), nrow = 3, ncol = 3, byrow = TRUE)

# Starting values for initial state probabilities
inits <- c(0.9, 0.09, 0.01)

# Building hidden Markov model with initial parameter values
init_hmm_bf <- build_hmm(
  observations = list(marr_seq, child_seq, left_seq),
  transition_probs = trans,
  emission_probs = list(emiss_marr, emiss_child, emiss_left),
  initial_probs = inits)

# Fitting the model with different optimization schemes

# Only EM with default values
hmm_1 <- fit_model(init_hmm_bf)
hmm_1$logLik # -24179.1

# Only L-BFGS
hmm_2 <- fit_model(init_hmm_bf, em_step = FALSE, local_step = TRUE)
hmm_2$logLik # -22267.75

# Global optimization via MLSL_LDS with L-BFGS as local optimizer and final polisher
# This can be slow, use parallel computing by adjusting threads argument
# (here threads = 1 for portability issues)
hmm_3 <- fit_model(
  init_hmm_bf, em_step = FALSE, global_step = TRUE, local_step = TRUE,
  control_global = list(maxeval = 5000, maxtime = 0), threads = 1)
hmm_3$logLik # -21675.42

# EM with restarts, much faster than MLSL
set.seed(123)
hmm_4 <- fit_model(init_hmm_bf, control_em = list(restart = list(times = 5)))
hmm_4$logLik # -21675.4

# Global optimization via StoGO with L-BFGS as final polisher
# This can be slow, use parallel computing by adjusting threads argument
# (here threads = 1 for portability issues)
set.seed(123)
hmm_5 <- fit_model(
   init_hmm_bf, em_step = FALSE, global_step = TRUE, local_step = TRUE,
   lb = -50, ub = 50, control_global = list(algorithm = "NLOPT_GD_STOGO",
   maxeval = 2500, maxtime = 0), threads = 1)
hmm_5$logLik # -21675.4

##############################################################

# Mixture HMM

data("biofam3c")

## Building sequence objects
marr_seq <- seqdef(biofam3c$married, start = 15,
  alphabet = c("single", "married", "divorced"))
child_seq <- seqdef(biofam3c$children, start = 15,
  alphabet = c("childless", "children"))
left_seq <- seqdef(biofam3c$left, start = 15,
  alphabet = c("with parents", "left home"))

## Choosing colors
attr(marr_seq, "cpal") <- c("#AB82FF", "#E6AB02", "#E7298A")
attr(child_seq, "cpal") <- c("#66C2A5", "#FC8D62")
attr(left_seq, "cpal") <- c("#A6CEE3", "#E31A1C")

## Starting values for emission probabilities
# Cluster 1
B1_marr <- matrix(
  c(0.8, 0.1, 0.1, # High probability for single
    0.8, 0.1, 0.1,
    0.3, 0.6, 0.1, # High probability for married
    0.3, 0.3, 0.4), # High probability for divorced
  nrow = 4, ncol = 3, byrow = TRUE)

B1_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.9, 0.1,
    0.9, 0.1),
  nrow = 4, ncol = 2, byrow = TRUE)
```

```
B1_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9, # High probability for having left home
    0.1, 0.9,
    0.1, 0.9),
  nrow = 4, ncol = 2, byrow = TRUE)

# Cluster 2

B2_marr <- matrix(
  c(0.8, 0.1, 0.1, # High probability for single
    0.8, 0.1, 0.1,
    0.1, 0.8, 0.1, # High probability for married
    0.7, 0.2, 0.1),
  nrow = 4, ncol = 3, byrow = TRUE)

B2_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.9, 0.1,
    0.1, 0.9),
  nrow = 4, ncol = 2, byrow = TRUE)

B2_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9,
    0.1, 0.9,
    0.1, 0.9),
  nrow = 4, ncol = 2, byrow = TRUE)

# Cluster 3
B3_marr <- matrix(
  c(0.8, 0.1, 0.1, # High probability for single
    0.8, 0.1, 0.1,
    0.8, 0.1, 0.1,
    0.1, 0.8, 0.1, # High probability for married
    0.3, 0.4, 0.3,
    0.1, 0.1, 0.8), # High probability for divorced
  nrow = 6, ncol = 3, byrow = TRUE)

B3_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.5, 0.5,
    0.5, 0.5,
    0.5, 0.5,
    0.1, 0.9),
  nrow = 6, ncol = 2, byrow = TRUE)


B3_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9,
    0.5, 0.5,
    0.5, 0.5,
    0.1, 0.9,
    0.1, 0.9),
  nrow = 6, ncol = 2, byrow = TRUE)

# Starting values for transition matrices
A1 <- matrix(
  c(0.80, 0.16, 0.03, 0.01,
    0,    0.90, 0.07, 0.03,
    0,    0,    0.90, 0.10,
    0,    0,    0,       1),
  nrow = 4, ncol = 4, byrow = TRUE)

A2 <- matrix(
  c(0.80, 0.10, 0.05, 0.03, 0.01, 0.01,
    0,    0.70, 0.10, 0.10, 0.05, 0.05,
    0,    0,    0.85, 0.01, 0.10, 0.04,
    0,    0,    0,    0.90, 0.05, 0.05,
    0,    0,    0,    0,    0.90, 0.10,
    0,    0,    0,    0,    0,       1),
  nrow = 6, ncol = 6, byrow = TRUE)

# Starting values for initial state probabilities
initial_probs1 <- c(0.9, 0.07, 0.02, 0.01)
initial_probs2 <- c(0.9, 0.04, 0.03, 0.01, 0.01, 0.01)

# Birth cohort
biofam3c$covariates$cohort <- cut(biofam3c$covariates$birthyr, c(1908, 1935, 1945, 1957))
biofam3c$covariates$cohort <- factor(
  biofam3c$covariates$cohort, labels=c("1909-1935", "1936-1945", "1946-1957"))

# Build mixture HMM
init_mhmm_bf <- build_mhmm(
  observations = list(marr_seq, child_seq, left_seq),
  initial_probs = list(initial_probs1, initial_probs1, initial_probs2),
  transition_probs = list(A1, A1, A2),
  emission_probs = list(list(B1_marr, B1_child, B1_left),
    list(B2_marr, B2_child, B2_left),
    list(B3_marr, B3_child, B3_left)),
  formula = ~sex + cohort, data = biofam3c$covariates,
  channel_names = c("Marriage", "Parenthood", "Residence"))


# Fitting the model with different settings

# Only EM with default values
mhmm_1 <- fit_model(init_mhmm_bf)
mhmm_1$logLik # -12713.08

# Only L-BFGS
mhmm_2 <- fit_model(init_mhmm_bf, em_step = FALSE, local_step = TRUE)
```

```
mhmm_2$logLik # -12966.51

# Use EM with multiple restarts
set.seed(123)
mhmm_3 <- fit_model(init_mhmm_bf, control_em = list(restart = list(times = 5, transition = FALSE)))
mhmm_3$logLik # -12713.08

## End(Not run)
```

`forward_backward` *Forward and Backward Probabilities for Hidden Markov Model*

Description

The `forward_backward` function computes scaled forward and backward probabilities of a hidden Markov model.

Usage

```
forward_backward(model, forward_only = FALSE, log_space = FALSE,
  threads = 1)
```

Arguments

`model`	Object of class `hmm` or `mhmm`.
`forward_only`	If `TRUE`, only forward probabilities are computed. The default is `FALSE`.
`log_space`	Compute forward and backward probabilities in logarithmic scale instead of scaling. The default is `FALSE`.
`threads`	Number of threads used in parallel computing. The default is 1.

Value

List with components

`forward_probs`	If `log_space = FALSE`, scaled forward probabilities, i.e. probability of state given observations up to that time point. If `log_space = TRUE`, logarithms of non-scaled forward probabilities.
`backward_probs`	Scaled backward probabilities (`log_space = FALSE`), or logarithms of non-scaled backward probabilities(`log_space = TRUE`).
`scaling_factors`	Sum of non-scaled forward probabilities at each time point. Only computed if `log_space = FALSE`.

In case of multiple observations, these are computed independently for each sequence.

Examples

```
# Load a pre-defined MHMM
data("mhmm_biofam")

# Compute forward and backward probabilities
fb <- forward_backward(mhmm_biofam)

# The most probable hidden state at time t
# given the observations up to time t for the first subject:
apply(fb$forward_probs[, , 1], 2, which.max)
```

---

gridplot *Plot Multidimensional Sequence Plots in a Grid*

Description

Function `gridplot` plots multiple `ssp` objects to a grid.

Usage

```
gridplot(x, nrow = NA, ncol = NA, byrow = FALSE, with.legend = "auto",
  legend.pos = "auto", legend.pos2 = "center", title.legend = "auto",
  ncol.legend = "auto", with.missing.legend = "auto", row.prop = "auto",
  col.prop = "auto", cex.legend = 1)
```

Arguments

`x`	A list of `ssp` objects.
`nrow, ncol`	Optional arguments to arrange plots.
`byrow`	Controls the order of plotting. Defaults to `FALSE`, i.e. plots are arranged column-wise.
`with.legend`	Defines if and how the legends for the states are plotted. The default value `"auto"` (equivalent to `TRUE` and `"many"`) creates separate legends for each requested plot. Other possibilities are `"combined"` (all legends combined) and `FALSE` (no legend).
`legend.pos`	Defines the positions of the legend boxes relative to the whole plot. Either one of `"bottom"` (equivalent to `"auto"`) or `"right"`, or a numerical vector of grid cells (by order) to print the legends to (the cells must be in one row/column).
`legend.pos2`	Defines the positions of the legend boxes relative to the cell(s). One of `"bottomright"`, `"bottom"`, `"bottomleft"`, `"left"`, `"topleft"`, `"top"` (the default), `"topright"`, `"right"` and `"center"`.
`title.legend`	The titles for the legend boxes. The default `"auto"` takes the titles from the channel labels provided by the first object in `x`. `NA` prints no title.
`ncol.legend`	(A vector of) the number of columns for the legend(s). The default `"auto"` creates one column for each legend.

`with.missing.legend`
: If set to `"auto"` (the default), a legend for the missing state is added automatically if one or more of the sequences in data contain missing states. With the value `TRUE` a legend for the missing state is added in any case; equivalently `FALSE` omits the legend for the missing state.

`row.prop`
: Sets the proportions of the row heights of the grid. The default value is `"auto"` for even row heights. Takes a vector of values from 0 to 1, with values summing to 1.

`col.prop`
: Sets the proportion of the column heights of the grid. The default value is `"auto"` for even column widths. Takes a vector of values from 0 to 1, with values summing to 1.

`cex.legend`
: Expansion factor for setting the size of the font for the labels in the legend. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.

See Also

`ssp` for defining the plot before using `gridplot`, and `plot.ssp` for plotting only one ssp object.

Examples

```
## Not run:
data("biofam3c")

# Creating sequence objects
child_seq <- seqdef(biofam3c$children, start = 15)
marr_seq <- seqdef(biofam3c$married, start = 15)
left_seq <- seqdef(biofam3c$left, start = 15)

## Choosing colors
attr(child_seq, "cpal") <- c("#66C2A5", "#FC8D62")
attr(marr_seq, "cpal") <- c("#AB82FF", "#E6AB02", "#E7298A")
attr(left_seq, "cpal") <- c("#A6CEE3", "#E31A1C")


# Preparing plot for state distribution plots of observations for women
ssp_f <- ssp(
  list(child_seq[biofam3c$covariates$sex == "woman",],
       marr_seq[biofam3c$covariates$sex == "woman",],
       left_seq[biofam3c$covariates$sex == "woman",]),
  type = "d", plots = "obs", title = "Women",
  ylab = c("Children", "Married", "Left home"))

# Preparing plot for state distribution plots of observations for men
# (Updating the previous plot, only arguments that change values)
ssp_m <- update(ssp_f, title = "Men",
  x = list(child_seq[biofam3c$covariates$sex == "man",],
       marr_seq[biofam3c$covariates$sex == "man",],
       left_seq[biofam3c$covariates$sex == "man",]))

# Plotting state distribution plots of observations for women and men in two columns
```

```
gridplot(list(ssp_f, ssp_m), ncol = 2, with.legend = FALSE)

# Preparing plots for women's state distributions
ssp_f2 <- ssp(
  list(marr_seq[biofam3c$covariates$sex == "woman",],
       child_seq[biofam3c$covariates$sex == "woman",],
       left_seq[biofam3c$covariates$sex == "woman",]),
  type = "d", border = NA, with.legend = FALSE,
  title = "State distributions for women", title.n = FALSE, xtlab = 15:30,
  ylab.pos = c(1, 2, 1), ylab = c("Married", "Children", "Left home"))

# The same plot with sequences instead of state distributions
ssp_f3 <- update(
  ssp_f2, type = "I", sortv = "mds.obs", title = "Sequences for women")

# State distributions with men's data
ssp_m2 <- update(
  ssp_f2, title = "State distributions for men",
  x = list(marr_seq[biofam3c$covariates$sex == "man",],
           child_seq[biofam3c$covariates$sex == "man",],
           left_seq[biofam3c$covariates$sex == "man",]))

# Men's sequences
ssp_m3 <- update(
  ssp_m2, type = "I", sortv = "mds.obs", title = "Sequences for men")

# Plotting state distributions and index plots of observations
# for women and men in two columns (+ one column for legends)
gridplot(
  list(ssp_f2, ssp_f3, ssp_m2, ssp_m3), ncol = 3, byrow = TRUE,
  with.legend = "combined", legend.pos = "right", col.prop = c(0.35, 0.35, 0.3))

# The same with different positioning and fixed cells for legends
gridplot(
  list(ssp_f2, ssp_f3, ssp_m2, ssp_m3), ncol = 2, nrow = 3, byrow = TRUE,
  # defining the legend positions by the cell numbers
  legend.pos = 3:4)

## End(Not run)
```

hidden_paths *Most Probable Paths of Hidden States*

Description

Function `hidden_paths` computes the most probable path of hidden states of a (mixture) hidden Markov model given the observed sequences.

Usage

```
hidden_paths(model)
```

Arguments

model	A hidden Markov model of class hmm or a mixture HMM of class mhmm.

Value

The most probable paths of hidden states as an stslist object (see seqdef). The log-probability is included as an attribute log_prob.

See Also

hmm_biofam for information on the model used in the example; and seqIplot, ssplot, or mssplot for plotting hidden paths.

Examples

```
# Load a pre-defined HMM
data("hmm_biofam")

# Compute the most probable hidden state paths given the data and the model
mpp <- hidden_paths(hmm_biofam)

# Plot hidden paths for the first 100 individuals
ssplot(mpp, type = "I", tlim = 1:100)

# Because the model structure is so sparse that the posterior probabilities are
# mostly peaked to single state at each time point, the joint probability of
# observations and most probable paths of hidden states is almost identical to
# log-likelihood:

sum(attr(mpp, "log_prob"))
logLik(hmm_biofam)
```

---

hmm_biofam *Hidden Markov model for the biofam data*

Description

A five-state hidden Markov model (HMM) fitted for the biofam data.

Format

A hidden Markov model of class hmm; a left-to-right model with four hidden states.

Details

The model is loaded by calling `data(hmm_biofam)`. It was created with the following code:

```
data("biofam3c")

# Building sequence objects
marr_seq <- seqdef(biofam3c$married, start = 15,
  alphabet = c("single", "married", "divorced"))
child_seq <- seqdef(biofam3c$children, start = 15,
  alphabet = c("childless", "children"))
left_seq <- seqdef(biofam3c$left, start = 15,
  alphabet = c("with parents", "left home"))

## Choosing colors
attr(marr_seq, "cpal") <- c("violetred2", "darkgoldenrod2", "darkmagenta")
attr(child_seq, "cpal") <- c("darkseagreen1", "coral3")
attr(left_seq, "cpal") <- c("lightblue", "red3")

init <- c(0.9, 0.05, 0.02, 0.02, 0.01)

# Starting values for transition matrix
trans <- matrix(
  c(0.8, 0.10, 0.05, 0.03, 0.02,
    0,    0.9, 0.05, 0.03, 0.02,
    0,      0,  0.9, 0.07, 0.03,
    0,      0,    0,  0.9,  0.1,
    0,      0,    0,    0,    1),
  nrow = 5, ncol = 5, byrow = TRUE)

# Starting values for emission matrices
emiss_marr <- matrix(
  c(0.9, 0.05, 0.05, # High probability for single
    0.9, 0.05, 0.05,
    0.05, 0.9, 0.05, # High probability for married
    0.05, 0.9, 0.05,
    0.3, 0.3, 0.4), # mixed group
  nrow = 5, ncol = 3, byrow = TRUE)

emiss_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.1, 0.9,
    0.1, 0.9,
    0.5, 0.5),
  nrow = 5, ncol = 2, byrow = TRUE)

emiss_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9,
    0.1, 0.9,
    0.1, 0.9,
    0.5, 0.5),
  nrow = 5, ncol = 2, byrow = TRUE)

initmod <- build_hmm(
  observations = list(marr_seq, child_seq, left_seq),
  initial_probs = init, transition_probs = trans,
  emission_probs = list(emiss_marr, emiss_child,
    emiss_left),
  channel_names = c("Marriage", "Parenthood", "Residence"))

fit_biofam <- fit_model(initmod, em = FALSE, local = TRUE)
hmm_biofam <- fit_biofam$model
```

See Also

Examples of building and fitting HMMs in `build_hmm` and `fit_model`; and `biofam` for the original data and `biofam3c` for the three-channel version used in this model.

Examples

```
# Plotting the model
plot(hmm_biofam)
```

`hmm_mvad` *Hidden Markov model for the mvad data*

Description

A hidden Markov model (MMM) fitted for the `mvad` data.

Format

A hidden Markov model of class `hmm`; unrestricted model with six hidden states.

Details

Model was created with the following code:

```
data("mvad", package = "TraMineR")

mvad_alphabet <-
  c("employment", "FE", "HE", "joblessness", "school", "training")
```

```
mvad_labels <- c("employment", "further education", "higher education",
  "joblessness", "school", "training")
mvad_scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad_seq <- seqdef(mvad, 17:86, alphabet = mvad_alphabet,
  states = mvad_scodes, labels = mvad_labels, xtstep = 6)

attr(mvad_seq, "cpal") <- colorpalette[[6]]

# Starting values for the emission matrix
emiss <- matrix(
  c(0.05, 0.05, 0.05, 0.05, 0.75, 0.05, # SC
    0.05, 0.75, 0.05, 0.05, 0.05, 0.05, # FE
    0.05, 0.05, 0.05, 0.4,  0.05, 0.4,  # JL, TR
    0.05, 0.05, 0.75, 0.05, 0.05, 0.05, # HE
    0.75, 0.05, 0.05, 0.05, 0.05, 0.05),# EM
  nrow = 5, ncol = 6, byrow = TRUE)

# Starting values for the transition matrix
trans <- matrix(0.025, 5, 5)
diag(trans) <- 0.9

# Starting values for initial state probabilities
initial_probs <- c(0.2, 0.2, 0.2, 0.2, 0.2)

# Building a hidden Markov model
init_hmm_mvad <- build_hmm(observations = mvad_seq,
  transition_probs = trans, emission_probs = emiss,
  initial_probs = initial_probs)

set.seed(21)
fit_hmm_mvad <- fit_model(init_hmm_mvad, control_em = list(restart = list(times = 100)))
hmm_mvad <- fit_hmm_mvad$model
```

See Also

Examples of building and fitting HMMs in build_hmm and fit_model; and mvad for more information on the data.

Examples

```
data("hmm_mvad")

# Plotting the model
plot(hmm_mvad)
```

`logLik.hmm` *Log-likelihood of the Hidden Markov Model*

Description

Function `logLik.hmm` computes the log-likelihood value of a hidden Markov model.

Usage

```
## S3 method for class 'hmm'
logLik(object, partials = FALSE, threads = 1,
  log_space = FALSE, ...)
```

Arguments

`object`	A hidden Markov model of class `hmm`.
`partials`	Return a vector containing the individual contributions of each sequence to the total log-likelihood. The default is `FALSE`, which returns the sum of all log-likelihood components.
`threads`	Number of threads to use in parallel computing. The default is 1.
`log_space`	Make computations using log-space instead of scaling for greater numerical stability at the cost of decreased computational performance. The default is `TRUE`.
`...`	Ignored.

Value

Log-likelihood of the hidden Markov model. This is an object of class `logLik` with attributes `nobs` and `df` inherited from the model object.

See Also

`build_hmm` and `fit_model` for building and fitting Hidden Markov models.

`logLik.mhmm` *Log-likelihood of the Mixture Hidden Markov Model*

Description

Function `logLik.mhmm` computes the log-likelihood value of a mixture hidden Markov model.

Usage

```
## S3 method for class 'mhmm'
logLik(object, partials = FALSE, threads = 1,
  log_space = FALSE, ...)
```

Arguments

`object`	A mixture hidden Markov model of class `mhmm`.
`partials`	Return a vector containing the individual contributions of each sequence to the total log-likelihood. The default is `FALSE`, which returns the sum of all log-likelihood components.
`threads`	Number of threads to use in parallel computing. The default is 1.
`log_space`	Make computations using log-space instead of scaling for greater numerical stability at the cost of decreased computational performance. The default is `TRUE`.
`...`	Ignored.

Value

Log-likelihood of the mixture hidden Markov model. This is an object of class `logLik` with attributes `nobs` and `df` inherited from the model object.

See Also

`build_mhmm` and `fit_model` for building and fitting mixture Hidden Markov models.

`mc_to_sc` *Transform a Multichannel Hidden Markov Model into a Single Channel Representation*

Description

Transforms data and parameters of a multichannel model into a single channel model. Observed states (symbols) are combined and parameters multiplied across channels.

Usage

```
mc_to_sc(model, combine_missing = TRUE, all_combinations = FALSE)
```

Arguments

`model`	An object of class `hmm` or `mhmm`.
`combine_missing`	Controls whether combined states of observations at time t are coded missing (coded with $*$ in `stslists`) if one or more of the channels include missing information at time t. Defaults to `TRUE`. `FALSE` keeps missing states as they are, producing more states in data; e.g. $single/childless/*$ where the observation in channel 3 is missing.
`all_combinations`	Controls whether all possible combinations of observed states are included in the single channel representation or only combinations that are found in the data. Defaults to `FALSE`, i.e. only actual observations are included.

Details

Note that in case of no missing observations, the log-likelihood of the original and transformed models are identical but the AIC and BIC can be different as the model attribute `df` is recomputed based on the single channel representation.

See Also

`build_hmm` and `fit_model` for building and fitting Hidden Markov models; and `hmm_biofam` for information on the model used in the example.

Examples

```
# Loading a hidden Markov model of the biofam data (hmm object)
data("hmm_biofam")

# Convert the multichannel model to a single-channel model
sc <- mc_to_sc(hmm_biofam)

# Likelihoods of the single-channel and the multichannel model are the same
# (Might not be true if there are missing observations)
logLik(sc)
logLik(hmm_biofam)
```

`mc_to_sc_data` *Merge Multiple Sequence Objects into One (from Multichannel to Single Channel Data)*

Description

Function `mc_to_sc_data` combines observed states of multiple sequence objects into one, time point by time point.

Usage

```
mc_to_sc_data(data, combine_missing = TRUE, all_combinations = FALSE)
```

Arguments

`data`	A list of state sequence objects (`stslists`) created with the `seqdef` function.
`combine_missing`	Controls whether combined states of observations at time t are coded missing (coded with * in `stslists`) if one or more of the channels include missing information at time t. Defaults to `TRUE`. `FALSE` keeps missing states as they are, producing more states in data; e.g. single/childless/* where the observation in channel 3 is missing.
`all_combinations`	Controls whether all possible combinations of observed states are included in the single channel representation or only combinations that are found in the data. Defaults to `FALSE`, i.e. only actual observations are included.

See Also

mc_to_sc for transforming multichannel hmm or mhmm objects into single-channel representations; ssplot for plotting multiple sequence data sets in the same plot; and seqdef for creating state sequence objects.

Examples

```
# Load three-channel sequence data
data("biofam3c")

# Building sequence objects
marr_seq <- seqdef(biofam3c$married, start = 15,
  alphabet = c("single", "married", "divorced"))
child_seq <- seqdef(biofam3c$children, start = 15,
  alphabet = c("childless", "children"))
left_seq <- seqdef(biofam3c$left, start = 15,
  alphabet = c("with parents", "left home"))

# Define colors
attr(marr_seq, "cpal") <- c("violetred2", "darkgoldenrod2", "darkmagenta")
attr(child_seq, "cpal") <- c("darkseagreen1", "coral3")
attr(left_seq, "cpal") <- c("lightblue", "red3")

# Converting multichannel data to single-channel data
sc_data <- mc_to_sc_data(list(marr_seq, child_seq, left_seq))

# 10 combined states
alphabet(sc_data)

# Colors for combined states
attr(sc_data, "cpal") <- colorpalette[[14]][1:10]

# Plotting sequences for the first 10 subjects
ssplot(list("Marriage" = marr_seq, "Parenthood" = child_seq,
  "Residence" = left_seq, "Combined" = sc_data), type = "I",
  tlim = 1:10)


# Including all combinations (whether or not available in data)
sc_data_all <- mc_to_sc_data(list(marr_seq, child_seq, left_seq),
  all_combinations = TRUE)

# 12 combined states, 2 with no observations in data
seqstatf(sc_data_all)
```

---

mhmm_biofam *Mixture hidden Markov model for the biofam data*

Description

A mixture hidden Markov model (MHMM) fitted for the `biofam` data.

Format

A mixture hidden Markov model of class `mhmm`: three clusters with left-to-right models including 4, 4, and 6 hidden states. Two covariates, `sex` and `cohort`, explaining the cluster membership.

Details

The model was created with the following code:

```
data("biofam3c")

## Building sequence objects
marr_seq <- seqdef(biofam3c$married, start = 15,
  alphabet = c("single", "married", "divorced"))
child_seq <- seqdef(biofam3c$children, start = 15,
  alphabet = c("childless", "children"))
left_seq <- seqdef(biofam3c$left, start = 15,
  alphabet = c("with parents", "left home"))

## Choosing colors
attr(marr_seq, "cpal") <- c("#AB82FF", "#E6AB02", "#E7298A")
attr(child_seq, "cpal") <- c("#66C2A5", "#FC8D62")
attr(left_seq, "cpal") <- c("#A6CEE3", "#E31A1C")

## Starting values for emission probabilities
# Cluster 1
B1_marr <- matrix(
  c(0.8, 0.1, 0.1, # High probability for single
    0.8, 0.1, 0.1,
    0.3, 0.6, 0.1, # High probability for married
    0.3, 0.3, 0.4), # High probability for divorced
  nrow = 4, ncol = 3, byrow = TRUE)

B1_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.9, 0.1,
    0.9, 0.1),
  nrow = 4, ncol = 2, byrow = TRUE)

B1_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9, # High probability for having left home
    0.1, 0.9,
    0.1, 0.9),
  nrow = 4, ncol = 2, byrow = TRUE)

# Cluster 2

B2_marr <- matrix(
  c(0.8, 0.1, 0.1, # High probability for single
    0.8, 0.1, 0.1,
    0.1, 0.8, 0.1, # High probability for married
    0.7, 0.2, 0.1),
  nrow = 4, ncol = 3, byrow = TRUE)

B2_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.9, 0.1,
    0.1, 0.9),
  nrow = 4, ncol = 2, byrow = TRUE)

B2_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9,
    0.1, 0.9,
    0.1, 0.9),
  nrow = 4, ncol = 2, byrow = TRUE)

# Cluster 3
B3_marr <- matrix(
  c(0.8, 0.1, 0.1, # High probability for single
    0.8, 0.1, 0.1,
    0.8, 0.1, 0.1,
    0.1, 0.8, 0.1, # High probability for married
    0.3, 0.4, 0.3,
    0.1, 0.1, 0.8), # High probability for divorced
  nrow = 6, ncol = 3, byrow = TRUE)

B3_child <- matrix(
  c(0.9, 0.1, # High probability for childless
    0.9, 0.1,
    0.5, 0.5,
    0.5, 0.5,
    0.5, 0.5,
    0.1, 0.9),
  nrow = 6, ncol = 2, byrow = TRUE)


B3_left <- matrix(
  c(0.9, 0.1, # High probability for living with parents
    0.1, 0.9,
    0.5, 0.5,
    0.5, 0.5,
    0.1, 0.9,
    0.1, 0.9),
  nrow = 6, ncol = 2, byrow = TRUE)

# Starting values for transition matrices
A1 <- matrix(
  c(0.80, 0.16, 0.03, 0.01,
    0,    0.90, 0.07, 0.03,
    0,    0,    0.90, 0.10,
    0,    0,    0,       1),
  nrow = 4, ncol = 4, byrow = TRUE)

A2 <- matrix(
  c(0.80, 0.10, 0.05, 0.03, 0.01, 0.01,
    0,    0.70, 0.10, 0.10, 0.05, 0.05,
    0,    0,    0.85, 0.01, 0.10, 0.04,
    0,    0,    0,    0.90, 0.05, 0.05,
    0,    0,    0,    0,    0.90, 0.10,
    0,    0,    0,    0,    0,       1),
  nrow = 6, ncol = 6, byrow = TRUE)

# Starting values for initial state probabilities
initial_probs1 <- c(0.9, 0.07, 0.02, 0.01)
initial_probs2 <- c(0.9, 0.04, 0.03, 0.01, 0.01, 0.01)

# Birth cohort
biofam3c$covariates$cohort <- factor(cut(biofam3c$covariates$birthyr,
   c(1908, 1935, 1945, 1957)), labels = c("1909-1935", "1936-1945", "1946-1957"))

# Build mixture HMM
init_mhmm_bf <- build_mhmm(
  observations = list(marr_seq, child_seq, left_seq),
  initial_probs = list(initial_probs1, initial_probs1, initial_probs2),
  transition_probs = list(A1, A1, A2),
  emission_probs = list(list(B1_marr, B1_child, B1_left),
    list(B2_marr, B2_child, B2_left),
    list(B3_marr, B3_child, B3_left)),
  formula = ~sex + cohort, data = biofam3c$covariates,
  channel_names = c("Marriage", "Parenthood", "Residence"))

# Fitting the model
mhmm_biofam <- fit_model(init_mhmm_bf)$model
```

See Also

Examples of building and fitting MHMMs in `build_mhmm` and `fit_model`; and `biofam` for the original data and `biofam3c` for the three-channel version used in this model.

Examples

```
data("mhmm_biofam")

# use conditional_se = FALSE for more accurate standard errors
# (these are considerebly slower to compute)
summary(mhmm_biofam$model)

if (interactive()) {
  # Plotting the model for each cluster (change with Enter)
  plot(mhmm_biofam)
}
```

---

mhmm_mvad *Mixture hidden Markov model for the mvad data*

Description

A mixture hidden Markov model (MHMM) fitted for the mvad data.

Format

A mixture hidden Markov model of class mhmm: two clusters including 3 and 4 hidden states. No covariates.

Details

The model is loaded by calling data(mhmm_mvad). It was created with the following code:

```
data("mvad", package = "TraMineR")

mvad_alphabet <-
  c("employment", "FE", "HE", "joblessness", "school", "training")
mvad_labels <- c("employment", "further education", "higher education",
  "joblessness", "school", "training")
mvad_scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad_seq <- seqdef(mvad, 17:86, alphabet = mvad_alphabet,
  states = mvad_scodes, labels = mvad_labels, xtstep = 6)

attr(mvad_seq, "cpal") <- colorpalette[[6]]

# Starting values for the emission matrices
emiss_1 <- matrix(
  c(0.01, 0.01, 0.01, 0.01, 0.01, 0.95,
    0.95, 0.01, 0.01, 0.01, 0.01, 0.01,
    0.01, 0.01, 0.01, 0.95, 0.01, 0.01),
  nrow = 3, ncol = 6, byrow = TRUE)
```

```
emiss_2 <- matrix(
  c(0.01, 0.01, 0.01, 0.06, 0.90, 0.01,
    0.01, 0.95, 0.01, 0.01, 0.01, 0.01,
    0.01, 0.01, 0.95, 0.01, 0.01, 0.01,
    0.95, 0.01, 0.01, 0.01, 0.01, 0.01),
  nrow = 4, ncol = 6, byrow = TRUE)

# Starting values for the transition matrix

trans_1 <-  matrix(
  c(0.95, 0.03, 0.02,
    0.01, 0.98, 0.01,
    0.01, 0.01, 0.98),
  nrow = 3, ncol = 3, byrow = TRUE)

trans_2 <-  matrix(
  c(0.97, 0.01, 0.01, 0.01,
    0.01, 0.97, 0.01, 0.01,
    0.01, 0.01, 0.97, 0.01,
    0.01, 0.01, 0.01, 0.97),
  nrow = 4, ncol = 4, byrow = TRUE)

# Starting values for initial state probabilities
initial_probs_1 <- c(0.5, 0.25, 0.25)
initial_probs_2 <- c(0.4, 0.4, 0.1, 0.1)

# Building a hidden Markov model with starting values
init_mhmm_mvad <- build_mhmm(observations = mvad_seq,
  transition_probs = list(trans_1, trans_2),
  emission_probs = list(emiss_1, emiss_2),
  initial_probs = list(initial_probs_1, initial_probs_2))

# Fit the model
set.seed(123)
mhmm_mvad <- fit_model(init_mhmm_mvad, control_em = list(restart = list(times = 25)))$model
```

See Also

Examples of building and fitting MHMMs in build_mhmm and fit_model; and mvad for more information on the data.

Examples

```
data("mhmm_mvad")

summary(mhmm_mvad)

if (interactive()) {
  # Plotting the model for each cluster (change with Enter)
  plot(mhmm_mvad)
}
```

---

mssplot *Interactive Stacked Plots of Multichannel Sequences and/or Most Probable Paths for Mixture Hidden Markov Models*

Description

Function mssplot plots stacked sequence plots of observation sequences and/or most probable hidden state paths for each model of the mhmm object (model chosen according to the most probable path).

Usage

```
mssplot(x, ask = FALSE, which.plots = NULL, hidden.paths = NULL,
  plots = "obs", type = "d", tlim = 0, sortv = NULL, sort.channel = 1,
  dist.method = "OM", with.missing = FALSE, missing.color = NULL,
  title = NA, title.n = TRUE, cex.title = 1, title.pos = 1,
  with.legend = "auto", ncol.legend = "auto",
  with.missing.legend = "auto", legend.prop = 0.3, cex.legend = 1,
  hidden.states.colors = "auto", hidden.states.labels = "auto",
  xaxis = TRUE, xlab = NA, xtlab = NULL, xlab.pos = 1, ylab = "auto",
  hidden.states.title = "Hidden states", yaxis = FALSE, ylab.pos = "auto",
  cex.lab = 1, cex.axis = 1, ...)
```

Arguments

x	Mixture hidden Markov model object of class mhmm.
ask	If TRUE and which.plots is NULL, plot.mhmm operates in interactive mode, via menu. Defaults to FALSE.
which.plots	The number(s) of the requested model(s) as an integer vector. The default NULL produces all plots.
hidden.paths	Output from the hidden_paths function. The default value NULL computes hidden paths automatically, if needed.
plots	What to plot. One of "obs" for observations (the default), "hidden.paths" for most probable paths of hidden states, or "both" for observations and hidden paths together.
type	The type of the plot. Available types are "I" for index plots and "d" for state distribution plots (the default). See seqplot for details.
tlim	Indexes of the subjects to be plotted (the default is 0, i.e. all subjects are plotted). For example, tlim = 1:10 plots the first ten subjects in data.

`sortv`	A sorting variable or a sort method (one of `"from.start"`, `"from.end"`, `"mds.obs"`, or `"mds.hidden"`) for `type = "I"`. The value `"mds.hidden"` is only available when `which = "both"` and `which = "hidden.paths"`. Options `"mds.obs"` and `"mds.hidden"` automatically arrange the sequences according to the scores of multidimensional scaling (using `cmdscale`) for the observed data or hidden states paths. MDS scores are computed from distances/dissimilarities using a metric defined in argument `dist.method`. See `plot.stslist` for more details on `"from.start"` and `"from.end"`.
`sort.channel`	The number of the channel according to which the `"from.start"` or `"from.end"` sorting is done. Sorting according to hidden states is called with value 0. The default value is 1 (the first channel).
`dist.method`	The metric to be used for computing the distances of the sequences if multidimensional scaling is used for sorting. One of "OM" (optimal matching, the default), "LCP" (longest common prefix), "RLCP" (reversed LCP, i.e. longest common suffix), "LCS" (longest common subsequence), "HAM" (Hamming distance), and "DHD" (dynamic Hamming distance). Transition rates are used for defining substitution costs if needed. See `seqdef` for more information on the metrics.
`with.missing`	Controls whether missing states are included in state distribution plots (`type = "d"`). The default is `FALSE`.
`missing.color`	Alternative color for representing missing values in the sequences. By default, this color is taken from the `missing.color` attribute of the sequence object.
`title`	A vector of main titles for the graphics. The default is `NA`: if `title.n = TRUE`, the name of the cluster and the number of subjects is plotted. `FALSE` prints no titles, even when `title.n = TRUE`.
`title.n`	Controls whether the number of subjects is printed in the main titles of the plots. The default is `TRUE`: n is plotted if `title` is anything but `FALSE`.
`cex.title`	Expansion factor for setting the size of the font for the main titles. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`title.pos`	Controls the position of the main titles of the plots. The default value is 1. Values greater than 1 will place the title higher.
`with.legend`	Defines if and where the legend for the states is plotted. The default value `"auto"` (equivalent to `TRUE` and `"right"`) creates separate legends for each requested plot and positiones them on the right-hand side of the plot. Other possible values are `"bottom"`, `"right.combined"`, and `"bottom.combined"`, of which the last two create a combined legend in the selected position. `FALSE` prints no legend.
`ncol.legend`	(A vector of) the number of columns for the legend(s). The default `"auto"` creates one column for each legend.
`with.missing.legend`	If set to `"auto"` (the default), a legend for the missing state is added automatically if one or more of the sequences in the data/channel contains missing states and `type = "I"`. If `type = "d"` missing states are omitted from the legends unless `with.missing = TRUE`. With the value `TRUE` a legend for the missing

	state is added in any case; equivalently `FALSE` omits the legend for the missing state.
`legend.prop`	Sets the proportion of the graphic area used for plotting the legend when `with.legend` is not `FALSE`. The default value is 0.3. Takes values from 0 to 1.
`cex.legend`	Expansion factor for setting the size of the font for the labels in the legend. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`hidden.states.colors`	A vector of colors assigned to hidden states. The default value `"auto"` uses the colors assigned to the `stslist` object (created with `seqdef`) if `hidden.paths` is given; otherwise colors from `colorpalette` are automatically used.
`hidden.states.labels`	Labels for the hidden states. The default value `"auto"` uses the names provided in `x$state_names` if `x` is an `hmm` object; otherwise the number of the hidden state.
`xaxis`	Controls whether an x-axis is plotted below the plot at the bottom. The default value is `TRUE`.
`xlab`	An optional label for the x-axis. If set to `NA`, no label is drawn.
`xtlab`	Optional labels for the x-axis tick labels. If unspecified, the column names of the `seqdata` sequence object are used (see `seqdef`).
`xlab.pos`	Controls the position of the x-axis label. The default value is 1. Values greater than 1 will place the label further away from the plot.
`ylab`	Labels for the channels shown as labels for y-axes. A vector of names for each channel (observations). The default value `"auto"` uses the names provided in `x$channel_names` if `x` is an `hmm` object; otherwise the names of the list in `x` if given, or the number of the channel if names are not given. `FALSE` prints no labels.
`hidden.states.title`	Optional label for the hidden state plot (in the y-axis). The default is `"Hidden states"`.
`yaxis`	Controls whether or not to plot the y-axis. The default is `FALSE`.
`ylab.pos`	Controls the position of the y axis labels (labels for channels and/or hidden states). Either `"auto"` or a numerical vector indicating how far away from the plots the titles are positioned. The default value `"auto"` positions all titles on line 1. Shorter vectors are recycled.
`cex.lab`	Expansion factor for setting the size of the font for the axis labels. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`cex.axis`	Expansion factor for setting the size of the font for the x-axis tick labels. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`...`	Other arguments to be passed on to `seqplot`.

See Also

build_mhmm and fit_model for building and fitting mixture hidden Markov models, hidden_paths for computing the most probable paths (Viterbi paths) of hidden states, plot.mhmm for plotting mhmm objects as directed graphs, and colorpalette for default colors.

Examples

```
# Loading mixture hidden Markov model (mhmm object)
# of the biofam data
data("mhmm_biofam")

# Plotting the first cluster only
mssplot(mhmm_biofam, which.plots = 1)

if (interactive()) {
  # Interactive plot
  mssplot(mhmm_biofam)
}
```

---

plot.hmm *Plot hidden Markov models*

Description

Function plot.hmm plots a directed graph with pie charts of emission probabilities as vertices/nodes.

Usage

```
## S3 method for class 'hmm'
plot(x, layout = "horizontal", pie = TRUE, vertex.size = 40,
  vertex.label = "initial.probs", vertex.label.dist = "auto",
  vertex.label.pos = "bottom", vertex.label.family = "sans",
  loops = FALSE, edge.curved = TRUE, edge.label = "auto",
  edge.width = "auto", cex.edge.width = 1, edge.arrow.size = 1.5,
  edge.label.family = "sans", label.signif = 2, label.scientific = FALSE,
  label.max.length = 6, trim = 1e-15, combine.slices = 0.05,
  combined.slice.color = "white", combined.slice.label = "others",
  with.legend = "bottom", ltext = NULL, legend.prop = 0.5,
  cex.legend = 1, ncol.legend = "auto", cpal = "auto", main = NULL,
  withlegend, ...)
```

Arguments

x	A hidden Markov model object of class hmm created with build_hmm (or build_mm). Multichannel hmm objects are automatically transformed into single-channel objects. See function mc_to_sc for more information on the transformation.

`layout`	specifies the layout of vertices (nodes). Accepts a numerical matrix, a `layout_` function (without quotation marks), or either of the predefined options `"horizontal"` (the default) and `"vertical"`. Options `"horizontal"` and `"vertical"` position vertices at the same horizontal or vertical line. A two-column numerical matrix can be used to give x and y coordinates of the vertices. The `layout_` functions available in the `igraph` package offer other automatic layouts for graphs.
`pie`	Are vertices plotted as pie charts of emission probabilities? Defaults to TRUE.
`vertex.size`	Size of vertices, given as a scalar or numerical vector. The default value is 40.
`vertex.label`	Labels for vertices. Possible options include `"initial.probs"`, `"names"`, `NA`, and a character or numerical vector. The default `"initial.probs"` prints the initial probabilities of the model and `"names"` prints the names of the hidden states as labels. `NA` prints no labels.
`vertex.label.dist`	Distance of the label of the vertex from its center. The default value `"auto"` places the label outside the vertex.
`vertex.label.pos`	Positions of vertex labels, relative to the center of the vertex. A scalar or numerical vector giving position(s) as radians or one of `"bottom"` (`pi/2` as radians), `"top"` (`-pi/2`), `"left"` (`pi`), or `"right"` (`0`).
`vertex.label.family, edge.label.family`	Font family to be used for vertex/edge labels. See argument `family` in `par` for more information.
`loops`	Defines whether transitions back to same states are plotted.
`edge.curved`	Defines whether to plot curved edges (arcs, arrows) between vertices. A logical or numerical vector or scalar. Numerical values specify curvatures of edges. The default value `TRUE` gives curvature of 0.5 to all edges. See `igraph.plotting` for more information.
`edge.label`	Labels for edges. Possible options include `"auto"`, `NA`, and a character or numerical vector. The default `"auto"` prints transition probabilities as edge labels. `NA` prints no labels.
`edge.width`	Width(s) for edges. The default `"auto"` determines widths according to transition probabilities between hidden states. Other possibilities are a scalar or a numerical vector of widths.
`cex.edge.width`	An expansion factor for edge widths. Defaults to 1.
`edge.arrow.size`	Size of the arrow in edges (constant). Defaults to 1.5.
`label.signif`	Rounds labels of model parameters to specified number of significant digits, 2 by default. Ignored for user-given labels.
`label.scientific`	Defines if scientific notation should be used to describe small numbers. Defaults to `FALSE`, e.g. 0.0001 instead of 1e-04. Ignored for user-given labels.
`label.max.length`	Maximum number of digits in labels of model parameters. Ignored for user-given labels.

`trim`	Scalar between 0 and 1 giving the highest probability of transitions that are plotted as edges, defaults to 1e-15.
`combine.slices`	Scalar between 0 and 1 giving the highest probability of emission probabilities that are combined into one state. The dafault value is 0.05.
`combined.slice.color`	Color of the combined slice that includes the smallest emission probabilities (only if argument `"combine.slices"` is greater than 0). The default color is white.
`combined.slice.label`	The label for combined states (when argument `"combine.slices"` is greater than 0) to appear in the legend.
`with.legend`	Defines if and where the legend of state colors is plotted. Possible values include `"bottom"` (the default), `"top"`, `"left"`, and `"right"`. `FALSE` omits the legend.
`ltext`	Optional description of (combined) observed states to appear in the legend. A vector of character strings. See `seqplot` for more information.
`legend.prop`	Proportion used for plotting the legend. A scalar between 0 and 1, defaults to 0.5.
`cex.legend`	Expansion factor for setting the size of the font for labels in the legend. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`ncol.legend`	The number of columns for the legend. The default value `"auto"` sets the number of columns automatically.
`cpal`	Optional color palette for (combinations of) observed states. The default value `"auto"` uses automatic color palette. Otherwise a vector of length `x$n_symbols` is given, i.e. the argument requires a color specified for all (combinations of) observed states even if they are not plotted (if the probability is less than `combine.slices`).
`main`	Main title for the plot. Omitted by default.
`withlegend`	Deprecated. Use `with.legend` instead.
`...`	Other parameters passed on to `plot.igraph` such as `vertex.color`, `vertex.label.cex`, or `edge.lty`.

See Also

`build_hmm` and `fit_model` for building and fitting Hidden Markov models, `mc_to_sc` for transforming multistate `hmm` objects into single-channel objects, `hmm_biofam` and `hmm_mvad` for information on the models used in the examples, and `plot.igraph` for the general plotting function of directed graphs.

Examples

```
# Multichannel data, left-to-right model

# Loading a HMM of the biofam data
data("hmm_biofam")

# Plotting hmm object
```

```
plot(hmm_biofam)

# Plotting HMM with
plot(hmm_biofam,
  # varying curvature of edges
  edge.curved = c(0, -0.7, 0.6, 0.7, 0, -0.7, 0),
  # legend with two columns and less space
  ncol.legend = 2, legend.prop = 0.4,
  # new label for combined slice
  combined.slice.label = "States with probability < 0.05")

# Plotting HMM with given coordinates
plot(hmm_biofam,
  # layout given in 2x5 matrix
  # x coordinates in the first column
  # y coordinates in the second column
  layout = matrix(c(1, 3, 3, 5,  3,
                    0, 0, 1, 0, -1), ncol = 2),
  # larger vertices
  vertex.size = 50,
  # straight edges
  edge.curved = FALSE,
  # thinner edges and arrows
  cex.edge.width = 0.5, edge.arrow.size = 1,
  # varying positions for vertex labels (initial probabilities)
  vertex.label.pos = c(pi, pi/2, -pi/2, 0, pi/2),
  # different legend properties
  with.legend = "top", legend.prop = 0.3, cex.legend = 1.1,
  # Fix axes to the right scale
  xlim = c(0.5, 5.5), ylim = c(-1.5, 1.5), rescale = FALSE,
  # all states (not combining states with small probabilities)
  combine.slices = 0,
  # legend with two columns
  ncol.legend = 2)

# Plotting HMM with own color palette
plot(hmm_biofam, cpal = 1:10,
  # States with emission probability less than 0.2 removed
  combine.slices = 0.2,
  # legend with two columns
  ncol.legend = 2)

# Plotting HMM without pie graph and with a layout function
require("igraph")
# Setting the seed for a random layout
set.seed(1234)
plot(hmm_biofam,
  # Without pie graph
  pie = FALSE,
  # Using an automatic layout function from igraph
  layout = layout_nicely,
  vertex.size = 30,
  # Straight edges and probabilities of moving to the same state
```

```
    edge.curved = FALSE, loops = TRUE,
    # Labels with three significant digits
    label.signif = 3,
    # Fixed edge width
    edge.width = 1,
    # Remove edges with probability less than 0.01
    trim = 0.01,
    # Hidden state names as vertex labels
    vertex.label = "names",
    # Labels insidde vertices
    vertex.label.dist = 0,
    # Fix x-axis (more space on the right-hand side)
    xlim = c(-1, 1.3))


  # Single-channel data, unrestricted model

  # Loading a hidden Markov model of the mvad data (hmm object)
  data("hmm_mvad")

  # Plotting HMM
  plot(hmm_mvad)

  require("igraph")
  plot(hmm_mvad,
    # Layout in circle (layout function from igraph)
    layout = layout_in_circle,
    # Less curved edges with smaller arrows, no labels
    edge.curved = 0.2, edge.arrow.size = 0.9, edge.label = NA,
    # Positioning vertex labels (initial probabilities)
    vertex.label.pos = c("right", "right", "left", "left", "right"),
    # Less space for the legend
    legend.prop = 0.3)
```

plot.mhmm *Interactive Plotting for Mixed Hidden Markov Model (mhmm)*

Description

Function plot.mhmm plots a directed graph of the parameters of each model with pie charts of emission probabilities as vertices/nodes.

Usage

```
## S3 method for class 'mhmm'
plot(x, interactive = TRUE, ask = FALSE,
  which.plots = NULL, nrow = NA, ncol = NA, byrow = FALSE,
  row.prop = "auto", col.prop = "auto", layout = "horizontal",
  pie = TRUE, vertex.size = 40, vertex.label = "initial.probs",
  vertex.label.dist = "auto", vertex.label.pos = "bottom",
  vertex.label.family = "sans", loops = FALSE, edge.curved = TRUE,
  edge.label = "auto", edge.width = "auto", cex.edge.width = 1,
  edge.arrow.size = 1.5, edge.label.family = "sans", label.signif = 2,
  label.scientific = FALSE, label.max.length = 6, trim = 1e-15,
  combine.slices = 0.05, combined.slice.color = "white",
  combined.slice.label = "others", with.legend = "bottom", ltext = NULL,
  legend.prop = 0.5, cex.legend = 1, ncol.legend = "auto",
  cpal = "auto", main = "auto", withlegend, ...)
```

Arguments

`x`	A hidden Markov model object of class `mhmm` created with `build_mhmm` (or `build_mmm` or `build_lcm`). Multichannel `mhmm` objects are automatically transformed into single-channel objects. See function `mc_to_sc` for more information on the transformation.
`interactive`	Whether to plot each cluster in succession or in a grid. Defaults to `TRUE`, i.e. clusters are plotted one after another.
`ask`	If `TRUE` and `which.plots` is NULL, `plot.mhmm` operates in interactive mode, via `menu`. Defaults to `FALSE`. Ignored if `interactive = FALSE`.
`which.plots`	The number(s) of the requested cluster(s) as an integer vector. The default `NULL` produces all plots.
`nrow, ncol`	Optional arguments to arrange plots in a grid. Ignored if `interactive = TRUE`.
`byrow`	Controls the order of plotting in a grid. Defaults to `FALSE`, i.e. plots are arranged column-wise. Ignored if `interactive = TRUE`.
`row.prop`	Sets the proportions of the row heights of the grid. The default value is `"auto"` for even row heights. Takes a vector of values from 0 to 1, with values summing to 1. Ignored if `interactive = TRUE`.
`col.prop`	Sets the proportion of the column heights of the grid. The default value is `"auto"` for even column widths. Takes a vector of values from 0 to 1, with values summing to 1. Ignored if `interactive = TRUE`.
`layout`	specifies the layout of vertices (nodes). Accepts a numerical matrix, a `layout_` function (without quotation marks), or either of the predefined options `"horizontal"` (the default) and `"vertical"`. Options `"horizontal"` and `"vertical"` position vertices at the same horizontal or vertical line. A two-column numerical matrix can be used to give x and y coordinates of the vertices. The `layout_` functions available in the `igraph` package offer other automatic layouts for graphs.
`pie`	Are vertices plotted as pie charts of emission probabilities? Defaults to TRUE.
`vertex.size`	Size of vertices, given as a scalar or numerical vector. The default value is 40.
`vertex.label`	Labels for vertices. Possible options include `"initial.probs"`, `"names"`, `NA`, and a character or numerical vector. The default `"initial.probs"` prints the initial probabilities of the model and `"names"` prints the names of the hidden states as labels. `NA` prints no labels.
`vertex.label.dist`	Distance of the label of the vertex from its center. The default value `"auto"` places the label outside the vertex.

`vertex.label.pos`	Positions of vertex labels, relative to the center of the vertex. A scalar or numerical vector giving position(s) as radians or one of `"bottom"` (`pi/2` as radians), `"top"` (`-pi/2`), `"left"` (`pi`), or `"right"` (`0`).
`vertex.label.family, edge.label.family`	Font family to be used for vertex/edge labels. See argument `family` in `par` for more information.
`loops`	Defines whether transitions back to same states are plotted.
`edge.curved`	Defines whether to plot curved edges (arcs, arrows) between vertices. A logical or numerical vector or scalar. Numerical values specify curvatures of edges. The default value `TRUE` gives curvature of 0.5 to all edges. See `igraph.plotting` for more information.
`edge.label`	Labels for edges. Possible options include `"auto"`, `NA`, and a character or numerical vector. The default `"auto"` prints transition probabilities as edge labels. `NA` prints no labels.
`edge.width`	Width(s) for edges. The default `"auto"` determines widths according to transition probabilities between hidden states. Other possibilities are a scalar or a numerical vector of widths.
`cex.edge.width`	An expansion factor for edge widths. Defaults to 1.
`edge.arrow.size`	Size of the arrow in edges (constant). Defaults to 1.5.
`label.signif`	Rounds labels of model parameters to specified number of significant digits, 2 by default. Ignored for user-given labels.
`label.scientific`	Defines if scientific notation should be used to describe small numbers. Defaults to `FALSE`, e.g. 0.0001 instead of 1e-04. Ignored for user-given labels.
`label.max.length`	Maximum number of digits in labels of model parameters. Ignored for user-given labels.
`trim`	Scalar between 0 and 1 giving the highest probability of transitions that are plotted as edges, defaults to 1e-15.
`combine.slices`	Scalar between 0 and 1 giving the highest probability of emission probabilities that are combined into one state. The dafault value is 0.05.
`combined.slice.color`	Color of the combined slice that includes the smallest emission probabilities (only if argument `"combine.slices"` is greater than 0). The default color is white.
`combined.slice.label`	The label for combined states (when argument `"combine.slices"` is greater than 0) to appear in the legend.
`with.legend`	Defines if and where the legend of state colors is plotted. Possible values include `"bottom"` (the default), `"top"`, `"left"`, and `"right"`. `FALSE` omits the legend.
`ltext`	Optional description of (combined) observed states to appear in the legend. A vector of character strings. See `seqplot` for more information.

`legend.prop`	Proportion used for plotting the legend. A scalar between 0 and 1, defaults to 0.5.
`cex.legend`	Expansion factor for setting the size of the font for labels in the legend. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`ncol.legend`	The number of columns for the legend. The default value `"auto"` sets the number of columns automatically.
`cpal`	Optional color palette for (combinations of) observed states. The default value `"auto"` uses automatic color palette. Otherwise a vector of length `x$n_symbols` is given, i.e. the argument requires a color specified for all (combinations of) observed states even if they are not plotted (if the probability is less than `combine.slices`).
`main`	Optional main titles for plots. The default `"auto"` uses `cluster_names` as titles, `NULL` prints no titles.
`withlegend`	Deprecated. Use `with.legend` instead.
`...`	Other parameters passed on to `plot.igraph` such as `vertex.color`, `vertex.label.cex`, or `edge.lty`.

See Also

`build_mhmm` and `fit_model` for building and fitting mixture hidden Markov models; `plot.igraph` for plotting directed graphs; and `mhmm_biofam` and `mhmm_mvad` for the models used in examples.

Examples

```
# Loading mixture hidden Markov model (mhmm object)
# of the biofam data
data("mhmm_biofam")

# Plotting only the first cluster
plot(mhmm_biofam, which.plots = 1)

if (interactive()) {
  # Plotting each cluster (change with Enter)
  plot(mhmm_biofam)

  # Choosing the cluster (one at a time)
  plot(mhmm_biofam, ask = TRUE)

  # Loading MHMM of the mvad data
  data("mhmm_mvad")

  # Plotting models in the same graph (in a grid)
  # Note: the plotting window must be high enough!
  set.seed(123)
  plot(mhmm_mvad, interactive = FALSE,
    # automatic layout, legend on the right-hand side
    layout = layout_nicely, with.legend = "right",
    # Smaller and less curved edges
    edge.curved = 0.2, cex.edge.width = 0.5, edge.arrow.size = 0.7,
    vertex.label.pos = -4 * pi / 5, vertex.label.dist = 5)
}
```

---

plot.ssp *Stack Multichannel Sequence Plots and/or Most Probable Paths Plots from Hidden Markov Models*

Description

Function `plot.ssp` plots stacked sequence plots from `ssp` objects defined with `ssp`.

Usage

```
## S3 method for class 'ssp'
plot(x, ...)
```

Arguments

`x`	An ssp object.
`...`	Ignored.

See Also

`ssp` for more examples and information on defining the plot before using `plot.ssp`; `ssplot` for straight plotting of `ssp` objects; and `gridplot` for plotting multiple `ssp` objects.

Examples

```
data("biofam3c")

## Building sequence objects
child_seq <- seqdef(biofam3c$children, start = 15)
marr_seq <- seqdef(biofam3c$married, start = 15)
left_seq <- seqdef(biofam3c$left, start = 15)

## Choosing colors
attr(child_seq, "cpal") <- c("#66C2A5", "#FC8D62")
attr(marr_seq, "cpal") <- c("#AB82FF", "#E6AB02", "#E7298A")
attr(left_seq, "cpal") <- c("#A6CEE3", "#E31A1C")


# Plotting state distribution plots of observations
ssp1 <- ssp(list(child_seq, marr_seq, left_seq))
plot(ssp1)
```

plot_colors *Plot Colorpalettes*

Description

Function `plot_colors` plots colors and their labels for easy visualization of a colorpalette.

Usage

```
plot_colors(x, labels = NULL)
```

Arguments

`x`	A vector of colors.
`labels`	A vector of labels for colors. If omitted, given color names are used.

See Also

See e.g. the `colorpalette` data and `RColorBrewer` package for ready-made color palettes.

Examples

```
plot_colors(colorpalette[[5]], labels = c("one", "two", "three", "four", "five"))

plot_colors(colorpalette[[10]])

plot_colors(1:7)

plot_colors(c("yellow", "orange", "red", "purple", "blue", "green"))

plot_colors(rainbow(15))
```

posterior_probs *Posterior Probabilities for (Mixture) Hidden Markov Models*

Description

Function `posterior_probs` computes the posterior probabilities of hidden states of a (mixture) hidden Markov model.

Usage

```
posterior_probs(model, log_space = FALSE)
```

Arguments

`model`	A (mixture) hidden Markov model of class `hmm` or `mhmm`.
`log_space`	Compute posterior probabilities in logarithmic scale. The default is `FALSE`.

Value

Posterior probabilities. In case of multiple observations, these are computed independentlsy for each sequence.

Examples

```
# Load a pre-defined MHMM
data("mhmm_biofam")

# Compute posterior probabilities
pb <- posterior_probs(mhmm_biofam)

# Locally most probable states for the first subject:
pb[, , 1]
```

`print.hmm` *Print Method for a Hidden Markov Model*

Description

Prints the parameters of a (mixture) hidden Markov model.

Usage

```
## S3 method for class 'hmm'
print(x, digits = 3, ...)

## S3 method for class 'mhmm'
print(x, digits = 3, ...)

## S3 method for class 'summary.mhmm'
print(x, digits = 3, ...)
```

Arguments

`x`	Hidden Markov model of class `hmm` or `mhmm`.
`digits`	Minimum number of significant digits to print.
`...`	Further arguments to `print.default`.

See Also

`build_hmm` and `fit_model` for building and fitting hidden Markov models.

separate_mhmm — *Reorganize a mixture hidden Markov model to a list of separate hidden Markov models (covariates ignored)*

Description

The `separate_mhmm` function reorganizes the parameters of a `mhmm` object into a list where each list component is an object of class `hmm` consisting of the parameters of the corresponding cluster.

Usage

```
separate_mhmm(model)
```

Arguments

`model`	Mixture hidden Markov model of class `mhmm`.

Value

List with components of class `hmm`.

See Also

`build_mhmm` and `fit_model` for building and fitting MHMMs; and `mhmm_biofam` for more information on the model used in examples.

Examples

```
# Loading mixture hidden Markov model (mhmm object)
# of the biofam data
data("mhmm_biofam")

# Separate models for clusters
sep_hmm <- separate_mhmm(mhmm_biofam)

# Plotting the model for the first cluster
plot(sep_hmm[[1]])
```

seqdef — *Imported Functions from* `TraMineR`

Description

Imported functions for convinience. For details, see the corresponding help pages of `seqstatf`, `alphabet` and `seqdef`.

`seqHMM` *The seqHMM package*

Description

The seqHMM package is designed for fitting hidden (or latent) Markov models (HMMs) and mixture hidden Markov models (MHMMs) for social sequence data and other categorical time series. The package supports models for one or multiple subjects with one or multiple interdependent sequences (channels). External covariates can be added to explain cluster membership in mixture models. The package provides functions for evaluating and comparing models, as well as functions for easy plotting of multichannel sequences and hidden Markov models. Common restricted versions of (M)HMMs are also supported, namely Markov models, mixture Markov models, and latent class models.

Details

Maximum likelihood estimation via the EM algorithm and direct numerical maximization with analytical gradients is supported. All main algorithms are written in C++. Parallel computation is implemented via OpenMP.

`seqHMM-deprecated` *Deprecated function(s) in the seqHMM package*

Description

These functions are provided for compatibility with older version of the seqHMM package. They will be eventually completely removed.

Usage

```
fit_hmm(model, em_step = TRUE, global_step = FALSE, local_step = FALSE,
  control_em = list(), control_global = list(), control_local = list(),
  lb, ub, threads = 1, log_space = FALSE, ...)

fit_mhmm(model, em_step = TRUE, global_step = FALSE, local_step = FALSE,
  control_em = list(), control_global = list(), control_local = list(),
  lb, ub, threads = 1, log_space = FALSE, ...)

trim_hmm(model, maxit = 0, return_loglik = FALSE, zerotol = 1e-08,
  verbose = TRUE, ...)
```

Arguments

`model` An object of class `hmm` or `mhmm`.

`em_step` Logical. Whether or not to use the EM algorithm at the start of the parameter estimation. The default is `TRUE`.

`global_step` Logical. Whether or not to use global optimization via `nloptr` (possibly after the EM step). The default is `FALSE`.

`local_step` Logical. Whether or not to use local optimization via `nloptr` (possibly after the EM and/or global steps). The default is `FALSE`.

`control_em` Optional list of control parameters for the EM algorithm. Possible arguments are

- **maxeval** The maximum number of iterations, the default is 1000.
- **print_level** The level of printing. Possible values are 0 (prints nothing), 1 (prints information at the start and the end of the algorithm), 2 (prints at every iteration), and for mixture models 3 (print also during optimization of coefficients).
- **reltol** Relative tolerance for convergence defined as $(logLik_new - logLik_old)/(abs(logLik_old) + 0.1)$. The default is 1e-10.
- **restart** A list containing options for possible EM restarts with the following components:
  - **times** Number of restarts of the EM algorithm using random initial values. The default is 0, i.e. no restarts.
  - **transition** Logical. Should the original transition probabilities be varied? The default is `TRUE`.
  - **emission** Logical. Should the original emission probabilities be varied? The default is `TRUE`.
  - **sd** Standard deviation for `rnorm` used in randomization. The default is 0.25.
  - **maxeval** Maximum number of iterations, the default is `control_em$maxeval`
  - **print_level** Level of printing in restarted EM steps. The default is `control_em$print_level`.
  - **reltol** Relative tolerance for convergence at restarted EM steps. The default is `control_em$reltol`. If the relative change of the final model of the restart phase is larger than the tolerance for the original EM phase, the final model is re-estimated with the original `reltol` and `maxeval` at the end of the EM step.
  - **n_optimum** Save the log-likelihood values of the `n_optimum` best models (from all estimated models including the the first EM run.). The default is `min(times + 1, 25)`.
  - **use_original** If `TRUE`. Use the initial values of the input model as starting points for the permutations. Otherwise permute the results of the first EM run.

`control_global` Optional list of additional arguments for `nloptr` argument `opts`. The default values are

- **algorithm** `"NLOPT_GD_MLSL_LDS"`
- **local_opts** `list(algorithm = "NLOPT_LD_LBFGS", ftol_rel = 1e-6, xtol_rel = 1e-4)`

	maxeval `10000` (maximum number of iterations in global optimization algorithm.)
	maxtime `60` (maximum time for global optimization. Set to 0 for unlimited time.)
`control_local`	Optional list of additional arguments for `nloptr` argument `opts`. The default values are **algorithm** `"NLOPT_LD_LBFGS"` **ftol_rel** `1e-10` **xtol_rel** `1e-8` **maxeval** `10000` (maximum number of iterations)
`lb`	Lower and upper bounds for parameters in Softmax parameterization. The default interval is $[pmin(-25, 2*initialvalues), pmax(25, 2*initialvalues)]$, except for gamma coefficients, where the scale of covariates is taken into account. Note that it might still be a good idea to scale covariates around unit scale. Bounds are used only in the global optimization step.
`ub`	Lower and upper bounds for parameters in Softmax parameterization. The default interval is $[pmin(-25, 2*initialvalues), pmax(25, 2*initialvalues)]$, except for gamma coefficients, where the scale of covariates is taken into account. Note that it might still be a good idea to scale covariates around unit scale. Bounds are used only in the global optimization step.
`threads`	Number of threads to use in parallel computing. The default is 1.
`log_space`	Make computations using log-space instead of scaling for greater numerical stability at a cost of decreased computational performance. The default is `FALSE`.
`...`	Additional arguments to `nloptr`.
`maxit`	Number of iterations. After zeroing small values, the model is refitted, and this is repeated until there is nothing to trim or `maxit` iterations are done.
`return_loglik`	Return the log-likelihood of the trimmed model together with the model object. The default is `FALSE`.
`zerotol`	Values smaller than this are trimmed to zero.
`verbose`	Print results of trimming. The default is `TRUE`.

`simulate_hmm` *Simulate hidden Markov models*

Description

Simulate sequences of observed and hidden states given parameters of a hidden Markov model.

Usage

```
simulate_hmm(n_sequences, initial_probs, transition_probs, emission_probs,
  sequence_length)
```

Arguments

`n_sequences`	Number of simulations.
`initial_probs`	A vector of initial state probabilities.
`transition_probs`	A matrix of transition probabilities.
`emission_probs`	A matrix of emission probabilities or a list of such objects (one for each channel).
`sequence_length`	Length for simulated sequences.

Value

A list of state sequence objects of class `stslist`.

See Also

`build_hmm` and `fit_model` for building and fitting hidden Markov models; `ssplot` for plotting multiple sequence data sets; `seqdef` for more information on state sequence objects; and `simulate_mhmm` for simulating mixture hidden Markov models.

Examples

```
# Parameters for the HMM
emission_probs <- matrix(c(0.5, 0.2, 0.5, 0.8), 2, 2)
transition_probs <- matrix(c(5/6, 1/6, 1/6, 5/6), 2, 2)
initial_probs <- c(1, 0)

# Setting the seed for simulation
set.seed(1)

# Simulating sequences
sim <- simulate_hmm(
  n_sequences = 10, initial_probs = initial_probs,
  transition_probs = transition_probs,
  emission_probs = emission_probs,
  sequence_length = 20)

ssplot(sim, sortv = "mds.obs", type = "I")
```

`simulate_initial_probs` *Simulate Parameters of Hidden Markov Models*

Description

These are helper functions for quick construction of initial values for various model building functions. Mostly useful for global optimization algorithms which do not depend on initial values.

Usage

```
simulate_initial_probs(n_states, n_clusters = 1)

simulate_transition_probs(n_states, n_clusters = 1, left_right = FALSE,
  diag_c = 0)

simulate_emission_probs(n_states, n_symbols, n_clusters = 1)
```

Arguments

`n_states`	Number of states in each cluster.
`n_clusters`	Number of clusters.
`left_right`	Constrain the transition probabilities to upper triangular. Default is `FALSE`.
`diag_c`	A constant value to be added to diagonal of transition matrices before scaling.
`n_symbols`	Number of distinct symbols in each channel.

See Also

`build_hmm`, `build_mhmm`, `build_mm`, `build_mmm`, and `build_lcm` for constructing different types of models.

`simulate_mhmm` *Simulate Mixture Hidden Markov Models*

Description

Simulate sequences of observed and hidden states given the parameters of a mixture hidden Markov model.

Usage

```
simulate_mhmm(n_sequences, initial_probs, transition_probs, emission_probs,
  sequence_length, formula, data, coefficients)
```

Arguments

`n_sequences`	The number of simulations.
`initial_probs`	A list containing vectors of initial state probabilities for the submodel of each cluster.
`transition_probs`	A list of matrices of transition probabilities for the submodel of each cluster.
`emission_probs`	A list which contains matrices of emission probabilities or a list of such objects (one for each channel) for the submodel of each cluster. Note that the matrices must have dimensions sxm where s is the number of hidden states and m is the number of unique symbols (observed states) in the data.

`sequence_length`	The length of the simulated sequences.
`formula`	Covariates as an object of class `formula`, left side omitted.
`data`	An optional data frame, a list or an environment containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`.
`coefficients`	An optional kxl matrix of regression coefficients for time-constant covariates for mixture probabilities, where l is the number of clusters and k is the number of covariates. A logit-link is used for mixture probabilities. The first column is set to zero.

Value

A list of state sequence objects of class `stslist`.

See Also

`build_mhmm` and `fit_model` for building and fitting mixture hidden Markov models; `ssplot` for plotting multiple sequence data sets; `seqdef` for more information on state sequence objects; and `simulate_hmm` for simulating hidden Markov models.

Examples

```
emission_probs_1 <- matrix(c(0.75, 0.05, 0.25, 0.95), 2, 2)
emission_probs_2 <- matrix(c(0.1, 0.8, 0.9, 0.2), 2, 2)
colnames(emission_probs_1) <- colnames(emission_probs_2) <-
  c("heads", "tails")

transition_probs_1 <- matrix(c(9, 0.1, 1, 9.9) / 10, 2, 2)
transition_probs_2 <- matrix(c(35, 1, 1, 35) / 36, 2, 2)
rownames(emission_probs_1) <- rownames(transition_probs_1) <-
  colnames(transition_probs_1) <- c("coin 1", "coin 2")
rownames(emission_probs_2) <- rownames(transition_probs_2) <-
  colnames(transition_probs_2) <- c("coin 3", "coin 4")

initial_probs_1 <- c(1, 0)
initial_probs_2 <- c(1, 0)

n <- 50
set.seed(123)
covariate_1 <- runif(n)
covariate_2 <- sample(c("A", "B"), size = n, replace = TRUE,
  prob = c(0.3, 0.7))
dataf <- data.frame(covariate_1, covariate_2)

coefs <- cbind(cluster_1 = c(0, 0, 0), cluster_2 = c(-1.5, 3, -0.7))
rownames(coefs) <- c("(Intercept)", "covariate_1", "covariate_2B")

sim <- simulate_mhmm(
  n = n, initial_probs = list(initial_probs_1, initial_probs_2),
  transition_probs = list(transition_probs_1, transition_probs_2),
  emission_probs = list(emission_probs_1, emission_probs_2),
  sequence_length = 25, formula = ~covariate_1 + covariate_2,
  data = dataf, coefficients = coefs)

ssplot(sim$observations, hidden.paths = sim$states, plots = "both",
  sortv = "from.start", sort.channel = 0, type = "I")

hmm <- build_mhmm(sim$observations,
  initial_probs = list(initial_probs_1, initial_probs_2),
  transition_probs = list(transition_probs_1, transition_probs_2),
  emission_probs = list(emission_probs_1, emission_probs_2),
  formula = ~covariate_1 + covariate_2,
  data = dataf)

fit <- fit_model(hmm)
fit$model

paths <- hidden_paths(fit$model)

ssplot(list(estimates = paths, true = sim$states), sortv = "from.start",
  sort.channel = 2, ylab = c("estimated paths", "true (simulated)"),
  type = "I")
```

ssp   *Define Arguments for Plotting Multichannel Sequences and/or Most Probable Paths from Hidden Markov Models*

Description

Function ssp defines the arguments for plotting with plot.ssp or gridplot.

Usage

```
ssp(x, hidden.paths = NULL, plots = "obs", type = "d", tlim = 0,
  sortv = NULL, sort.channel = 1, dist.method = "OM",
  with.missing = FALSE, missing.color = NULL, title = NA,
  title.n = TRUE, cex.title = 1, title.pos = 1, with.legend = "auto",
  ncol.legend = "auto", with.missing.legend = "auto", legend.prop = 0.3,
  cex.legend = 1, hidden.states.colors = "auto",
  hidden.states.labels = "auto", xaxis = TRUE, xlab = NA, xtlab = NULL,
  xlab.pos = 1, ylab = "auto", hidden.states.title = "Hidden states",
  yaxis = FALSE, ylab.pos = "auto", cex.lab = 1, cex.axis = 1,
  withlegend, ...)
```

Arguments

x	Either a hidden Markov model object of class hmm or a state sequence object of class stslist (created with the seqdef) function) or a list of state sequence objects.

`hidden.paths`	Output from `hidden_paths` function. Optional, if `x` is a `hmm` object or if `type = "obs"`.
`plots`	What to plot. One of `"obs"` for observations (the default), `"hidden.paths"` for most probable paths of hidden states, or `"both"` for observations and hidden paths together.
`type`	The type of the plot. Available types are `"I"` for sequence index plots and `"d"` for state distribution plots (the default). See `seqplot` for details.
`tlim`	Indexes of the subjects to be plotted (the default is 0, i.e. all subjects are plotted). For example, `tlim = 1:10` plots the first ten subjects in data.
`sortv`	A sorting variable or a sort method (one of `"from.start"`, `"from.end"`, `"mds.obs"`, or `"mds.hidden"`) for `type = "I"`. The value `"mds.hidden"` is only available when hidden paths are available. Options `"mds.obs"` and `"mds.hidden"` automatically arrange the sequences according to the scores of multidimensional scaling (using `cmdscale`) for the observed data or hidden states paths. MDS scores are computed from distances/dissimilarities using a metric defined in argument `dist.method`. See `plot.stslist` for more details on `"from.start"` and `"from.end"`.
`sort.channel`	The number of the channel according to which the `"from.start"` or `"from.end"` sorting is done. Sorting according to hidden states is called with value 0. The default value is 1 (the first channel).
`dist.method`	The metric to be used for computing the distances of the sequences if multidimensional scaling is used for sorting. One of "OM" (optimal matching, the default), "LCP" (longest common prefix), "RLCP" (reversed LCP, i.e. longest common suffix), "LCS" (longest common subsequence), "HAM" (Hamming distance), and "DHD" (dynamic Hamming distance). Transition rates are used for defining substitution costs if needed. See `seqdef` for more information on the metrics.
`with.missing`	Controls whether missing states are included in state distribution plots (`type = "d"`). The default is `FALSE`.
`missing.color`	Alternative color for representing missing values in the sequences. By default, this color is taken from the `missing.color` attribute of the sequence object.
`title`	Main title for the graphic. The default is `NA`: if `title.n = TRUE`, only the number of subjects is plotted. `FALSE` prints no title, even when `title.n = TRUE`.
`title.n`	Controls whether the number of subjects (in the first channel) is printed in the title of the plot. The default is `TRUE`: n is plotted if `title` is anything but `FALSE`.
`cex.title`	Expansion factor for setting the size of the font for the title. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`title.pos`	Controls the position of the main title of the plot. The default value is 1. Values greater than 1 will place the title higher.
`with.legend`	Defines if and where the legend for the states is plotted. The default value `"auto"` (equivalent to `TRUE` and `"right"`) creates separate legends for each requested plot and positiones them on the right-hand side of the plot. Other possible values are `"bottom"`, `"right.combined"`, and `"bottom.combined"`, of which the last two create a combined legend in the selected position. `FALSE` prints no legend.

`ncol.legend`	(A vector of) the number of columns for the legend(s). The default `"auto"` determines number of columns depending on the position of the legend.
`with.missing.legend`	If set to `"auto"` (the default), a legend for the missing state is added automatically if one or more of the sequences in the data/channel contains missing states and `type = "I"`. If `type = "d"` missing states are omitted from the legends unless `with.missing = TRUE`. With the value `TRUE` a legend for the missing state is added in any case; equivalently `FALSE` omits the legend for the missing state.
`legend.prop`	Sets the proportion of the graphic area used for plotting the legend when `with.legend` is not `FALSE`. The default value is 0.3. Takes values from 0 to 1.
`cex.legend`	Expansion factor for setting the size of the font for the labels in the legend. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`hidden.states.colors`	A vector of colors assigned to hidden states. The default value `"auto"` uses the colors assigned to the `stslist` object (created with `seqdef`) if `hidden.paths` is given; otherwise colors from `colorpalette` are automatically used.
`hidden.states.labels`	Labels for the hidden states. The default value `"auto"` uses the names provided in `x$state_names` if `x` is an `hmm` object; otherwise the number of the hidden state.
`xaxis`	Controls whether an x-axis is plotted below the plot at the bottom. The default value is `TRUE`.
`xlab`	An optional label for the x-axis. If set to `NA`, no label is drawn.
`xtlab`	Optional labels for the x-axis tick labels. If unspecified, the column names of the `seqdata` sequence object are used (see `seqdef`).
`xlab.pos`	Controls the position of the x-axis label. The default value is 1. Values greater than 1 will place the label further away from the plot.
`ylab`	Labels for the channels shown as labels for y-axes. A vector of names for each channel (observations). The default value `"auto"` uses the names provided in `x$channel_names` if `x` is an `hmm` object; otherwise the names of the list in `x` if given, or the number of the channel if names are not given. `FALSE` prints no labels.
`hidden.states.title`	Optional label for the hidden state plot (in the y-axis). The default is `"Hidden states"`.
`yaxis`	Controls whether or not to plot the y-axis. The default is `FALSE`.
`ylab.pos`	Controls the position of the y axis labels (labels for channels and/or hidden states). Either `"auto"` or a numerical vector indicating how far away from the plots the titles are positioned. The default value `"auto"` positions all titles on line 1. Shorter vectors are recycled.
`cex.lab`	Expansion factor for setting the size of the font for the axis labels. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.

`cex.axis`	Expansion factor for setting the size of the font for the x-axis tick labels. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`withlegend`	Deprecated. Use `with.legend` instead.
`...`	Other arguments to be passed on to `seqplot`.

Value

Object of class `ssp`.

See Also

`plot.ssp` for plotting objects created with the `ssp` function; `gridplot` for plotting multiple `ssp` objects; `build_hmm` and `fit_model` for building and fitting hidden Markov models; `hidden_paths` for computing the most probable paths of hidden states; and `biofam3c` and `hmm_biofam` for information on the data and model used in the example.

Examples

```
data("biofam3c")

## Building sequence objects
child_seq <- seqdef(biofam3c$children, start = 15)
marr_seq <- seqdef(biofam3c$married, start = 15)
left_seq <- seqdef(biofam3c$left, start = 15)

## Choosing colors
attr(child_seq, "cpal") <- c("#66C2A5", "#FC8D62")
attr(marr_seq, "cpal") <- c("#AB82FF", "#E6AB02", "#E7298A")
attr(left_seq, "cpal") <- c("#A6CEE3", "#E31A1C")


# Defining the plot for state distribution plots of observations
ssp1 <- ssp(list("Parenthood" = child_seq, "Marriage" = marr_seq,
                 "Residence" = left_seq))
# Plotting ssp1
plot(ssp1)

## Not run:
# Defining the plot for sequence index plots of observations
ssp2 <- ssp(
  list(child_seq, marr_seq, left_seq), type = "I", plots = "obs",
  # Sorting subjects according to the beginning of the 2nd channel (marr_seq)
  sortv = "from.start", sort.channel = 2,
  # Controlling the size, positions, and names for channel labels
  ylab.pos = c(1, 2, 1), cex.lab = 1, ylab = c("Children", "Married", "Residence"),
  # Plotting without legend
  with.legend = FALSE)
plot(ssp2)

# Plotting hidden Markov models
# Loading data
data("hmm_biofam")

# Plotting observations and most probable hidden states paths
ssp3 <- ssp(
  hmm_biofam, type = "I", plots = "both",
  # Sorting according to multidimensional scaling of hidden states paths
  sortv = "mds.hidden",
  # Controlling title
  title = "Biofam", cex.title = 1.5,
  # Labels for x axis and tick marks
  xtlab = 15:30, xlab = "Age")
plot(ssp3)

# Computing the most probable paths of hidden states
hid <- hidden_paths(hmm_biofam)
# Giving names for hidden states
library(TraMineR)
alphabet(hid) <- paste("Hidden state", 1:5)

# Plotting observations and hidden state paths
ssp4 <- ssp(
  hmm_biofam, type = "I", plots = "hidden.paths",
  # Sequence object of most probable paths
  hidden.paths = hid,
  # Sorting according to the end of hidden state paths
  sortv = "from.end", sort.channel = 0,
  # Contolling legend position, type, and proportion
  with.legend = "bottom.combined", legend.prop = 0.15,
  # Plotting without title and y label
  title = FALSE, ylab = FALSE)
plot(ssp4)

## End(Not run)
```

ssplot — *Stacked Plots of Multichannel Sequences and/or Most Probable Paths from Hidden Markov Models*

Description

Function `ssplot` plots stacked sequence plots of sequence object created with the `seqdef` function or observations and/or most probable paths of `hmm` objects.

Usage

```
ssplot(x, hidden.paths = NULL, plots = "obs", type = "d", tlim = 0,
  sortv = NULL, sort.channel = 1, dist.method = "OM",
  with.missing = FALSE, missing.color = NULL, title = NA,
  title.n = TRUE, cex.title = 1, title.pos = 1, with.legend = "auto",
  ncol.legend = "auto", with.missing.legend = "auto", legend.prop = 0.3,
  cex.legend = 1, hidden.states.colors = "auto",
  hidden.states.labels = "auto", xaxis = TRUE, xlab = NA, xtlab = NULL,
  xlab.pos = 1, ylab = "auto", hidden.states.title = "Hidden states",
  yaxis = FALSE, ylab.pos = "auto", cex.lab = 1, cex.axis = 1, ...)
```

Arguments

`x`	Either a hidden Markov model object of class `hmm` or a state sequence object of class `stslist` (created with the `seqdef`) function) or a list of state sequence objects.
`hidden.paths`	Output from `hidden_paths` function. Optional, if `x` is a `hmm` object or if `type = "obs"`.
`plots`	What to plot. One of `"obs"` for observations (the default), `"hidden.paths"` for most probable paths of hidden states, or `"both"` for observations and hidden paths together.
`type`	The type of the plot. Available types are `"I"` for sequence index plots and `"d"` for state distribution plots (the default). See `seqplot` for details.
`tlim`	Indexes of the subjects to be plotted (the default is 0, i.e. all subjects are plotted). For example, `tlim = 1:10` plots the first ten subjects in data.
`sortv`	A sorting variable or a sort method (one of `"from.start"`, `"from.end"`, `"mds.obs"`, or `"mds.hidden"`) for `type = "I"`. The value `"mds.hidden"` is only available when hidden paths are available. Options `"mds.obs"` and `"mds.hidden"` automatically arrange the sequences according to the scores of multidimensional scaling (using `cmdscale`) for the observed data or hidden states paths. MDS scores are computed from distances/dissimilarities using a metric defined in argument `dist.method`. See `plot.stslist` for more details on `"from.start"` and `"from.end"`.
`sort.channel`	The number of the channel according to which the `"from.start"` or `"from.end"` sorting is done. Sorting according to hidden states is called with value 0. The default value is 1 (the first channel).
`dist.method`	The metric to be used for computing the distances of the sequences if multidimensional scaling is used for sorting. One of "OM" (optimal matching, the default), "LCP" (longest common prefix), "RLCP" (reversed LCP, i.e. longest common suffix), "LCS" (longest common subsequence), "HAM" (Hamming distance), and "DHD" (dynamic Hamming distance). Transition rates are used for defining substitution costs if needed. See `seqdef` for more information on the metrics.
`with.missing`	Controls whether missing states are included in state distribution plots (`type = "d"`). The default is `FALSE`.
`missing.color`	Alternative color for representing missing values in the sequences. By default, this color is taken from the `missing.color` attribute of the sequence object.
`title`	Main title for the graphic. The default is `NA`: if `title.n = TRUE`, only the number of subjects is plotted. `FALSE` prints no title, even when `title.n = TRUE`.
`title.n`	Controls whether the number of subjects (in the first channel) is printed in the title of the plot. The default is `TRUE`: n is plotted if `title` is anything but `FALSE`.

`cex.title`	Expansion factor for setting the size of the font for the title. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`title.pos`	Controls the position of the main title of the plot. The default value is 1. Values greater than 1 will place the title higher.
`with.legend`	Defines if and where the legend for the states is plotted. The default value `"auto"` (equivalent to `TRUE` and `"right"`) creates separate legends for each requested plot and positiones them on the right-hand side of the plot. Other possible values are `"bottom"`, `"right.combined"`, and `"bottom.combined"`, of which the last two create a combined legend in the selected position. `FALSE` prints no legend.
`ncol.legend`	(A vector of) the number of columns for the legend(s). The default `"auto"` determines number of columns depending on the position of the legend.
`with.missing.legend`	If set to `"auto"` (the default), a legend for the missing state is added automatically if one or more of the sequences in the data/channel contains missing states and `type = "I"`. If `type = "d"` missing states are omitted from the legends unless `with.missing = TRUE`. With the value `TRUE` a legend for the missing state is added in any case; equivalently `FALSE` omits the legend for the missing state.
`legend.prop`	Sets the proportion of the graphic area used for plotting the legend when `with.legend` is not `FALSE`. The default value is 0.3. Takes values from 0 to 1.
`cex.legend`	Expansion factor for setting the size of the font for the labels in the legend. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`hidden.states.colors`	A vector of colors assigned to hidden states. The default value `"auto"` uses the colors assigned to the `stslist` object (created with `seqdef`) if `hidden.paths` is given; otherwise colors from `colorpalette` are automatically used.
`hidden.states.labels`	Labels for the hidden states. The default value `"auto"` uses the names provided in `x$state_names` if `x` is an `hmm` object; otherwise the number of the hidden state.
`xaxis`	Controls whether an x-axis is plotted below the plot at the bottom. The default value is `TRUE`.
`xlab`	An optional label for the x-axis. If set to `NA`, no label is drawn.
`xtlab`	Optional labels for the x-axis tick labels. If unspecified, the column names of the `seqdata` sequence object are used (see `seqdef`).
`xlab.pos`	Controls the position of the x-axis label. The default value is 1. Values greater than 1 will place the label further away from the plot.
`ylab`	Labels for the channels shown as labels for y-axes. A vector of names for each channel (observations). The default value `"auto"` uses the names provided in `x$channel_names` if `x` is an `hmm` object; otherwise the names of the list in `x` if given, or the number of the channel if names are not given. `FALSE` prints no labels.

`hidden.states.title`	Optional label for the hidden state plot (in the y-axis). The default is `"Hidden states"`.
`yaxis`	Controls whether or not to plot the y-axis. The default is `FALSE`.
`ylab.pos`	Controls the position of the y axis labels (labels for channels and/or hidden states). Either `"auto"` or a numerical vector indicating how far away from the plots the titles are positioned. The default value `"auto"` positions all titles on line 1. Shorter vectors are recycled.
`cex.lab`	Expansion factor for setting the size of the font for the axis labels. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`cex.axis`	Expansion factor for setting the size of the font for the x-axis tick labels. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`...`	Other arguments to be passed on to `seqplot`.

See Also

`ssp` for creating ssp objects and `plot.ssp` and `gridplot` for plotting these; `build_hmm` and `fit_model` for building and fitting hidden Markov models; `hidden_paths` for computing the most probable paths of hidden states; and `biofam3c` `hmm_biofam` for information on the data and model used in the example.

Examples

```
data("biofam3c")

# Creating sequence objects
child_seq <- seqdef(biofam3c$children, start = 15)
marr_seq <- seqdef(biofam3c$married, start = 15)
left_seq <- seqdef(biofam3c$left, start = 15)

## Choosing colors
attr(child_seq, "cpal") <- c("#66C2A5", "#FC8D62")
attr(marr_seq, "cpal") <- c("#AB82FF", "#E6AB02", "#E7298A")
attr(left_seq, "cpal") <- c("#A6CEE3", "#E31A1C")


# Plotting state distribution plots of observations
ssplot(list("Children" = child_seq, "Marriage" = marr_seq,
"Residence" = left_seq))

## Not run:
# Plotting sequence index plots of observations
ssplot(
  list(child_seq, marr_seq, left_seq), type = "I",
  # Sorting subjects according to the beginning of the 2nd channel (marr_seq)
  sortv = "from.start", sort.channel = 2,
  # Controlling the size, positions, and names for channel labels
  ylab.pos = c(1, 2, 1), cex.lab = 1, ylab = c("Children", "Married", "Residence"),
  # Plotting without legend
  with.legend = FALSE)

# Plotting hidden Markov models

# Loading a ready-made HMM for the biofam data
data("hmm_biofam")

# Plotting observations and hidden states paths
ssplot(
  hmm_biofam, type = "I", plots = "both",
  # Sorting according to multidimensional scaling of hidden states paths
  sortv = "mds.hidden",
  ylab = c("Children", "Married", "Left home"),
  # Controlling title
  title = "Biofam", cex.title = 1.5,
  # Labels for x axis and tick marks
  xtlab = 15:30, xlab = "Age")

# Computing the most probable paths of hidden states
hidden.paths <- hidden_paths(hmm_biofam)
hidden.paths_seq <- seqdef(hidden.paths, labels = paste("Hidden state", 1:5))

# Plotting observations and hidden state paths
ssplot(
  hmm_biofam, type = "I", plots = "hidden.paths",
  # Sequence object of most probable paths
  hidden.paths = hidden.paths_seq,
  # Sorting according to the end of hidden state paths
  sortv = "from.end", sort.channel = 0,
  # Contolling legend position, type, and proportion
  with.legend = "bottom", legend.prop = 0.15,
  # Plotting without title and y label
  title = FALSE, ylab = FALSE)

## End(Not run)
```

`summary.mhmm` *Summary method for mixture hidden Markov models*

Description

Function `summary.mhmm` gives a summary of a mixture hidden Markov model.

Usage

```
## S3 method for class 'mhmm'
summary(object, parameters = FALSE, conditional_se = TRUE,
  log_space = FALSE, ...)
```

Arguments

`object`	Mixture hidden Markov model of class `mhmm`.
`parameters`	Whether or not to return transition, emission, and initial probabilities. `FALSE` by default.
`conditional_se`	Return conditional standard errors of coefficients. See `vcov.mhmm` for details. `TRUE` by default.
`log_space`	Make computations using log-space instead of scaling for greater numerical stability at cost of decreased computational performance. Default is `FALSE`.
`...`	Further arguments to `vcov.mhmm`.

Details

The `summary.mhmm` function computes features from a mixture hidden Markov model and stores them as a list. A `print` method prints summaries of these: log-likelihood and BIC, coefficients and standard errors of covariates, means of prior cluster probabilities, and information on most probable clusters.

Value

transition_probs Transition probabilities. Only returned if `parameters = TRUE`.

emission_probs Emission probabilities. Only returned if `parameters = TRUE`.

initial_probs Initial state probabilities. Only returned if `parameters = TRUE`.

logLik Log-likelihood.

BIC Bayesian information criterion.

most_probable_cluster The most probable cluster according to posterior probabilities.

coefficients Coefficients of covariates.

vcov Variance-covariance matrix of coefficients.

prior_cluster_probabilities Prior cluster probabilities (mixing proportions) given the covariates.

posterior_cluster_probabilities Posterior cluster membership probabilities.

classification_table Cluster probabilities (columns) by the most probable cluster (rows).

See Also

`build_mhmm` and `fit_model` for building and fitting mixture hidden Markov models; and `mhmm_biofam` for information on the model used in examples.

Examples

```
# Loading mixture hidden Markov model (mhmm object)
# of the biofam data
data("mhmm_biofam")

# Model summary
summary(mhmm_biofam)
```

trim_model *Trim Small Probabilities of Hidden Markov Model*

Description

Function `trim_model` tries to set small insignificant probabilities to zero without decreasing the likelihood.

Usage

```
trim_model(model, maxit = 0, return_loglik = FALSE, zerotol = 1e-08,
  verbose = TRUE, ...)
```

Arguments

`model`	Model of class `hmm` or `mhmm` for which trimming is performed.
`maxit`	Number of iterations. After zeroing small values, the model is refitted, and this is repeated until there is nothing to trim or `maxit` iterations are done.
`return_loglik`	Return the log-likelihood of the trimmed model together with the model object. The default is `FALSE`.
`zerotol`	Values smaller than this are trimmed to zero.
`verbose`	Print results of trimming. The default is `TRUE`.
`...`	Further parameters passed on to `fit_model`.

See Also

`build_hmm` and `fit_model` for building and fitting hidden Markov models; and `hmm_biofam` for information on the model used in the example.

Examples

```
data("hmm_biofam")

# Testing if changing parameter values smaller than 1e-03 to zero
# leads to improved log-likelihood.
hmm_trim <- trim_model(hmm_biofam, zerotol = 1e-03, maxit = 10)
```

vcov.mhmm *Variance-Covariance Matrix for Coefficients of Covariates of Mixture Hidden Markov Model*

Description

Returns the asymptotic covariances matrix of maximum likelihood estimates of the coefficients corresponding to the explanatory variables of the model.

Usage

```
## S3 method for class 'mhmm'
vcov(object, conditional = TRUE, threads = 1,
  log_space = FALSE, ...)
```

Arguments

`object`	Object of class `mhmm`.
`conditional`	If `TRUE` (default), the standard errors are computed conditional on other model parameters. See details.
`threads`	Number of threads to use in parallel computing. Default is 1.
`log_space`	Make computations using log-space instead of scaling for greater numerical stability at cost of decreased computational performance. Default is `FALSE`.
`...`	Additional arguments to function `jacobian` of `numDeriv` package.

Details

The conditional standard errors are computed using analytical formulas by assuming that the coefficient estimates are not correlated with other model parameter estimates (or that the other parameters are assumed to be fixed). This often underestimates the true standard errors, but is substantially faster approach for preliminary analysis. The non-conditional standard errors are based on the numerical approximation of the full Hessian of the coefficients and the model parameters corresponding to nonzero probabilities. Computing the non-conditional standard errors can be slow for large models as the Jacobian of analytical gradients is computed using finite difference approximation.

Value

Matrix containing the variance-covariance matrix of coefficients.

Index