Package: seqHMM (via r-universe)

October 28, 2026

Title Mixture Hidden Markov Models for Social Sequence Data and Other Multivariate, Multichannel Categorical Time Series

Version 2.2.0

Description Designed for estimating variants of hidden (latent) Markov models (HMMs), mixture HMMs, and non-homogeneous HMMs (NHMMs) for social sequence data and other categorical time series. Special cases include feedback-augmented NHMMs, Markov models without latent layer, 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 as well as initial, transition and emission probabilities in NHMMs. The package provides functions for evaluating and comparing models, as well as functions for visualizing of multichannel sequence data and HMMs. For NHMMs, methods for computing average causal effects and marginal state and emission probabilities are available. Models are estimated using maximum likelihood via the EM algorithm or direct numerical maximization with analytical gradients. Documentation is available via several vignettes, and Helske and Helske (2019, <doi:10.18637/jss.v088.i03>). For methodology behind the NHMMs, see Helske (2025, <doi:10.48550/arXiv.2503.16014>).

LazyData true

LinkingTo nloptr, Rcpp (>= 0.12.0), RcppArmadillo

Depends R (>= 4.1.0)

Imports checkmate, cli, collapse, data.table (>= 1.16.0), future.apply, ggplot2, ggseqplot, graphics, grDevices, grid, gridBase, igraph, lhs, Matrix, methods, nloptr, numDeriv, patchwork, progressr, Rcpp (>= 0.12.0), RcppHungarian, rlang, stats, TraMineR (>= 2.2-7), utils

Suggests covr, knitr, MASS, nnet, testthat (>= 3.0.0),

License GPL (>= 2)

Encoding UTF-8

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

VignetteBuilder knitr

RoxygenNote 7.3.3

Config/testthat/edition 3

SystemRequirements GNU make

Biarch true

NeedsCompilation yes

Author Jouni Helske [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7130-793X>), Satu Helske [aut]
(ORCID: <https://orcid.org/0000-0003-0532-0153>)

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

Config/pak/sysreqs make

Repository https://cran.r-universe.dev

Date/Publication 2026-03-16 08:30:02 UTC

RemoteUrl https://github.com/cran/seqHMM

RemoteRef HEAD

RemoteSha 1a1c950c882426739ac2040f5740353f75ffaa82

Contents

seqHMM-package 4
biofam3c 5
bootstrap_coefs 6
build_hmm 8
build_lcm 12
build_mhmm 16
build_mm 22
build_mmm 24
cluster_names 27
cluster_names<- 27
coef.nhmm 28
colorpalette 28
data_to_stslist 29
estimate_mnhmm 30
estimate_nhmm 33
fanhmm_leaves 36
fit_model 37
forward_backward 49
get_cluster_probs 50
get_emission_probs 51
get_initial_probs 51
get_marginals 52
get_transition_probs 53

gridplot . . . 54
hidden_paths . . . 55
hmm_biofam . . . 56
hmm_mvad . . . 58
leaves . . . 60
logLik.hmm . . . 61
logLik.nhmm . . . 61
mc_to_sc . . . 62
mc_to_sc_data . . . 63
mhmm_biofam . . . 65
mhmm_mvad . . . 68
most_probable_cluster . . . 70
mssplot . . . 71
nobs.hmm . . . 74
permute_states . . . 75
plot.hmm . . . 76
plot.mhmm . . . 81
plot.ssp . . . 85
plot_colors . . . 86
posterior_cluster_probabilities . . . 86
posterior_probs . . . 87
predict.nhmm . . . 88
print.hmm . . . 89
return_msg . . . 90
separate_mhmm . . . 90
seqHMM-deprecated . . . 91
simulate_hmm . . . 91
simulate_initial_probs . . . 92
simulate_mhmm . . . 93
simulate_mnhmm . . . 95
simulate_nhmm . . . 97
sort_sequences . . . 99
ssp . . . 99
ssplot . . . 103
stacked_sequence_plot . . . 106
state_names . . . 108
summary.mhmm . . . 109
summary.mnhmm . . . 110
trim_model . . . 111
update.nhmm . . . 112
vcov.mhmm . . . 112

Index **114**

`seqHMM-package` *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 for pre-2.0.0 functions, while estimation of non-homogenous models support parallelization via `future` package by parallelization of multistart optimizations and bootstrap sampling.

Author(s)

Maintainer: Jouni Helske <jouni.helske@iki.fi> (ORCID)

Authors:

- Satu Helske (ORCID)

References

Helske S. and Helske J. (2019). Mixture Hidden Markov Models for Sequence Data: The seqHMM Package in R, Journal of Statistical Software, 88(3), 1-32. doi:10.18637/jss.v088.i03

See Also

Useful links:

- Report bugs at https://github.com/helske/seqHMM/issues

`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 weight variables.

Details

This data is constructed from the `TraMineR::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) %in% rownames(div) & bf == 7] <- "children"

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) %in% rownames(wp) & bf == 7] <- "with parents"

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

Source

`TraMineR::biofam()` data constructed from the Swiss Household Panel https://forscenter.ch/projects/swiss-household-panel/

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 l a Société francophone de classification (SFC 2007), Paris, 5 - 7 septembre 2007*, pp. 157–160.

`bootstrap_coefs` *Bootstrap Sampling of NHMM Coefficients*

Description

The model estimation for each bootstrap sample uses the same method and tolerances as the original fit. If you want to change these, you can modify the elements of the input model such as `model$estimation_results$method` and `model$controls` before passing it to `bootstrap_coefs()`.

Usage

```
bootstrap_coefs(model, ...)

## S3 method for class 'nhmm'
bootstrap_coefs(
  model,
  nsim,
  type = c("nonparametric", "parametric"),
  append = FALSE,
  ...
)

## S3 method for class 'mnhmm'
bootstrap_coefs(
  model,
  nsim,
  type = c("nonparametric", "parametric"),
  append = FALSE,
  ...
)
```

Arguments

model	An nhmm or mnhmm object.
...	Ignored.
nsim	number of bootstrap samples.
type	Either "nonparametric" (default) or "parametric", to define whether non-parametric or parametric bootstrap should be used. The former samples sequences with replacement, whereas the latter simulates new datasets based on the model.
append	If TRUE, in case the model already contains bootstrap samples, new samples are appended to model$boot. If FALSE (default), old samples are discarded.

Details

It is possible to parallelize the bootstrap runs using the future package, e.g., by calling future::plan(multisession, workers = 2) before bootstrap_coefs(). See future::plan() for details.

bootstrap_coefs() is compatible with progressr package, so you can use progressr::with_progress(bootstrap_coef
to track the progress of bootstrapping.

Value

The original model with additional element model$boot.

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 TraMineR::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, or 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.
- sequence_lengths
  A vector of sequence lengths.
- 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, 15:86,
  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"),
  cpal = c("violetred2", "darkgoldenrod2", "darkmagenta")
)
child_seq <- seqdef(biofam3c$children,
  start = 15,
  alphabet = c("childless", "children"),
  cpal = c("darkseagreen1", "coral3")
)
left_seq <- seqdef(biofam3c$left,
  start = 15,
  alphabet = c("with parents", "left home"),
  cpal = c("lightblue", "red3")
)

# You could also define the colors using cpal function from TraMineR
# cpal(marr_seq) <- c("violetred2", "darkgoldenrod2", "darkmagenta")
# cpal(child_seq) <- c("darkseagreen1", "coral3")
# cpal(left_seq) <- 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 = NULL,
  data = NULL,
  coefficients = NULL,
  cluster_names = NULL,
  channel_names = NULL
)
```

Arguments

observations	An stslist object (see TraMineR::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	Optional formula of class formula() for the mixture probabilities. Left side omitted.
data	A data frame containing the variables used in the formula. Ignored if no formula is provided.
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.
- sequence_lengths
  A vector of sequence lengths.
- 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], 500, TRUE, prob = c(0.1, 0.6, 0.3)), 50, 10),
  matrix(sample(letters[1:3], 200, TRUE, prob = c(0.4, 0.4, 0.2)), 20, 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, 15:86,
```

```
  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) + stats::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 = NULL,
  data = NULL,
  coefficients = NULL,
  cluster_names = NULL,
  state_names = NULL,
  channel_names = NULL,
  ...
)
```

Arguments

observations	An stslist object (see TraMineR::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, or 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	Optional formula of class formula() for the mixture probabilities. Left side omitted.
data	A data frame containing the variables used in the formula. Ignored if no formula is provided.
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.
- `sequence_lengths`
  A vector of sequence lengths.
- `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.

References

Helske S. and Helske J. (2019). Mixture Hidden Markov Models for Sequence Data: The seqHMM Package in R, Journal of Statistical Software, 88(3), 1-32. doi:10.18637/jss.v088.i03

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"),
```

```
  cpal = c("#AB82FF", "#E6AB02", "#E7298A")
)
child_seq <- seqdef(biofam3c$children,
  start = 15,
  alphabet = c("childless", "children"),
  cpal = c("#66C2A5", "#FC8D62")
)
left_seq <- seqdef(biofam3c$left,
  start = 15,
  alphabet = c("with parents", "left home"),
  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 TraMineR::seqdef()) containing the sequences.

Details

Unlike the other build functions in seqHMM, the build_mm() function automatically estimates the model parameters. In case of no missing values, initial and transition probabilities are directly estimated from the observed initial state probabilities and transition counts. In case of missing values, the EM algorithm is run once.

Note that it is possible that the data contains a symbol from which there are no transitions anywhere (even to itself), which would lead to a row in transition matrix full of zeros. In this case the build_mm() (as well as the EM algorithm) assumes that the the state is absorbing in a way that probability of staying in this state is 1.

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.
- `sequence_lengths`
  A vector of sequence lengths.
- `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, 15:86,
  alphabet = mvad_alphabet,
  states = mvad_scodes, labels = mvad_labels, xtstep = 6,
  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 = NULL,
  data = NULL,
  coefficients = NULL,
  cluster_names = NULL,
  ...
)
```

Arguments

observations	An stslist object (see TraMineR::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	Optional formula of class formula() for the mixture probabilities. Left side omitted.
data	A data frame containing the variables used in the formula. Ignored if no formula is provided.

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.
- sequence_lengths
  A vector of sequence lengths.
- 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, 15:86,
  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)
```

`cluster_names` *Get Cluster Names from Mixture HMMs*

Description

Get Cluster Names from Mixture HMMs

Usage

```
cluster_names(object)
```

Arguments

`object`	An object of class `mhmm` or `mnhmm`.

Value

A character vector containing the cluster names.

`cluster_names<-` *Set Cluster Names for Mixture Models*

Description

Set Cluster Names for Mixture Models

Usage

```
cluster_names(object) <- value
```

Arguments

`object`	An object of class `mhmm` or `mnhmm`.
`value`	A character vector containing the new cluster names.

Value

The modified object with updated cluster names.

coef.nhmm — *Get the Estimated Regression Coefficients of Non-Homogeneous Hidden Markov Models*

Description

Get the Estimated Regression Coefficients of Non-Homogeneous Hidden Markov Models

Usage

```
## S3 method for class 'nhmm'
coef(object, probs = NULL, ...)

## S3 method for class 'mnhmm'
coef(object, probs = NULL, ...)
```

Arguments

object	An object of class nhmm or mnhmm.
probs	Vector defining the quantiles of interest. When NULL (default), no quantiles are computed. The quantiles are based on bootstrap samples of coefficients, stored in object$boot.
...	Ignored.

Value

A list of data tables with the estimated coefficients for initial, transition, emission (separate data.table for each response), and cluster probabilities (in case of mixture model).

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 https://medialab.github.io/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]]
```

---

data_to_stslist *Transform TraMineR's state sequence object to data.table and vice versa*

Description

Transform TraMineR's state sequence object to data.table and vice versa

Usage

```
data_to_stslist(x, id, time, responses, seqdef_args = NULL, ...)

stslist_to_data(x, id = "id", time = "time", responses, ...)
```

Arguments

x	For data_to_stslist, a data.frame type of object in long format, or a model object of class nhmm or mnhmm. For stslist_to_data, an object of class stslist or list of such objects.
id	A character string specifying the id variable. Ignored if x is NHMM.
time	A character string specifying the time variable. Ignored if x is NHMM.
responses	A character vector specifying the name(s) of the response variable(s). Ignored if x is NHMM.
seqdef_args	A list of additional arguments to TraMineR::seqdef() in case of data_to_stslist. In case of length(responses) > 1, a list of lists. Ignored in stslist_to_data.
...	Ignored

Examples

```
# Convert multichannel stslist to long format data.table
biofam_seqs <- mhmm_biofam$observations # list of three stslist objects
d <- stslist_to_data(
  biofam_seqs, id = "Individual", time = "Age",
  responses = c("Marriage", "Children", "Residence")
)
head(d)

# Convert back to stslist
biofam_seqs2 <- data_to_stslist(
  d, id = "Individual", time = "Age",
  responses = c("Marriage", "Children", "Residence")
)
all.equal(biofam_seqs, biofam_seqs2, check.attributes = FALSE)
```

`estimate_mnhmm` *Estimate a Mixture Non-homogeneous Hidden Markov Model*

Description

Function `estimate_mnhmm` estimates a mixture version of non-homogeneous hidden Markov model (MNHMM) where initial, transition, emission, and mixture probabilities can depend on covariates. See `estimate_nhmm()` for further details.

Usage

```
estimate_mnhmm(
  n_states,
  n_clusters,
  emission_formula,
  initial_formula = ~1,
  transition_formula = ~1,
  cluster_formula = ~1,
  data,
  time,
  id,
  lambda = 0,
  prior_obs = "fixed",
  state_names = NULL,
  cluster_names = NULL,
  inits = "random",
  init_sd = NULL,
  restarts = 0L,
  method = "EM-DNM",
  bound = Inf,
  control_restart = list(),
  control_mstep = list(),
```

```
  check_rank = NULL,
  ...
)
```

Arguments

n_states	An integer > 1 defining the number of hidden states.
n_clusters	A positive integer defining the number of clusters (mixtures).
emission_formula	of class formula() for the state emission probabilities, or a list of such formulas in case of multiple response variables. The left-hand side of formulas define the responses. For multiple responses having same formula, you can use a form c(y1, y2) ~ x, where y1 and y2 are the response variables.
initial_formula	of class formula() for the initial state probabilities. Left-hand side of the formula should be empty.
transition_formula	of class formula() for the state transition probabilities. Left-hand side of the formula should be empty.
cluster_formula	of class formula() for the mixture probabilities.
data	A data frame containing the variables used in the model formulas.
time	Name of the time index variable in data.
id	Name of the id variable in data identifying different sequences.
lambda	Penalization factor lambda for penalized log-likelihood, where the penalization is 0.5 * lambda * sum(eta^2). Note that with method = "L-BFGS" both objective function (log-likelihood) and the penalization term is scaled with number of non-missing observations. Default is 0, but small values such as 1e-4 can help to ensure numerical stability of L-BFGS by avoiding extreme probabilities. See also argument bound for hard constraints.
prior_obs	Either "fixed" or a list of vectors given the prior distributions for the responses at time "zero". See details.
state_names	A vector of optional labels for the hidden states. If this is NULL (the default), numbered states are used.
cluster_names	A vector of optional labels for the clusters. If this is NULL (the default), numbered clusters are used.
inits	If inits = "random" (default), random initial values are used. Otherwise inits should be list of initial values. If coefficients are given using list elements eta_pi, eta_A, eta_B, and eta_omega, these are used as is, alternatively initial values can be given in terms of the initial state, transition, emission, and mixture probabilities using list elements initial_probs, emission_probs, transition_probs, and cluster_probs. These can also be mixed, i.e. you can give only initial_probs and eta_A.
init_sd	Standard deviation of the normal distribution used to generate random initial values. If initial values are given and restart = 0, then by default init_sd is set to zero. Otherwise the default is 2.

restarts	Number of times to run optimization using random starting values (in addition to the final run). Default is 0.
method	Optimization method used. Option "EM" uses EM algorithm with L-BFGS in the M-step. Option "DNM" uses direct maximization of the log-likelihood, by default using L-BFGS. Option "EM-DNM" (the default) runs first a maximum of 10 iterations of EM and then switches to L-BFGS (but other algorithms of NLopt can be used).
bound	Positive value defining the hard lower and upper bounds for the working parameters η, which are used to avoid extreme probabilities and corresponding numerical issues especially in the M-step of EM algorithm. Default is Inf, i.e., no bounds. Note that the bounds are not enforced for M-step in intercept-only case with lambda = 0.
control_restart	Controls for restart steps, see details.
control_mstep	Controls for M-step of EM algorithm, see details.
check_rank	If TRUE, the rank of the design matrices are checked for identifiability issues. Default is NULL, in which case checks are performed only if the number of sequences is 1000 or less, as the QR decomposition quickly becomes computationally demanding. If check is not performed, a warning is given, which can be circumvented by explicitly using check_rank = FALSE.
...	Additional arguments to nloptr::nloptr() and EM algorithm. See details.

Value

Object of class mnhmm.

See Also

estimate_nhmm() for further details.

Examples

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

d <- reshape(mvad, direction = "long", varying = list(15:86),
  v.names = "activity")

## Not run:
set.seed(1)
fit <- estimate_mnhmm(n_states = 3, n_clusters = 2,
  data = d, time = "time", id = "id",
  cluster_formula = ~ male + catholic + gcse5eq + Grammar +
    funemp + fmpr + livboth +
  N.Eastern + Southern + S.Eastern + Western,
  emission_formula = activity ~ male + catholic + gcse5eq,
  initial_formula = ~ 1,
  transition_formula = ~ male + gcse5eq
  )

## End(Not run)
```

estimate_nhmm *Estimate a Non-homogeneous Hidden Markov Model*

Description

Function estimate_nhmm estimates a non-homogeneous hidden Markov model (NHMM) where initial, transition, and emission probabilities can depend on covariates. Transition and emission probabilities can also depend on past responses, in which case the model is called feedback-augmented NHMM (FAN-HMM) (Helske, 2025).

Usage

```
estimate_nhmm(
  n_states,
  emission_formula,
  initial_formula = ~1,
  transition_formula = ~1,
  data,
  time,
  id,
  lambda = 0,
  prior_obs = "fixed",
  state_names = NULL,
  inits = "random",
  init_sd = NULL,
  restarts = 0L,
  method = "EM-DNM",
  bound = Inf,
  control_restart = list(),
  control_mstep = list(),
  check_rank = NULL,
  ...
)
```

Arguments

n_states	An integer > 1 defining the number of hidden states.
emission_formula	of class formula() for the state emission probabilities, or a list of such formulas in case of multiple response variables. The left-hand side of formulas define the responses. For multiple responses having same formula, you can use a form c(y1, y2) ~ x, where y1 and y2 are the response variables.
initial_formula	of class formula() for the initial state probabilities. Left-hand side of the formula should be empty.

transition_formula
: of class formula() for the state transition probabilities. Left-hand side of the formula should be empty.

data
: A data frame containing the variables used in the model formulas.

time
: Name of the time index variable in data.

id
: Name of the id variable in data identifying different sequences.

lambda
: Penalization factor lambda for penalized log-likelihood, where the penalization is 0.5 * lambda * sum(eta^2). Note that with method = "L-BFGS" both objective function (log-likelihood) and the penalization term is scaled with number of non-missing observations. Default is 0, but small values such as 1e-4 can help to ensure numerical stability of L-BFGS by avoiding extreme probabilities. See also argument bound for hard constraints.

prior_obs
: Either "fixed" or a list of vectors given the prior distributions for the responses at time "zero". See details.

state_names
: A vector of optional labels for the hidden states. If this is NULL (the default), numbered states are used.

inits
: If inits = "random" (default), random initial values are used. Otherwise inits should be list of initial values. If coefficients are given using list elements eta_pi, eta_A, eta_B, these are used as is, alternatively initial values can be given in terms of the initial state, transition, and emission probabilities using list elements initial_probs, emission_probs, and transition_probs. These can also be mixed, i.e. you can give only initial_probs and eta_A.

init_sd
: Standard deviation of the normal distribution used to generate random initial values. If initial values are given and restart = 0, then by default init_sd is set to zero. Otherwise the default is 2.

restarts
: Number of times to run optimization using random starting values (in addition to the final run). Default is 0.

method
: Optimization method used. Option "EM" uses EM algorithm with L-BFGS in the M-step. Option "DNM" uses direct maximization of the log-likelihood, by default using L-BFGS. Option "EM-DNM" (the default) runs first a maximum of 10 iterations of EM and then switches to L-BFGS (but other algorithms of NLopt can be used).

bound
: Positive value defining the hard lower and upper bounds for the working parameters η, which are used to avoid extreme probabilities and corresponding numerical issues especially in the M-step of EM algorithm. Default is Inf, i.e., no bounds. Note that the bounds are not enforced for M-step in intercept-only case with lambda = 0.

control_restart
: Controls for restart steps, see details.

control_mstep
: Controls for M-step of EM algorithm, see details.

check_rank
: If TRUE, the rank of the design matrices are checked for identifiability issues. Default is NULL, in which case checks are performed only if the number of sequences is 1000 or less, as the QR decomposition quickly becomes computationally demanding. If check is not performed, a warning is given, which can be circumvented by explicitly using check_rank = FALSE.

...
: Additional arguments to nloptr::nloptr() and EM algorithm. See details.

Details

In case of FAN-HMM with autoregressive dependency on the observational level, (i.e. response y_t depend on y_{t-1}), the emission probabilities at the first time point need special attention. By default, the model is initialized with fixed values for the first time point (`prior_obs = "fixed"`), meaning that if the input data consists of time points $t = 1, 2, \ldots$, then the model is defined from $t = 2$ onward and the data on $t = 1$ is used only for defining the emission probabilities at $t = 2$. Note that in this case also the initial state probabilities correspond to $t = 2$.

Alternatively, you can define `prior_obs` as a list of vectors, where the number of vectors is equal to the number of responses, and each vector gives the prior distribution for the response at $t = 0$. For example, if you have response variables `y` and `x`, where `y` has 3 categories and `x` 2 categories, you can define `prior_obs = list(y = c(0.5, 0.3, 0.2), x = c(0.7, 0.3))`. These distributions are then used to marginalize out y_0 and x_0 in the relevant emission probabilities.

By default, the model parameters are estimated using EM-DNM algorithm which first runs some iterations (100 by default) of EM algorithm, and then switches to L-BFGS. Other options include any numerical optimization algorithm of `nloptr::nloptr()`, or plain EM algorithm where the M-step uses L-BFGS (provided by the NLopt library).

With multiple runs of optimization (by using the `restarts` argument), it is possible to parallelize these runs using the `future` package, e.g., by calling `future::plan(multisession, workers = 2)` before `estimate_nhmm()`. See `future::plan()` for details. This is compatible with `progressr` package, so you can use `progressr::with_progress()` to track the progress of these multiple runs. Using argument `save_all_solutions = TRUE` allows to save the optimization results of all restarts, which is mostly useful for debugging purposes or if you are interested in seeing how much the parameter estimates vary due to variation in initial values.

During the estimation, the log-likelihood is scaled by the number of non-missing observations (`nobs(model)`), and the the covariate data is standardardized before optimization.

By default, the convergence is claimed when the relative change of the objective function is less than `1e-12`, the absolute change is less than `1e-8` or the relative or absolute change of the working parameters eta is less than `1e-6`. These can be changed by passing arguments `ftol_rel`, `ftol_abs`, `xtol_rel`, and `xtol_abs` via `...`. These, as well as, `maxeval` (maximum number of iterations, 1e5 by default), and `print_level` (default is `0`, no console output, larger values are more verbose), are used by the chosen main optimization method. The number of initial EM iterations in `EM-DNM` can be set using argument `maxeval_em_dnm` (default is 100), and algorithm for direct numerical optimization can be defined using argument `algorithm` (see `nloptr::nloptr()` for possible options).

For controlling these stopping criteria for the multistart phase, argument `control_restart` takes a list such as `list(ftol_rel = 0.01, print_level = 1)`. Default are as in the case of main optimization (which is always run once after the restarts, using best solution from restarts as initial value) Additionally, same options can be defined separately for the M-step of EM algorithm via list `control_mstep`. For `control_mstep`, the default values are `ftol_rel = 1e-10`, and `maxeval = 1000`, and otherwise identical to previous defaults above.

By default, EM algorithm uses SQUAREM acceleration (Varadhan and Roland, 2008) to improve the convergence speed near optimum. This can be turned of by using control argument `use_squarem = FALSE`.

Value

Object of class `nhmm` or `fanhmm`.

References

Helske J (2025). Feedback-augmented Non-homogeneous Hidden Markov Models for Longitudinal Causal Inference. arXiv preprint. doi:10.48550/arXiv.2503.16014.

Johnson SG. The NLopt nonlinear-optimization package, http://github.com/stevengj/nlopt.

Varadhan R and Roland C (2008). Simple and Globally Convergent Methods for Accelerating the Convergence of Any EM Algorithm. Scandinavian Journal of Statistics, 35: 335-353. doi:10.1111/j.1467-9469.2007.00585.x

Examples

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

d <- reshape(mvad, direction = "long", varying = list(15:86),
  v.names = "activity")

## Not run:
set.seed(1)
fit <- estimate_nhmm(n_states = 3,
  data = d, time = "time", id = "id",
  emission_formula = activity ~ gcse5eq, initial_formula = ~ 1,
  transition_formula = ~ male + gcse5eq,
  method = "DNM", maxeval = 2 # very small number of iterations for CRAN
  )

## End(Not run)
```

---

fanhmm_leaves — *A feedback-augmented non-homogeneous hidden Markov Model for leaves data*

Description

A FAN-HMM fitted for the`leaes` data.

Format

A model of class `fanhmm` with three hidden states

Details

The model is loaded by calling `data(fanhmm_leaves)`. The code used to estimate the model is available on Github in `data-raw` folder.

Examples

```
data("fanhmm_leaves")

fanhmm_leaves

get_marginals(fanhmm_leaves)
```

`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 maximum likelihood. Initial values for estimation are taken from the corresponding elements 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,
  constraints = NULL,
  fixed_inits = NULL,
  fixed_emissions = NULL,
  fixed_transitions = NULL,
  ...
)
```

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::nloptr()` (possibly after the EM step). The default is `FALSE`.

`local_step` Logical. Whether or not to use local optimization via `nloptr::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. Note that iteration counter starts with -1 so with `maxeval = 1` you get already two iterations. This is for backward compatibility reasons.
- `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 elements:
  - `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 `stats::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::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::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 `c(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` (but was `TRUE` in 2.0.0 version of the package).

`constraints` Integer vector defining equality constraints for emission distributions. Not supported for EM algorithm. See details.

`fixed_inits` Can be used to fix some of the probabilities to their initial values.Should have same structure as `model$initial_probs`, where each element is either `TRUE` (fixed) or `FALSE` (to be estimated). Note that zero probabilities are always fixed to 0. Not supported for EM algorithm. See details.

`fixed_emissions`
Can be used to fix some of the probabilities to their initial values. Should have same structure as `model$emission_probs`, where each element is either `TRUE` (fixed) or `FALSE` (to be estimated). Note that zero probabilities are always fixed to 0. Not supported for EM algorithm. See details.

`fixed_transitions`
Can be used to fix some of the probabilities to their initial values. Should have same structure as `model$transition_probs`, where each element is either `TRUE` (fixed) or `FALSE` (to be estimated). Note that zero probabilities are always fixed to 0. Not supported for EM algorithm. See details.

`...` Additional arguments to `nloptr::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 should 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.

Equality constraints for emission distributions can be defined using the argument `constraints`. This should be a vector with length equal to the number of states, with numbers starting from 1 and increasing for each unique row of the emission probability matrix. For example in case of five states with emissions of first and third states being equal, `constraints = c(1, 2, 1, 3, 4)`. Similarly, some of the model parameters can be fixed to their initial values by using arguments `fixed_inits`, `fixed_emissions`, and `fixed_transitions`, where the structure of the arguments should be same as the corresponding model components, so that TRUE value means that the parameter should be fixed and FALSE otherwise (it is still treated as fixed if it is zero though). For both types of constrains, only numerical optimisation (local or global) is available, and currently the gradients are computed numerically (if needed) in these cases.

In a case where the is no transitions from one state to anywhere (even to itself), the state is defined as absorbing in a way that probability of staying in this state is fixed to 1. See also `build_mm` function.

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.

References

Helske S. and Helske J. (2019). Mixture Hidden Markov Models for Sequence Data: The seqHMM Package in R, Journal of Statistical Software, 88(3), 1-32. doi:10.18637/jss.v088.i03

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; and `plot.hmm()` and `plot.mhmm()` for plotting model objects.

Examples

```
# Hidden Markov model for mvad 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, 15:86,
  alphabet = mvad_alphabet,
  states = mvad_scodes, labels = mvad_labels, xtstep = 6,
  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 unless there are missing observations

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$married,
  start = 15,
  alphabet = c("single", "married", "divorced"),
  cpal = c("violetred2", "darkgoldenrod2", "darkmagenta")
)
child_seq <- seqdef(biofam3c$children,
  start = 15,
  alphabet = c("childless", "children"),
  cpal = c("darkseagreen1", "coral3")
)
left_seq <- seqdef(biofam3c$left,
  start = 15,
  alphabet = c("with parents", "left home"),
  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"),
```

```
  cpal = c("violetred2", "darkgoldenrod2", "darkmagenta")
)
child_seq <- seqdef(biofam3c$children,
  start = 15,
  alphabet = c("childless", "children"),
  cpal = c("darkseagreen1", "coral3")
)
left_seq <- seqdef(biofam3c$left,
  start = 15,
  alphabet = c("with parents", "left home"),
  cpal = c("lightblue", "red3")
)

## 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)

# Left-to-right HMM with equality constraint:

set.seed(1)

# Transition matrix
# Either stay or move to next state
A <- diag(c(0.9, 0.95, 0.95, 1))
A[1, 2] <- 0.1
A[2, 3] <- 0.05
A[3, 4] <- 0.05

# Emission matrix, rows 1 and 3 equal
B <- rbind(
  c(0.4, 0.2, 0.3, 0.1),
  c(0.1, 0.5, 0.1, 0.3),
  c(0.4, 0.2, 0.3, 0.1),
  c(0, 0.2, 0.4, 0.4)
)

# Start from first state
init <- c(1, 0, 0, 0)

# Simulate sequences
sim <- simulate_hmm(
  n_sequences = 100,
  sequence_length = 20, init, A, B
)

# initial model, use true values as inits for faster estimation here
model <- build_hmm(sim$observations, init = init, trans = A, emiss = B)
```

```
# estimate the model subject to constraints:
# First and third row of emission matrix are equal (see details)
fit <- fit_model(model,
  constraints = c(1, 2, 1, 3),
  em_step = FALSE, local_step = TRUE
)
fit$model

## Fix some emissions:

fixB <- matrix(FALSE, 4, 4)
fixB[2, 1] <- fixB[1, 3] <- TRUE # these are fixed to their initial values
fit <- fit_model(model,
  fixed_emissions = fixB,
  em_step = FALSE, local_step = TRUE
)
fit$model$emission_probs
```

---

forward_backward *Forward and Backward Probabilities for Hidden Markov Model*

Description

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

Usage

```
forward_backward(model, ...)

## S3 method for class 'hmm'
forward_backward(model, forward_only = FALSE, ...)

## S3 method for class 'mhmm'
forward_backward(model, forward_only = FALSE, ...)

## S3 method for class 'nhmm'
forward_backward(model, forward_only = FALSE, ...)

## S3 method for class 'mnhmm'
forward_backward(model, forward_only = FALSE, ...)
```

Arguments

model	A hidden Markov model.
...	Ignored.
forward_only	If TRUE, only forward probabilities are computed. The default is FALSE.

Value

A `data.frame` with log-values of forward and backward probabilities.

Examples

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

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

head(fb)
```

`get_cluster_probs` *Extract the Prior Cluster Probabilities of MHMM or MNHMM*

Description

Prior cluster probability refers to the probability of sequence i belonging to cluster k without conditioning on the observed sequence, i.e., $P(i \in M_k|X_i)$, where M_k is cluster k and X_i are the covariates for individual i. The posterior cluster probability refers to the probability of sequence i belonging to cluster k given the observed sequence, i.e., $P(i \in M_k|y_i, X_i)$, where y_i is the observed sequence for individual i.

Usage

```
get_cluster_probs(model, ...)

## S3 method for class 'mnhmm'
get_cluster_probs(model, type = "prior", ids = NULL, ...)

## S3 method for class 'mhmm'
get_cluster_probs(model, type = "prior", ...)
```

Arguments

`model`	A hidden Markov model.
`...`	Ignored.
`type`	Character string indicating the type of cluster probabilities. Either `"prior"` (the default), or `"posterior"`. See details.
`ids`	Optional vector of sequence IDs. If `NULL` (the default), probabilities are computed for all sequences in the data, unless the probabilities are equal for all sequences, i.e., covariates do not depend on ID.

See Also

`posterior_cluster_probabilities()`.

get_emission_probs *Extract the Emission Probabilities of Hidden Markov Model*

Description

Extract the Emission Probabilities of Hidden Markov Model

Usage

```
get_emission_probs(model, ...)

## S3 method for class 'nhmm'
get_emission_probs(model, ids = NULL, ...)

## S3 method for class 'mnhmm'
get_emission_probs(model, ids = NULL, ...)

## S3 method for class 'hmm'
get_emission_probs(model, ...)

## S3 method for class 'mhmm'
get_emission_probs(model, ...)
```

Arguments

model	A hidden Markov model.
...	Ignored.
ids	Optional vector of sequence IDs. If NULL (the default), probabilities are computed for all sequences in the data, unless the probabilities are equal for all sequences, i.e., covariates do not depend on ID.

get_initial_probs *Extract the Initial State Probabilities of Hidden Markov Model*

Description

Extract the Initial State Probabilities of Hidden Markov Model

Usage

```
get_initial_probs(model, ...)

## S3 method for class 'nhmm'
get_initial_probs(model, ids = NULL, ...)
```

```
## S3 method for class 'mnhmm'
get_initial_probs(model, ids = NULL, ...)

## S3 method for class 'hmm'
get_initial_probs(model, ...)

## S3 method for class 'mhmm'
get_initial_probs(model, ...)
```

Arguments

model	A hidden Markov model.
...	Ignored.
ids	Optional vector of sequence IDs. If NULL (the default), probabilities are computed for all sequences in the data, unless the probabilities are equal for all sequences, i.e., covariates do not depend on ID.

Value

A data.table containing the initial state probabilities for each sequence defined by ids.

---

get_marginals *Compute the Marginal Probabilities from NHMMs*

Description

get_marginals returns the marginal state, response, transition, and emission probabilities, optionally per grouping defined by condition. By default, the marginalization weights sequences by the corresponding posterior probabilities of the latent states, i.e., conditional probabilities of the latent states given all data (weighting = "posterior"). If weighting = "forward", marginalization is based on forward probabilities, i.e. state probabilities given data up to that point which allows you to compute, for example, state marginals of form $P(state_t|data_1, \ldots, data_t)$ (whereas in posterior probability weighting the conditioning is on $data_1, \ldots, data_T$. If weighting = "none", all individuals and time points are treated equally, without accounting for the probability that individual is at particular state at particular time.

Usage

```
get_marginals(
  model,
  probs = NULL,
  condition = NULL,
  newdata = NULL,
  type = c("state", "response", "transition", "emission"),
  weighting = c("posterior", "forward", "none")
)
```

Arguments

model	An object of class nhmm or mnhmm.
probs	Vector defining the quantiles of interest. Default is NULL, in which case no quantiles are computed. The quantiles are based on bootstrap samples of coefficients, stored in object$boot.
condition	An optional vector of variable names used for conditional marginal probabilities. Default is NULL, in which case marginalization is done over all variables, so that for example marginal emission probabilities are computed over all individuals and time points.
newdata	An optional data frame containing the new data to be used in computing the probabilities.
type	A character vector defining the marginal probabilities of interest. Can be one or multiple of "state", "response", "transition", and "emission". Default is to compute all of these.
weighting	A character string defining the type of weighting used in marginalization. One of "posterior" , "forward", "none". See details.

---

get_transition_probs *Extract the State Transition Probabilities of Hidden Markov Model*

Description

Extract the State Transition Probabilities of Hidden Markov Model

Usage

```
get_transition_probs(model, ...)

## S3 method for class 'nhmm'
get_transition_probs(model, ids = NULL, ...)

## S3 method for class 'mnhmm'
get_transition_probs(model, ids = NULL, ...)

## S3 method for class 'hmm'
get_transition_probs(model, ...)

## S3 method for class 'mhmm'
get_transition_probs(model, ...)
```

Arguments

model	A hidden Markov model.
...	Ignored.

ids	Optional vector of sequence IDs. If NULL (the default), probabilities are computed for all sequences in the data, unless the probabilities are equal for all sequences, i.e., covariates do not depend on ID.

---

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.

`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, ...)

## S3 method for class 'hmm'
hidden_paths(model, as_stslist = FALSE, ...)

## S3 method for class 'mhmm'
hidden_paths(model, as_stslist = FALSE, ...)

## S3 method for class 'nhmm'
hidden_paths(model, as_stslist = FALSE, ...)

## S3 method for class 'mnhmm'
hidden_paths(model, as_stslist = FALSE, ...)
```

Arguments

model	A hidden Markov model.
...	Ignored.
as_stslist	Logical. If TRUE, the output the is converted to an stslist object. Default is FALSE, which returns a data.table.

Value

The most probable paths of hidden states as an data.table. The log-probability is included as an attribute log_prop.

See Also

hmm_biofam for information on the model used in the example; and ggseqplot::ggseqiplot() and stacked_sequence_plot() 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)
head(mpp)
# Plot hidden paths for the first 100 individuals
seqs <- data_to_stslist(mpp, "id", "time", "state")
stacked_sequence_plot(seqs, type = "i", ids = 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 TraMineR::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$married,
  start = 15,
  alphabet = c("single", "married", "divorced"),
  cpal = c("violetred2", "darkgoldenrod2", "darkmagenta")
)
child_seq <- seqdef(biofam3c$children,
  start = 15,
  alphabet = c("childless", "children"),
  cpal = c("darkseagreen1", "coral3")
)
left_seq <- seqdef(biofam3c$left,
  start = 15,
  alphabet = c("with parents", "left home"),
  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 `TraMineR::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 `TraMineR::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, 15:86, alphabet = mvad_alphabet,
  states = mvad_scodes, labels = mvad_labels, xtstep = 6,
  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 `TraMineR::mvad()` for more information on the data.

Examples

```
data("hmm_mvad")

# Plotting the model
plot(hmm_mvad)
```

`leaves` *Synthetic data on fathers' parental leaves in Finland*

Description

Synthetic data on fathers' parental leaves in Finland

Format

A `data.table` with 9281 rows and 9 variables

Details

The `leaves` data is a synthetic version of the Finnish fathers' leave-taking data used in Helske et al. (2024) and Helske (2025). The data consists of variables

- workplace: Workplace ID.
- father: father ID within workplace. More accurately, this is the birth of a child, i.e. same father can have multiple entries in data, but each entry has separate ID.
- year: Year when the child was born.
- leave: Factor of leave-taking of the father.
- Occupation: Factor of skill level of the father's occupation
- reform2013: Factor indicating whether the father was eligible for the leave under the 2013 reform.
- same_occupation: Logical value, TRUE if father had same occupation as the previous father.
- lag_reform2013: Factor indicating whether the previous father was eligible for the reform.
- lag_occupation: Factor indiciting the occupation of previous father.

References

Helske S, Helske J, Chapman SN, Kotimäki S, Salin M, and Tikka S (2024). Heterogeneous workplace peer effects in fathers' parental leave uptake in Finland. doi: 10.31235/osf.io/p3chf

Helske J (2025). Feedback-augmented Non-homogeneous Hidden Markov Models for Longitudinal Causal Inference. ArXiv preprint. doi:10.48550/arXiv.2503.16014

Examples

```
data("leaves")
head(leaves)
# convert to stslist
leaves_sequences <- data_to_stslist(
  leaves, id = "workplace", time = "father", responses = "leave",
  seqdef_args = list(cpal = c("tomato", "navyblue", "goldenrod"))
)
stacked_sequence_plot(leaves_sequences)
```

logLik.hmm *Log-likelihood of a Hidden Markov Model*

Description

Log-likelihood of a Hidden Markov Model

Usage

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

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

Arguments

object	A hidden Markov model.
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 FALSE.
...	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.

logLik.nhmm *Log-likelihood of a Non-homogeneous Hidden Markov Model*

Description

Log-likelihood of a Non-homogeneous Hidden Markov Model

Usage

```
## S3 method for class 'nhmm'
logLik(object, partials = FALSE, ...)

## S3 method for class 'mnhmm'
logLik(object, partials = FALSE, ...)
```

Arguments

object	A hidden Markov model.
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.
...	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.

---

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, cpal)
```

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.
cpal	The color palette used for the new combined symbols. Optional in a case where the number of symbols is less or equal to 200 (in which case the seqHMM::colorpalette is used).

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, cpal)
```

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.
cpal	The color palette used for the new combined symbols. Optional in a case where the number of symbols is less or equal to 200 (in which case the seqHMM::colorpalette is used).

See Also

mc_to_sc() for transforming multichannel hmm or mhmm objects into single-channel representations; stacked_sequence_plot 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$married,
  start = 15,
  alphabet = c("single", "married", "divorced"),
  cpal = c("violetred2", "darkgoldenrod2", "darkmagenta")
)
child_seq <- seqdef(biofam3c$children,
  start = 15,
  alphabet = c("childless", "children"),
  cpal = c("darkseagreen1", "coral3")
)
left_seq <- seqdef(biofam3c$left,
  start = 15,
  alphabet = c("with parents", "left home"),
  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
stacked_sequence_plot(
  list(
    "Marriage" = marr_seq, "Parenthood" = child_seq,
    "Residence" = left_seq, "Combined" = sc_data
  ),
  type = "i",
  ids = 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 TraMineR::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$married,
  start = 15,
  alphabet = c("single", "married", "divorced"),
  cpal = c("violetred2", "darkgoldenrod2", "darkmagenta")
)
child_seq <- seqdef(biofam3c$children,
  start = 15,
  alphabet = c("childless", "children"),
  cpal = c("darkseagreen1", "coral3")
)
left_seq <- seqdef(biofam3c$left,
  start = 15,
  alphabet = c("with parents", "left home"),
  cpal = c("lightblue", "red3")
)

## 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 `TraMineR::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 `TraMineR::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, 15:86, alphabet = mvad_alphabet,
  states = mvad_scodes, labels = mvad_labels, xtstep = 6,
  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 `TraMineR::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)
}
```

---

most_probable_cluster *Extract Most Probable Cluster for Each Sequence*

Description

Extract Most Probable Cluster for Each Sequence

Usage

```
most_probable_cluster(x, type = "viterbi", hp = NULL)
```

Arguments

x	An object of class mhmm or mnhmm.
type	A character string specifying the method to use. Either "viterbi" (default) or "posterior". Former uses the most probable hidden path to determine the cluster membership for each sequence, while the latter finds the cluster which has the largest sum of posterior probabilities of states of that cluster.
hp	An output from `hidden_paths()` function. Only used in case of type = "viterbi". If missing, hidden paths will be computed using x.

Value

A vector containing the most probable cluster for each sequence.

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,
  respect_void = TRUE,
```

```
  ...
)
```

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 TraMineR::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 stats::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 TraMineR::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 TraMineR::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 positions 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 `TraMineR::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 `TraMineR::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.

respect_void
: If TRUE (default), states at the time points corresponding to TraMineR's void in the observed sequences are set to void in the hidden state sequences as well.

...
: Other arguments to be passed on to TraMineR::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.

---

nobs.hmm *Number of Observations in Hidden Markov Model*

Description

Extract the number of non-missing observations of HMM. 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.

Usage

```
## S3 method for class 'hmm'
nobs(object, ...)

## S3 method for class 'mhmm'
nobs(object, ...)

## S3 method for class 'nhmm'
nobs(object, ...)

## S3 method for class 'mnhmm'
nobs(object, ...)
```

Arguments

object	An object of class hmm, mhmm, nhmm, or mnhmm.
...	Ignored.

---

permute_states *Permute the states of NHMM using Hungarian algorithm*

Description

This function finds the permutation (according to Hungarian algorithm) of estimated model coefficients γ which best matches some reference values. The main purpose is to permute the bootstrap samples of model coefficients to match as closely as possible to the maximum likelihood estimates obtained from the original data.

Usage

```
permute_states(estimates, reference)
```

Arguments

estimates	A list γ coefficients as in model$gammas, where model is an nhmm object.
reference	Another list of γ coefficients for which to match the estimates.

Details

The cost matrix in Hungarian algorithm is based on the sum of L2 norms of the differences of estimated and reference values of π, A and B.

This function is mostly meant for internal usage within the bootstrap methods of seqHMM.

Value

Permuted version of estimates, with added attribute permutation which contains the permutations used to obtain for example the new gamma_pi as estimates$gamma_pi[perm, , drop = FALSE].

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",
  cpal.legend = "auto",
  legend.order = TRUE,
  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 igraph::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 igraph::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::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 TraMineR::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).

cpal.legend
: Optional color palette for the legend, only considered when legend.order is FALSE. Should match ltext.

legend.order
: Whether to use the default order in the legend, i.e., order by appearance (first by hidden state, then by emission probability). TRUE by default.

main
: Main title for the plot. Omitted by default.

withlegend
: Deprecated. Use with.legend instead.

...
: Other parameters passed on to igraph::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 igraph::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 the HMM
plot(hmm_mvad)

# Checking the order of observed states (needed for the next call)
require(TraMineR)
alphabet(hmm_mvad$observations)

# Plotting the HMM with own legend (note: observation "none" nonexistent in the observations)
plot(hmm_mvad,
  # Override the default order in the legend
  legend.order = FALSE,
  # Colours in the pies (ordered by the alphabet of observations)
  cpal = c("purple", "pink", "brown", "lightblue", "orange", "green"),
  # Colours in the legend (matching to ltext)
  cpal.legend = c("orange", "pink", "brown", "green", "lightblue", "purple", "gray"),
  # Labels in the legend (matching to cpal.legend)
 ltext = c("school", "further educ", "higher educ", "training", "jobless", "employed", "none")
)
```

```
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 utils::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 igraph::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 igraph::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::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 `TraMineR::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 `igraph::plot.igraph()` such as `vertex.color`, `vertex.label.cex`, or `edge.lty`.

References

Helske S. and Helske J. (2019). Mixture Hidden Markov Models for Sequence Data: The seqHMM Package in R, Journal of Statistical Software, 88(3), 1-32. doi:10.18637/jss.v088.i03

See Also

`build_mhmm()` and `fit_model()` for building and fitting mixture hidden Markov models; `igraph::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.

References

Helske S. and Helske J. (2019). Mixture Hidden Markov Models for Sequence Data: The seqHMM Package in R, Journal of Statistical Software, 88(3), 1-32. doi:10.18637/jss.v088.i03

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.

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(grDevices::rainbow(15))
```

posterior_cluster_probabilities *Extract Posterior Cluster Probabilities*

Description

Extract Posterior Cluster Probabilities

Usage

```
posterior_cluster_probabilities(x)
```

Arguments

`x`	An object of class `mhmm` or `mnhmm`.

Value

a data.frame of posterior cluster probabilities for each sequence and cluster.

---

posterior_probs *Posterior Probabilities for Hidden Markov Models*

Description

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

Usage

```
posterior_probs(model, ...)

## S3 method for class 'hmm'
posterior_probs(model, ...)

## S3 method for class 'mhmm'
posterior_probs(model, ...)

## S3 method for class 'nhmm'
posterior_probs(model, ...)

## S3 method for class 'mnhmm'
posterior_probs(model, ...)
```

Arguments

model	A hidden Markov model object.
...	Ignored.

Value

A data frame of posterior probabilities for each state and sequence.

Examples

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

# Compute posterior probabilities
pb <- posterior_probs(mhmm_biofam)
```

predict.nhmm *Predictions from Non-homogeneous Hidden Markov Models*

Description

This function computes the marginal forward predictions for NHMMs and MNHMMs, where the marginalization is (by default) over individuals and time points, weighted by the latent state probabilities.

Usage

```
## S3 method for class 'nhmm'
predict(
  object,
  newdata,
  newdata2 = NULL,
  condition = NULL,
  type = c("state", "response", "transition", "emission"),
  probs = c(0.025, 0.975),
  boot_idx = FALSE,
  ...
)

## S3 method for class 'mnhmm'
predict(
  object,
  newdata,
  newdata2 = NULL,
  condition = NULL,
  type = c("state", "response", "transition", "emission"),
  probs = c(0.025, 0.975),
  boot_idx = FALSE,
  ...
)
```

Arguments

object	An object of class nhmm or mnhmm.
newdata	A data frame used for computing the predictions.
newdata2	An optional data frame for predictions, in which case the estimates are differences between predictions using newdata and newdata2.
condition	An optional vector of variable names used for conditional predictions.
type	A character vector defining the marginal predictions of interest. Can be one or multiple of "state", "response", "transition", and "emission". Default is to compute all of these.

probs	A numeric vector of quantiles to compute.
boot_idx	Logical indicating whether to use bootstrap samples in marginalization when computing quantiles. Default is FALSE. Currently only used in case where condition is NULL and
...	Ignored.

---

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 'nhmm'
print(x, digits = 3, ...)

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

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

Arguments

x	Hidden Markov model.
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.

return_msg *Convert return code from estimate_nhmm and estimate_mnhmm to text*

Description

Convert return code from estimate_nhmm and estimate_mnhmm to text

Usage

```
return_msg(code)
```

Arguments

code	Integer return code from model$estimation_results$return_code.

Value

Code translated to informative message.

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 elements 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]])
```

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

Description

These functions still work but will be removed (defunct) in the next version of `seqHMM`.

Details

- `ssplot`, `ssp`, `mssplot`, `plot.ssp`. Use `stacked_sequence_plot()` instead.
- `gridplot` Use `stacked_sequence_plot()`, `ggseqplot`, and `patchwork` packages instead.

`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	The number of sequences to simulate.
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; `stacked_sequence_plot()` 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
)

stacked_sequence_plot(sim, sort_by = "mds", 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, alpha = 1)

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

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

Arguments

n_states	Number of states in each cluster.
n_clusters	Number of clusters.
alpha	A scalar, or a vector of length S (number of states) or M (number of symbols) defining the parameters of the Dirichlet distribution used to simulate the probabilities.
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.

---

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 = NULL,
  data = NULL,
  coefficients = NULL
)
```

Arguments

n_sequences	The number of sequences to simulate.
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.

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 <- 30
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 = 20, formula = ~ covariate_1 + covariate_2,
  data = dataf, coefficients = coefs
)

stacked_sequence_plot(sim,
  sort_by = "start", sort_channel = "states", 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, as_stslist = TRUE)

stacked_sequence_plot(
  list(
    "estimated paths" = paths,
    "true (simulated)" = sim$states
  ),
  sort_by = "start",
  sort_channel = "true (simulated)",
  type = "i"
)
```

`simulate_mnhmm` *Simulate Mixture Non-homogeneous Hidden Markov Models*

Description

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

Usage

```
simulate_mnhmm(
  n_states,
  n_clusters,
  emission_formula,
  initial_formula = ~1,
  transition_formula = ~1,
  cluster_formula = ~1,
  data,
  id,
  time,
  coefs = NULL,
  init_sd = 2 * is.null(coefs),
  check_rank = NULL
)
```

Arguments

`n_states`	An integer > 1 defining the number of hidden states.
`n_clusters`	The number of clusters/mixtures.
`emission_formula`	of class `formula()` for the state emission probabilities, or a list of such formulas in case of multiple response variables. The left-hand side of formulas define the responses. For multiple responses having same formula, you can use a form `c(y1, y2) ~ x`, where `y1` and `y2` are the response variables.
`initial_formula`	of class `formula()` for the initial state probabilities. Left-hand side of the formula should be empty.
`transition_formula`	of class `formula()` for the state transition probabilities. Left-hand side of the formula should be empty.
`cluster_formula`	of class `formula()` for the mixture probabilities.
`data`	A data frame containing the variables used in the model formulas. Note that this should also include also the response variable(s), which are used to define the number of observed symbols (using `levels()`) and the length of sequences. The actual values of the response variables does not matter though, as they are replaced by the simulated values. The exception is the first time point in FAN-HMM case: If the `emission_formula` contains lagged responses, the response variable values at the first time point are used to define the emissions at the second time point, and the simulations are done from the second time point onward. This matches the case `prior_obs = "fixed"` in `estimate_nhmm()`. Note that

	compared to `estimate_*` functions, unused factor levels are not automatically dropped from `data`.
`id`	Name of the id variable in `data` identifying different sequences.
`time`	Name of the time index variable in `data`.
`coefs`	Same as argument `inits` in `estimate_mnhmm()`. If `NULL`, (default), the model parameters are generated randomly. If you want to simulate new sequences based on an estimated model `fit`, you can use `coefs = fit$etas` and `init_sd = 0`.
`init_sd`	Standard deviation of the normal distribution used to generate random coefficients. Default is `2` when `coefs` is `NULL` and `0` otherwise.
`check_rank`	If `TRUE`, the rank of the design matrices are checked for identifiability issues. Default is `NULL`, in which case checks are performed only if the number of sequences is 1000 or less, as the QR decomposition quickly becomes computationally demanding. If check is not performed, a warning is given, which can be circumvented by explicitly using `check_rank = FALSE`.

Value

A list with elements

- `model`: A `nhmm` model object corresponding the simulations.
- `states`: A `data.table` containing the simulated hidden states.
- `data`: A `data.table` of original data filled with simulated observations.

`simulate_nhmm` *Simulate Non-homogeneous Hidden Markov Models*

Description

Simulate sequences of observed and hidden states given the parameters of a non-homogeneous hidden Markov model.

Usage

```
simulate_nhmm(
  n_states,
  emission_formula,
  initial_formula = ~1,
  transition_formula = ~1,
  data,
  id,
  time,
  coefs = NULL,
  init_sd = 2 * is.null(coefs),
  check_rank = NULL
)
```

Arguments

n_states	An integer > 1 defining the number of hidden states.
emission_formula	of class formula() for the state emission probabilities, or a list of such formulas in case of multiple response variables. The left-hand side of formulas define the responses. For multiple responses having same formula, you can use a form c(y1, y2) ~ x, where y1 and y2 are the response variables.
initial_formula	of class formula() for the initial state probabilities. Left-hand side of the formula should be empty.
transition_formula	of class formula() for the state transition probabilities. Left-hand side of the formula should be empty.
data	A data frame containing the variables used in the model formulas. Note that this should also include also the response variable(s), which are used to define the number of observed symbols (using levels()) and the length of sequences. The actual values of the response variables does not matter though, as they are replaced by the simulated values. The exception is the first time point in FAN-HMM case: If the emission_formula contains lagged responses, the response variable values at the first time point are used to define the emissions at the second time point, and the simulations are done from the second time point onward. This matches the case prior_obs = "fixed" in estimate_nhmm(). Note that compared to estimate_* functions, unused factor levels are not automatically dropped from data.
id	Name of the id variable in data identifying different sequences.
time	Name of the time index variable in data.
coefs	Same as argument inits in estimate_nhmm(). If NULL, (default), the model parameters are generated randomly. If you want to simulate new sequences based on an estimated model fit, you can use coefs = fit$etas and init_sd = 0.
init_sd	Standard deviation of the normal distribution used to generate random coefficients. Default is 2 when coefs is NULL and 0 otherwise.
check_rank	If TRUE, the rank of the design matrices are checked for identifiability issues. Default is NULL, in which case checks are performed only if the number of sequences is 1000 or less, as the QR decomposition quickly becomes computationally demanding. If check is not performed, a warning is given, which can be circumvented by explicitly using check_rank = FALSE.

Value

A list with elements

- model: A nhmm model object corresponding the simulations.
- states: A data.table containing the simulated hidden states.
- data: A data.table of original data filled with simulated observations.

sort_sequences *Sort sequences in a sequence object*

Description

Sort sequences in a sequence object

Usage

```
sort_sequences(x, sort_by = "start", sort_channel = 1, dist_method = "OM")
```

Arguments

x	An stslist object or a list of of such objects of same size, typically created with TraMineR::seqdef() or data_to_stslist().
sort_by	A character string specifying the sorting criterion. Options are "none" (no sorting), "start" (sort by the first state), "end" (sort by last state), and "mds" (sort by the multidimensional scaling).
sort_channel	An integer or character string specifying the channel to sort by (unless sort_by = "mds in which case all channels are used for defining the sorting).
dist_method	A character string specifying the distance method to use when sorting by the multidimensional scaling. Passed to TraMineR::seqdist(), or TraMineR::seqMD() in the multichannel case.

Value

Sorted version of the input, with additional attribute ordering indicating the indices used to sort the data, i.e., output = input[ordering,].

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,
  respect_void = TRUE,
  ...
)
```

Arguments

x	Either a hidden Markov model object of class hmm or a state sequence object of class stslist (created with the TraMineR::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 TraMineR::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 stats::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 TraMineR::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 TraMineR::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 positions 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 TraMineR::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 TraMineR::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.

respect_void
: If TRUE (default), states at the time points corresponding to TraMineR's void in the observed sequences are set to void in the hidden state sequences as well.

...
: Other arguments to be passed on to TraMineR::seqplot().

Details

This function is deprecated, use stacked_sequence_plot() instead.

Value

Object of class ssp.

---

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,
```

```
  respect_void = TRUE,
  ...
)
```

Arguments

x	Either a hidden Markov model object of class hmm or a state sequence object of class stslist (created with the TraMineR::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 TraMineR::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 stats::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 TraMineR::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 TraMineR::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 positions 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 `TraMineR::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 `TraMineR::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.

`respect_void`
: If `TRUE` (default), states at the time points corresponding to TraMineR's void in the observed sequences are set to void in the hidden state sequences as well.

`...`
: Other arguments to be passed on to `TraMineR::seqplot()`.

Details

This function is deprecated and will be removed in future versions of `seqHMM`.

---

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

Description

Function `stacked_sequence_plot` draws stacked sequence plots of sequence object created with the TraMineR::seqdef function or observations and/or most probable paths of model objects of `seqHMM` (e.g., `hmm` and `mhmm`).

Usage

```
stacked_sequence_plot(
  x,
  plots = "observations",
  type = "distribution",
  ids,
  sort_by = "none",
  sort_channel,
  dist_method = "OM",
  group = NULL,
  legend_position = "right",
  ...
)
```

Arguments

x	Either a hidden Markov model object of class hmm, mhmm, nhmm, or mnhmm, a sequence object of class stslist (created with the TraMineR::seqdef() function) or a list of stslist objects.
plots	What to plot. One of "observations" for observations (the default), "hidden states" (or "hidden_paths", for the backward compatibility) for most probable paths of hidden states, or "both" for observations and hidden paths together. Latter two options are only possible for cases whe x is a hidden Markov model.
type	The type of the plot. Available types are "index" for sequence index plots and "distribution" for state distribution plots (the default). See ggseqplot::ggseqiplot() and ggseqplot::ggseqdplot() for details.
ids	Indexes of the subjects to be plotted (the default is all). For example, 'ids = c(1:10, 15) plots the first ten subjects and subject 15 in the data.
sort_by	A sorting variable or a sort method (one of "none, "start", "end", or "mds" for type = "index". Option "mds" arranges the sequences according to the scores of multidimensional scaling (using stats::cmdscale()). Default is "none", i.e., no sorting. Numeric vectors are passed to sortv argument of ggseqplot::ggseqiplot().
sort_channel	Name of the channel which should be used for the sorting. Alternatively value "Hidden states" uses the hidden state sequences for sorting. Default is to sort by the first channel in the data. If sort_by = "mds", all channels are used for defining the sorting.
dist_method	The metric to be used for computing the distances of the sequences if multidimensional scaling is used for sorting. Default is "OM" (optimal matching). For other more information and other options, see TraMineR::seqdist(). Transition rates are used for defining substitution costs if needed. Note that not all methods are available for sorting multichannel data.
group	Variable used for grouping the sequences in each channel, which is passed to ggseqplot::ggseqiplot() and ggseqplot::ggseqdplot(). By default, no grouping is done, except for mixture models where the grouping is based on most probable clusters (defined by the most probable hidden paths). Grouping by clusters can be overloaded by supplying variable for group or by setting group = NA.
legend_position	Position of legend for each channel, passed to legend.position argument of ggplot2::theme(). Either a vector of length 1, or of length matching the number of channels to be plotted.
...	Other arguments to ggseqplot::ggseqiplot() or ggseqplot::ggseqdplot().

Examples

```
p <- stacked_sequence_plot(
  mhmm_biofam,
  plots = "both",
  type = "d",
  legend_position = c("right", "right", "right", "none")
```

```
)
library("ggplot2")
p & theme(plot.margin = unit(c(1, 1, 0, 2), "mm"))
```

`state_names` *Get State Names of Hidden Markov Model*

Description

Get State Names of Hidden Markov Model

Set State Names of Hidden Markov Model

Usage

```
state_names(object)

## S3 method for class 'hmm'
state_names(object)

## S3 method for class 'mhmm'
state_names(object)

## S3 method for class 'nhmm'
state_names(object)

## S3 method for class 'mnhmm'
state_names(object)

state_names(object) <- value

## S3 replacement method for class 'hmm'
state_names(object) <- value

## S3 replacement method for class 'mhmm'
state_names(object) <- value

## S3 replacement method for class 'nhmm'
state_names(object) <- value

## S3 replacement method for class 'mnhmm'
state_names(object) <- value
```

Arguments

`object`	object An object of class `hmm`, `mhmm`, `nhmm`, or `mnhmm`.
`value`	A character vector containing the new state names, or a list of such vectors in case of mixture models.

Value

A character vector containing the state names, or a list of such vectors in case of mixture models.

The original object with updated state names.

`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, ...)
```

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.
`...`	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)
```

`summary.mnhmm` *Summary method for mixture non-homogenous hidden Markov models*

Description

Summary method for mixture non-homogenous hidden Markov models

Usage

```
## S3 method for class 'mnhmm'
summary(object, ...)
```

Arguments

`object`	Non-homogeneous hidden Markov model of class `mnhmm`.
`...`	Ignored

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)
```

update.nhmm *Update Covariate Values of NHMM*

Description

This function can be used to replace original covariate values of NHMMs. The responses, model formulae and estimated coefficients are not altered.

Usage

```
## S3 method for class 'nhmm'
update(object, newdata, drop_levels = FALSE, ...)

## S3 method for class 'mnhmm'
update(object, newdata, drop_levels = FALSE, ...)
```

Arguments

object	An object of class nhmm or mnhmm.
newdata	A data frame containing the new covariate values.
drop_levels	if TRUE, drops unused factor levels from newdata before creating design matrices. The default is FALSE.
...	Ignored.

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.

Alternatively, by using the non-homogeneous model via `estimate_mnhmm` you can compute the standard errors of the coefficients using the bootstrap method.

Value

Matrix containing the variance-covariance matrix of coefficients.

Index

∗ **datasets**
biofam3c, 5
colorpalette, 28
fanhmm_leaves, 36
hmm_biofam, 56
hmm_mvad, 58
leaves, 60
mhmm_biofam, 65
mhmm_mvad, 68

biofam3c, 5
biofam3c(), *58*, *68*
bootstrap_coefs, 6
build_hmm, 8
build_hmm(), *41*, *58, 59*, *63*, *77, 78*, *89*, *92*, *111*
build_lcm, 12
build_lcm(), *41*, *82*
build_mhmm, 16
build_mhmm(), *41*, *68*, *70*, *74*, *82*, *84*, *90*, *94*, *110*
build_mm, 22
build_mm(), *22*, *41*, *77*
build_mmm, 24
build_mmm(), *24*, *41*, *82*

cluster_names, 27
cluster_names<-, 27
coef.mnhmm *(*coef.nhmm*)*, 28
coef.nhmm, 28
colorpalette, 28
colorpalette(), *73, 74*, *86*, *102*, *105*

data_to_stslist, 29
data_to_stslist(), *99*

estimate_mnhmm, 30
estimate_mnhmm(), *97*
estimate_nhmm, 33
estimate_nhmm(), *30*, *32*, *96*, *98*

fanhmm_leaves, 36
fit_model, 37
fit_model(), *9*, *14*, *18*, *26*, *58, 59*, *63*, *68*, *70*, *74*, *78*, *84*, *89, 90*, *92*, *94*, *110, 111*
formula(), *13*, *17*, *24*, *31*, *33, 34*, *94*, *96*, *98*
forward_backward, 49
future::plan(), *7*, *35*

get_cluster_probs, 50
get_emission_probs, 51
get_initial_probs, 51
get_marginals, 52
get_transition_probs, 53
ggplot2::theme(), *107*
ggseqplot::ggseqdplot(), *107*
ggseqplot::ggseqiplot(), *56*, *107*
gridplot, 54
gridplot(), *85*, *99*

hidden_paths, 55
hidden_paths(), *70*, *72*, *74*, *100*, *104*
hmm_biofam, 56, *56*
hmm_biofam(), *63*, *78*, *111*
hmm_mvad, 58
hmm_mvad(), *78*

igraph::igraph.plotting(), *77*, *83*
igraph::layout_(), *77*, *82*
igraph::plot.igraph(), *78*, *84*

leaves, 60
levels(), *96*, *98*
logLik.hmm, 61
logLik.mhmm *(*logLik.hmm*)*, 61
logLik.mnhmm *(*logLik.nhmm*)*, 61
logLik.nhmm, 61

mc_to_sc, 62
mc_to_sc(), *64*, *77, 78*, *82*
mc_to_sc_data, 63
menu(), *72*

mhmm_biofam, 65
mhmm_biofam(), *84*, *90*, *110*
mhmm_mvad, 68
mhmm_mvad(), *84*
most_probable_cluster, 70
mssplot, 71

nloptr::nloptr(), *32*, *34, 35*, *37–39*
nobs.hmm, 74
nobs.mhmm *(*nobs.hmm*)*, 74
nobs.mnhmm *(*nobs.hmm*)*, 74
nobs.nhmm *(*nobs.hmm*)*, 74

par(), *77*, *83*
permute_states, 75
plot.hmm, 76
plot.hmm(), *9*, *23*, *41*
plot.mhmm, 81
plot.mhmm(), *14*, *18*, *26*, *41*, *74*
plot.ssp, 85
plot.ssp(), *99*
plot_colors, 86
plot_colors(), *29*
posterior_cluster_probabilities, 86
posterior_cluster_probabilities(), *50*
posterior_probs, 87
predict.mnhmm *(*predict.nhmm*)*, 88
predict.nhmm, 88
print.hmm, 89
print.mhmm *(*print.hmm*)*, 89
print.mnhmm *(*print.hmm*)*, 89
print.nhmm *(*print.hmm*)*, 89
print.summary_mhmm *(*print.hmm*)*, 89
progressr::with_progress(), *35*

return_msg, 90

separate_mhmm, 90
separate_mhmm(), *14*, *18*, *26*, *41*
seqdef(), *63, 64*, *92*, *103*
seqHMM *(*seqHMM-package*)*, 4
seqHMM-deprecated, 91
seqHMM-package, 4
simulate_emission_probs *(*simulate_initial_probs*)*, 92
simulate_hmm, 91
simulate_initial_probs, 92
simulate_mhmm, 93
simulate_mhmm(), *92*
simulate_mnhmm, 95
simulate_nhmm, 97
simulate_transition_probs *(*simulate_initial_probs*)*, 92
simulate_transition_probs(), *8*
sort_sequences, 99
ssp, 99
ssp(), *54*, *85*
ssplot, 103
ssplot(), *85*
stacked_sequence_plot, *64*, 106
stacked_sequence_plot(), *56*, *91, 92*, *102*
state_names, 108
state_names<- *(*state_names*)*, 108
stats::cmdscale(), *72*, *101*, *104*, *107*
stats::rnorm(), *38*
stslist_to_data *(*data_to_stslist*)*, 29
summary.mhmm, 109
summary.mhmm(), *14*, *18*, *26*, *41*
summary.mnhmm, 110

TraMineR::biofam(), *5, 6*, *56*, *58*, *65*, *68*
TraMineR::mvad(), *58, 59*, *68*, *70*
TraMineR::plot.stslist(), *72*, *101*, *104*
TraMineR::seqdef, *106*
TraMineR::seqdef(), *8*, *12*, *16*, *22*, *24*, *29*, *72, 73*, *99–102*, *104, 105*, *107*
TraMineR::seqdist(), *99*, *107*
TraMineR::seqMD(), *99*
TraMineR::seqplot(), *72*, *74*, *78*, *84*, *100*, *102*, *104*, *106*
trim_model, 111

update.mnhmm *(*update.nhmm*)*, 112
update.nhmm, 112
utils::menu(), *82*

vcov.mhmm, 112
vcov.mhmm(), *109*