Package: plotHMM (via r-universe)

September 7, 2024

Type Package

Title Plot Hidden Markov Models

Version 2023.8.28

Description Hidden Markov Models are useful for modeling sequential data. This package provides several functions implemented in C++ for explaining the algorithms used for Hidden Markov Models (forward, backward, decoding, learning).

License GPL (>= 2)

Imports Rcpp (>= 1.0.7)

LinkingTo Rcpp, RcppArmadillo

Suggests testthat, knitr, markdown, R.utils, covr, depmixS4, data.table, ggplot2, neuroblastoma, microbenchmark

VignetteBuilder knitr

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

RemoteUrl https://github.com/tdhock/plothmm

RemoteRef HEAD

RemoteSha e235971997eeeefbec7c33763060a87d47e66e10

Contents

backward_interface *Backward algorithm*

Description

Efficient implementation of backward algorithm in C++ code, for N data and S states.

Usage

```
backward_interface(
  log_emission_mat, log_transition_mat)
```

Arguments

`log_emission_mat`	N x S numeric matrix of log likelihood of observing each data point in each state.
`log_transition_mat`	S x S numeric matrix; log_transition_mat[i,j] is the log probability of going from state i to state j.

Value

N x S numeric matrix of backward log likelihood.

Author(s)

Toby Dylan Hocking

Examples

```
##simulated data.
seg.mean.vec <- c(2, 0, -1, 0)
data.mean.vec <- rep(seg.mean.vec, each=10)
set.seed(1)
N.data <- length(data.mean.vec)
y.vec <- rnorm(N.data, data.mean.vec)
##model.
n.states <- 3
log.A.mat <- log(matrix(1/n.states, n.states, n.states))
state.mean.vec <- c(-1, 0, 1)*0.1
sd.param <- 1
log.emission.mat <- dnorm(
  y.vec,
  matrix(state.mean.vec, N.data, n.states, byrow=TRUE),
  sd.param,
  log=TRUE)
plotHMM::backward_interface(log.emission.mat, log.A.mat)
```

buggy.5states *Buggy data with 5 states*

Description

Data was observed to error with depmixS4 and five states

Usage

```
data("buggy.5states")
```

Format

The format is a data table.

buggy.data *Buggy data with one state*

Description

This data set was known to produce an error with depmixS4 using one state.

Usage

```
data("buggy.data")
```

Format

The format is a data table.

eln *Log probability arithmetic*

Description

Binary operators in log probability space, to avoid numerical underflow.

Usage

```
elnproduct(elnx, elny)
elnsum(elnx, elny)
logsumexp(exponents.vec)
```

Arguments

`elnx, elny, exponents.vec`
: numeric vectors of log probabilities.

Value

Numeric vector with one (logsumexp) or more (others) log probability value(s).

Author(s)

Toby Dylan Hocking

References

http://bozeman.genome.washington.edu/compbio/mbt599_2006/hmm_scaling_revised.pdf

Examples

```
px <- c(0.1, 0.5, 0.9)
py <- c(0.001, 0.123, 0.999)
lx <- log(px)
ly <- log(py)
library(plotHMM)
elnproduct(lx, ly)
elnsum(lx, ly)
logsumexp(ly)
```

`forward_interface` *Forward algorithm*

Description

Efficient implementation of forward algorithm in C++ code, for N data and S states.

Usage

```
forward_interface(
  log_emission_mat, log_transition_mat, log_initial_prob_vec)
```

Arguments

`log_emission_mat`
: N x S numeric matrix of log likelihood of observing each data point in each state.

`log_transition_mat`
: S x S numeric matrix; log_transition_mat[i,j] is the log probability of going from state i to state j.

`log_initial_prob_vec`	S numeric vector of log probabilities of observing each state at the beginning of the sequence.

Value

list with two elements

`log_alpha`	N x S numeric matrix of forward log likelihood at each data/state.
`log_lik`	numeric scalar total log likelihood of data given model parameters.

Author(s)

Toby Dylan Hocking

Examples

```
##simulated data.
seg.mean.vec <- c(2, 0, -1, 0)
data.mean.vec <- rep(seg.mean.vec, each=10)
set.seed(1)
N.data <- length(data.mean.vec)
y.vec <- rnorm(N.data, data.mean.vec)
##model.
n.states <- 3
log.A.mat <- log(matrix(1/n.states, n.states, n.states))
state.mean.vec <- c(-1, 0, 1)*0.1
sd.param <- 1
log.pi.vec <- log(rep(1/n.states, n.states))
log.emission.mat <- dnorm(
  y.vec,
  matrix(state.mean.vec, N.data, n.states, byrow=TRUE),
  sd.param,
  log=TRUE)
plotHMM::forward_interface(log.emission.mat, log.A.mat, log.pi.vec)
```

`multiply_interface` *Multiply algorithm*

Description

Efficient implementation of multiply algorithm in C++ code, for N data and S states.

Usage

```
multiply_interface(
  log_alpha_mat, log_beta_mat)
```

Arguments

`log_alpha_mat`, `log_beta_mat`
: N x S numeric matrices of log probabilities, from forward and backward algorithms.

Value

N x S numeric matrix of overall log likelihood.

Author(s)

Toby Dylan Hocking

Examples

```
##simulated data.
seg.mean.vec <- c(2, 0, -1, 0)
data.mean.vec <- rep(seg.mean.vec, each=10)
set.seed(1)
N.data <- length(data.mean.vec)
y.vec <- rnorm(N.data, data.mean.vec)
##model.
n.states <- 3
log.A.mat <- log(matrix(1/n.states, n.states, n.states))
state.mean.vec <- c(-1, 0, 1)*0.1
sd.param <- 1
log.emission.mat <- dnorm(
  y.vec,
  matrix(state.mean.vec, N.data, n.states, byrow=TRUE),
  sd.param,
  log=TRUE)
log.pi.vec <- log(rep(1/n.states, n.states))
f.list <- plotHMM::forward_interface(log.emission.mat, log.A.mat, log.pi.vec)
b.mat <- plotHMM::backward_interface(log.emission.mat, log.A.mat)
log.gamma.mat <- plotHMM::multiply_interface(f.list$log_alpha, b.mat)
prob.mat <- exp(log.gamma.mat)
rowSums(prob.mat)
```

`pairwise_interface` *Pairwise algorithm*

Description

Efficient implementation of pairwise algorithm in C++ code, for N data and S states.

Usage

```
pairwise_interface(
  log_emission_mat, log_transition_mat, log_alpha_mat, log_beta_mat)
```

Arguments

`log_emission_mat, log_alpha_mat, log_beta_mat`
: N x S numeric matrices of log likelihood.

`log_transition_mat`
: S x S numeric matrix; log_transition_mat[i,j] is the log probability of going from state i to state j.

Value

S x S x N-1 numeric array of log likelihood.

Author(s)

Toby Dylan Hocking

Examples

```
##simulated data.
seg.mean.vec <- c(2, 0, -1, 0)
data.mean.vec <- rep(seg.mean.vec, each=10)
set.seed(1)
N.data <- length(data.mean.vec)
y.vec <- rnorm(N.data, data.mean.vec)
##model.
n.states <- 3
log.A.mat <- log(matrix(1/n.states, n.states, n.states))
state.mean.vec <- c(-1, 0, 1)*0.1
sd.param <- 1
log.emission.mat <- dnorm(
  y.vec,
  matrix(state.mean.vec, N.data, n.states, byrow=TRUE),
  sd.param,
  log=TRUE)
log.pi.vec <- log(rep(1/n.states, n.states))
f.list <- plotHMM::forward_interface(log.emission.mat, log.A.mat, log.pi.vec)
b.mat <- plotHMM::backward_interface(log.emission.mat, log.A.mat)
log.gamma.mat <- plotHMM::multiply_interface(f.list$log_alpha, b.mat)
prob.mat <- exp(log.gamma.mat)
plotHMM::pairwise_interface(log.emission.mat, log.A.mat, f.list$log_alpha, b.mat)
```

`transition_interface` *Transition algorithm*

Description

Efficient implementation of transition algorithm in C++ code, for T transitions and S states.

Usage

```
transition_interface(
  log_gamma_mat, log_xi_array)
```

Arguments

`log_gamma_mat`	T x S numeric matrix, taken by removing the last row from the log probabilities from multiply.
`log_xi_array`	S x S x T numeric array of log probabilities from pairwise.

Value

S x S numeric array of log probabilities (new transition matrix).

Author(s)

Toby Dylan Hocking

Examples

```
##simulated data.
seg.mean.vec <- c(2, 0, -1, 0)
data.mean.vec <- rep(seg.mean.vec, each=10)
set.seed(1)
N.data <- length(data.mean.vec)
y.vec <- rnorm(N.data, data.mean.vec)
##model.
n.states <- 3
log.A.mat <- log(matrix(1/n.states, n.states, n.states))
state.mean.vec <- c(-1, 0, 1)*0.1
sd.param <- 1
log.emission.mat <- dnorm(
  y.vec,
  matrix(state.mean.vec, N.data, n.states, byrow=TRUE),
  sd.param,
  log=TRUE)
log.pi.vec <- log(rep(1/n.states, n.states))
f.list <- plotHMM::forward_interface(log.emission.mat, log.A.mat, log.pi.vec)
b.mat <- plotHMM::backward_interface(log.emission.mat, log.A.mat)
log.gamma.mat <- plotHMM::multiply_interface(f.list$log_alpha, b.mat)
prob.mat <- exp(log.gamma.mat)
log.xi.array <- plotHMM::pairwise_interface(
  log.emission.mat, log.A.mat, f.list$log_alpha, b.mat)
plotHMM::transition_interface(log.gamma.mat[-N.data,], log.xi.array)
```

`viterbi_interface` *Viterbi algorithm*

Description

Efficient implementation of Viterbi algorithm in C++ code, for N data and S states.

Usage

```
viterbi_interface(
  log_emission_mat, log_transition_mat, log_initial_prob_vec)
```

Arguments

`log_emission_mat`	N x S numeric matrix of log likelihood of observing each data point in each state.
`log_transition_mat`	S x S numeric matrix; log_transition_mat[i,j] is the log probability of going from state i to state j.
`log_initial_prob_vec`	S numeric vector of log probabilities of observing each state at the beginning of the sequence.

Value

list with elements

`log_max_prob`	N x S numeric matrix of max log probabilities.
`best_state`	N x S integer matrix. First row is fixed at zero, other rows indicate best states (from 1 to S).
`state_seq`	N integer vector, best overall state sequence (entries from 1 to S).

Author(s)

Toby Dylan Hocking

Examples

```
##simulated data.
seg.mean.vec <- c(2, 0, -1, 0)
data.mean.vec <- rep(seg.mean.vec, each=2)
N.data <- length(data.mean.vec)
sd.param <- 0.1
set.seed(1)
y.vec <- rnorm(N.data, data.mean.vec, sd.param)
##model.
state.mean.vec <- unique(seg.mean.vec)
```

```
n.states <- length(state.mean.vec)
log.A.mat <- log(matrix(1/n.states, n.states, n.states))
log.pi.vec <- log(rep(1/n.states, n.states))
log.emission.mat <- dnorm(
  y.vec,
  matrix(state.mean.vec, N.data, n.states, byrow=TRUE),
  sd.param,
  log=TRUE)
plotHMM::viterbi_interface(log.emission.mat, log.A.mat, log.pi.vec)
```

Index