Package ‘label.switching’

July 2, 2014

Type Package
Title Relabelling MCMC outputs of mixture models
Version 1.2
Date 2013-12-04
Author Panagiotis Papastamoulis
Maintainer Panagiotis Papastamoulis <papast@yahoogr>
Description The Bayesian estimation of mixture models (and more general hidden Markov models) suffers from the label switching phenomenon, making the MCMC output non-identifiable. This package can be used in order to deal with this problem using various relabelling algorithms.
Depends lpSolve, combinat
License GPL-2
NeedsCompilation no
Repository CRAN
Date/Publication 2013-12-04 11:42:13

R topics documented:

label.switching-package ........................................... 2
compare.clust ....................................................... 3
data_list .......................................................... 4
ecr ................................................................. 5
ecr.iterative.1 ....................................................... 6
ecr.iterative.2 ....................................................... 8
label.switching ...................................................... 9
lamb ................................................................. 12
permute.mcmc ........................................................ 13
pra ................................................................. 14
sjw ................................................................. 15
stephens ........................................................... 17
Description

This package can be used to reorder MCMC outputs of parameters of mixture models (or more general ones, like hidden Markov). The label switching phenomenon is a fundamental problem to MCMC estimation of the parameters of such models. This package contains six label switching solving algorithms: the default and iterative versions of ECR algorithm (Papastamoulis and Iliopoulos, 2010, 2011, Rodriguez and Walker, 2012, Papastamoulis, 2013), the Kullback-Leibler based algorithm of Stephens (2000), the probabilistic relabelling algorithm of Sperrin et al (2010) and the PRA algorithm (Marin et al, 2005, Marin and Robert, 2007). The user input depends on each method.

Details

Table: Function Method Input
<table>
<thead>
<tr>
<th>Function</th>
<th>Method</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecr</td>
<td>ECR (default)</td>
<td>zpivot, z, K</td>
</tr>
<tr>
<td>ecr.iterative.1</td>
<td>ECR (iterative vs. 1)</td>
<td>z, K</td>
</tr>
<tr>
<td>ecr.iterative.2</td>
<td>ECR (iterative vs. 2)</td>
<td>z, K, p</td>
</tr>
<tr>
<td>pra</td>
<td>PRA</td>
<td>mcmc, pivot</td>
</tr>
<tr>
<td>stephens</td>
<td>Stephens</td>
<td>p</td>
</tr>
<tr>
<td>sjw</td>
<td>Probabilistic</td>
<td>mcmc, z, complete, x</td>
</tr>
</tbody>
</table>

This is NOT a package to simulate MCMC samples from the posterior distribution of mixture models. MCMC output and related information serves as input to the available methods. There are six functions that can be used to post-process the MCMC output:

Each function returns an $m \times K$ array of permutations, where $m$ and $K$ denote the MCMC sample size and number of mixture components, respectively. Next, these permutations can be applied to reorder the MCMC sample by applying the function permute.mcmc. The user can call any of the above functions simultaneously using the main function of the package: label.switching.
**Note**

The methods `ecr`, `ecr.iterative.1`, `ecr.iterative.2`, `stephens` are solving the label switching problem using the function `lpAssign` of the package `lpSolve`. This is an integer programming algorithm for the solution of the assignment problem. Hence, these functions are computationally efficient even in cases where the number of components is quite large. On the other hand, methods `pra` and `sjw` are not designed in this way, so they are not suggested for large $K$.

**Author(s)**

Panagiotis Papastamoulis
Maintainer: <papast@yahoo.gr>

**References**


**See Also**

`label.switching`

**Description**

Given a pivot allocation vector, a set of simulated allocations and a set of permutations from different relabelling algorithms, this function relabels the permutations so that all methods maximize their similarity with the pivot. This is helpful when comparing different different label switching algorithms.
Usage

```r
call.compare.clust(pivot.clust, perms, z, K)
```

Arguments

- **pivot.clust**: a pivot allocation vector of the `n` observations among the `K` clusters.
- **perms**: a list containing `f` permutation arrays, as returned by `label.switching` function.
- **z**: a set of simulated allocation arrays.
- **K**: number of mixture components

Value

- **similarity**: `(f+1)K \times (f+1)` matrix containing the similarity coefficient of the resulting clusters.
- **clusters**: `f \times n` array of single best clusterings, relabelled in order to maximize their similarity with `pivot.clust`.

Author(s)

Panagiotis Papastamoulis

See Also

- `label.switching`

---

### data_list

*Simulated MCMC sample and related information*

**Description**

This is a (very) small MCMC sample corresponding to data of 5 observations from a mixture 2 normal distributions. The MCMC sample consists of 300 iterations. It is stored to `data_list$mcmc.pars.data_list$mcmc.pars[,1]` corresponds to means, `data_list$mcmc.pars[,2]` corresponds to variances and `data_list$mcmc.pars[,3]` corresponds to weights.

**Usage**

```r
data_list
```

**Format**

A list containing simulated MCMC sample and all information required for the relabelling algorithms.
Description

This function applies the standard version of Equivalence Classes Representatives (ECR) algorithm (Papastamoulis and Iliopoulos, 2010). The set of all allocation variables is partitioned into equivalence classes and exactly one representative is chosen from each class. The practical implementation of this idea is to reorder the output so that all simulated allocation vectors (z) are as similar as possible with a pivot allocation vector (zpivot). The user has to specify this pivot allocation vector as a good allocation of the observations among the mixture components. Some typical choices are the allocations that correspond to the complete or non-complete MAP estimate.

Usage

ecr(zpivot, z, K)

Arguments

- zpivot: n-dimensional integer vector (z1, ..., zn) with zi ∈ {1, ..., K}, i = 1, ..., n.
- z: m × n integer array of the latent allocation vectors generated from an MCMC algorithm.
- K: the number of mixture components (at least equal to 2).

Details

zpivot should be chosen as an allocation vector that corresponds to a high-posterior density area, or in general as an allocation that is considered as a good allocation of the observations among the K components.

Value

- permutations: m × K dimensional array of permutations

Author(s)

Panagiotis Papastamoulis

References


See Also

permute.mcmc, label.switching, ecr.iterative.1, ecr.iterative.2
### Examples

```r
# load a toy example: MCMC output consists of the random beta model
# applied to a normal mixture of \code{K=2} components. The
# number of observations is equal to \code{n=5}. The number
# of MCMC samples is equal to \code{m=300}. The 300
# simulated allocations are stored to array \code{z}. The
# complete MAP estimate corresponds to iteration \code{mapindex}.

data("mcmc_output")
z<-data_list$"z"
K<-data_list$"K"
mapindex<-data_list$"mapindex"

# mcmc parameters are stored to array \code{mcmc_pars}
mcmc_pars<-data_list$"mcmc_pars"
# mcmc_pars[,1]: simulated means of the two components
# mcmc_pars[,2]: simulated variances
# mcmc_pars[,3]: simulated weights
run<-ecr(pivot = z[mapindex,], z = z, K = K)
# apply the permutations returned by typing:
reordered_mcmc<-permute.mcmc(mcmc_pars, run$permutations)
# reordered_mcmc[,1]: reordered means of the two components
# reordered_mcmc[,2]: reordered variances
# reordered_mcmc[,3]: reordered weights
```

---

**ecr.iterative.1**  
**ECR algorithm (iterative version 1)**

---

### Description

This function applies the first iterative version of Equivalence Classes Representatives (ECR) algorithm (Papastamoulis and Iliopoulos, 2010, Rodriguez and Walker, 2012). The set of all allocation variables is partitioned into equivalence classes and exactly one representative is chosen from each class. The difference with the default version of ECR algorithm is that no pivot is required and the method is iterative, until a fixed pivot has been found.

### Usage

```r
ecr.iterative.1(z, K, opt_init)
```

### Arguments

- **z**  
  \( m \times n \) integer array of the latent allocation vectors generated from an MCMC algorithm.

- **K**  
  the number of mixture components (at least equal to 2).

- **opt_init**  
  An (optional) \( m \times K \) array of permutations to initialize the algorithm. The identity permutation is used if it is not specified.
Value

- **permutations**: $m \times K$ dimensional array of permutations
- **iterations**: integer denoting the number of iterations until convergence

**Author(s)**

Panagiotis Papastamoulis

**References**


**See Also**

- `permute.mcmc`
- `label.switching`
- `ecr`
- `ecr.iterative`

**Examples**

```r
# load a toy example: MCMC output consists of the random beta model
# applied to a normal mixture of K=2 components. The number of
# observations is equal to n=5. The number of MCMC samples is
# equal to m=1000. The 300 simulated allocations are stored to
# array z.
data("mcmc_output")
# mcmc parameters are stored to array mcmc.pars
mcmc.pars<-data_list$mcmc.pars
z<-data_list"z"
K<-data_list"K"
# mcmc.pars[,1]: simulated means of the two components
# mcmc.pars[,2]: simulated variances
# mcmc.pars[,3]: simulated weights
# the relabelling algorithm will run with the default initialization
# (no opt_init is specified)
run<-ecr.iterative.1(z = z, K = K)
# apply the permutations returned by typing:
reordered.mcmc<-permute.mcmc(mcmc.pars,run$permutations)
# reordered.mcmc[,1]: reordered means of the two components
# reordered.mcmc[,2]: reordered variances
# reordered.mcmc[,3]: reordered weights
```
Description

This function applies the second iterative version of Equivalence Classes Representatives (ECR) algorithm (Papastamoulis and Iliopoulos, 2010, Rodriguez and Walker, 2012). The set of all allocation variables is partitioned into equivalence classes and exactly one representative is chosen from each class. In this version the $m \times n \times K$ of allocation probabilities should be given as input as well.

Usage

ecr.iterative.2(z, K, p)

Arguments

z $m \times n$ integer array of the latent allocation vectors generated from an MCMC algorithm.

K the number of mixture components (at least equal to 2).

p $m \times n \times K$ dimensional array of allocation probabilities of the n observations among the K mixture components, for each iteration $t = 1, \ldots, m$ of the MCMC algorithm.

Details

For a given MCMC iteration $t = 1, \ldots, m$, let $w_k^{(t)}$ and $\theta_k^{(t)}$, $k = 1, \ldots, K$ denote the simulated mixture weights and component specific parameters respectively. Then, the $(t, i, k)$ element of p corresponds to the conditional probability that observation $i = 1, \ldots, n$ belongs to component k and is proportional to $p_{tik} \propto w_k^{(t)} f(x_i|\theta_k^{(t)}), k = 1, \ldots, K$, where $f(x_i|\theta_k)$ denotes the density of component k. This means that:

$$p_{tik} = \frac{w_k^{(t)} f(x_i|\theta_k^{(t)})}{w_1^{(t)} f(x_i|\theta_1^{(t)}) + \ldots + w_K^{(t)} f(x_i|\theta_K^{(t)})}.$$

In case of hidden Markov models, the probabilities $w_k$ should be replaced with the proper left (normalized) eigenvector of the state-transition matrix.

Value

permutations $m \times K$ dimensional array of permutations

iterations integer denoting the number of iterations until convergence

Author(s)

Panagiotis Papastamoulis
References


See Also

permute.mcmc, label.switching, ecr, ecr.iterative.1, stephens

Examples

# load a toy example: MCMC output consists of the random beta model
# applied to a normal mixture of \code{K=2} components. The number of
# observations is equal to \code{n=5}. The number of MCMC samples is
# equal to \code{m=1000}. The 300 simulated allocations are stored to
# array \code{z}. The matrix of allocation probabilities is stored to
# array \code{p}.

data("mcmc_output")
z<-data_list$"z"
K<-data_list$"K"
p<-data_list$"p"

# mcmc parameters are stored to array \code{mcmc.pars}
mcmc.pars<-data_list$"mcmc.pars"

# mcmc.pars[,1]: simulated means of the two components
# mcmc.pars[,2]: simulated variances
# mcmc.pars[,3]: simulated weights
# the relabelling algorithm will run with the default initialization
# (no opt_init is specified)
run<-ecr.iterative.2(z = z, K = 2, p = p)
# apply the permutations returned by typing:
reordered.mcmc<-permute.mcmc(mcmc.pars,run$permutations)

# reordered.mcmc[,1]: reordered means of the two mixture components
# reordered.mcmc[,2]: reordered variances of the two components
# reordered.mcmc[,3]: reordered weights of the two components

label.switching  

Main calling function

Description

This is the main function of the package. It is used to reorder a simulated MCMC sample of the parameters of a mixture (or more general a hidden Markov model) according to six label switching solving methods: ECR algorithm (default version), ECR algorithm (two iterative versions), PRA algorithm, Stephens’ algorithm and a probabilistic relabelling algorithm. The input depends on the type of the label switching method. The output contains a list with the permutation returned by each method, the corresponding single best clusterings and the CPU time demanded for each
method. In what follows: $m$ denotes the number of MCMC iterations, $n$ denotes the sample size of the observed data, $K$ denotes the number of mixture components and $J$ the number of different types of parameters of the model.

Usage

```r
label.switching(method, zpivot, z, K, prapivot, p, complete, mcmc, sjwinit, data)
```

Arguments

- `method`: any non-empty subset of `c("ECR","ECR-ITERATIVE-1","PRA","ECR-ITERATIVE-2","STEPHENS","SJW")` indicating the desired label-switching solving method.
- `zpivot`: $d \times n$-dimensional array of pivot allocation vectors, where $d$ denotes the number of pivots. This is demanded by the `ecr` method. The method will be applied $d$ times.
- `z`: $m \times n$ integer array of the latent allocation vectors generated from an MCMC algorithm.
- `K`: the number of mixture components. This is demanded by the `ecr`, `ecr.iterative.1` and `ecr.iterative.2` methods.
- `prapivot`: $K \times J$ array containing the parameter that will be used as a pivot by the `pra` method.
- `p`: $m \times n \times K$ dimensional array of allocation probabilities of the $n$ observations among the $K$ mixture components, for each iteration $t = 1, \ldots, m$ of the MCMC algorithm. This is demanded by the `ecr.iterative.2` and `stephens` methods.
- `complete`: function that returns the complete log-likelihood of the mixture model. Demanded by the `sjw` method.
- `mcmc`: $m \times K \times J$ array of simulated MCMC parameters. Needed by the `sjw` and `pra` methods.
- `sjwinit`: an index pointing at the MCMC iteration whose parameters will initialize the `sjw` algorithm (optional).
- `data`: $n$-dimensional data vector/array. Needed by the `sjw` algorithm.

Details

The methods `sjw` and `pra` are not suggested for large number of components.

Value

- `permutations`: an $m \times K$ array of permutations per method.
- `clusters`: an $n$ dimensional vector of best clustering of the the observations for each method.
- `timings`: CPU time needed for each relabelling method.
- `similarity`: correlation matrix between the label switching solving methods in terms of their matching best-clustering allocations.
Author(s)
Panagiotis Papastamoulis

See Also
ecr, ecr.iterative.1, ecr.iterative.2, stephens.pra, sjw

Examples

# We will apply four methods:
# ECR, ECR-ITERATIVE-1, PRA and STEPHENS.
# default ECR will use two different pivots.

# load a toy example: MCMC output consists of the random beta model
# applied to a normal mixture of \( K=2 \) components. The number of
# observations is equal to \( n=5 \). The number of MCMC samples is
# equal to \( m=300 \). simulated allocations are stored to array \( z \).
# mcmc output
mcmc.pars$<-$data_list$mcmc.pars$
# mcmc parameters are stored to array \( mcmc.pars \)
# mcmc.pars[,1]: simulated means of the two components
# mcmc.pars[,2]: simulated variances
# mcmc.pars[,3]: simulated weights
# We will use two pivots for default ECR algorithm:
# the first one corresponds to iteration \( \text{mapindex} \) (complete MAP)
# the second one corresponds to iteration \( \text{mapindex.non} \) (observed MAP)
# The array \( p \) corresponds to the allocation probabilities
z$<-$data_list$z$
K$<-$data_list$K$
p$<-$data_list$p$
mapindex$<-$data_list$mapindex$
mapindex.non$<-$data_list$mapindex.non$
# The PRA method will use as pivot the iteration that corresponds to
# the non-complete MAP estimate (mapindex).

# Apply the four methods by typing:
ls$<-$label.switching(method=c("ECR","ECR-ITERATIVE-1","PRA","STEPHENS"),
zpivot=z[c(mapindex,mapindex.non),],z = z,K = K,
prapivot = mcmc.pars[mapindex,],p=p,mcmc = mcmc.pars)

# plot the raw and reordered means of the \( K=2 \) normal mixture components for each method
par(mfrow = c(2,3))
# raw MCMC output for the means (with label switching)
matplot(mcmc.pars[,1],type="l",
 xlab="iteration",main="Raw MCMC output",ylab = "means")
# Reordered outputs
matplot(permute.mcmc(mcmc.pars,ls$permutations$"ECR-1")$output[,1],type="l",
 xlab="iteration",main="ECR (1st pivot)",ylab = "means")
matplot(permute.mcmc(mcmc.pars,ls$permutations$"ECR-2")$output[,1],type="l",
 xlab="iteration",main="ECR (2nd pivot)",ylab = "means")
matplot(permute.mcmc(mcmc.pars,ls$permutations$"ECR-ITERATIVE-1")$output[,1],
type="1",xlab="iteration",main="ECR-iterative-1",ylab = "means")
matplot(permute.mcmc(mcmc.pars,ls$permutations$"PRA")$output[,1],type="l",
  xlab="iteration",main="PRA",ylab = "means")
matplot(permute.mcmc(mcmc.pars,ls$permutations$"STEPHENS")$output[,1],type="l",
  xlab="iteration",main="STEPHENS",ylab = "means")

# if the user wants to apply the SJW algorithm as well:
# The SJW method needs to define the complete log-likelihood of the
# model. For the univariate normal mixture, this is done as follows:

complete.normal.loglikelihood<-function(x,z,pars){
  #x: denotes the n data points
  #z: denotes an allocation vector (size=n)
  #pars: K\times 3 vector of means, variance, weights
  # pars[k,1]: corresponds to the mean of component k
  # pars[k,2]: corresponds to the variance of component k
  # pars[k,3]: corresponds to the weight of component k
  g <- dim(pars)[1]
  n <- length(x)
  logl<- rep(0, n)
  logpi <- log(pars[,3])
  mean <- pars[,1]
  sigma <- sqrt(pars[,2])
  logl<-logpi[z] + dnorm(x,mean = mean[z],sd = sigma[z],log = T)
  return(sum(logl))
}

# and then run (after removing all #):
#ls<-lable.switching(method=c("ECR","ECR-ITERATIVE-1","ECR-ITERATIVE-2",
#"PRA","STEPHENS","SJW"),zpivot=z[mapindex,mapindex.non,],z = z,
#K = K,prapivot = mcmc.pars[mapindex,],p=p,
#complete = complete.normal.loglikelihood,mcmc.pars,
#data = x)

---

**lamb**  
*Fetal lamb dataset*

**Description**

240 body movement measurements of a fetal lamb at consecutive 5 second intervals.

**Usage**

`lamb`

**Format**

Count data.
permute.mcmc

References
Leroux B, Puterman M (1992). Maximum Penalized Likelihood estimation for independent and

permute.mcmc  Reorder MCMC samples

Description
This function applies the permutation returned by any relabelling algorithm to a simulated MCMC
output.

Usage
permute.mcmc(mcmc, permutations)

Arguments
mcmc  \( m \times K \times J \) array of simulated MCMC parameters.
permutations  \( m \times K \) dimensional array of permutations.

Value
output  \( m \times K \times J \) array of reordered MCMC parameters.

Author(s)
Panagiotis Papastamoulis

See Also
label.switching, ecr, ecr.iterative.1, ecr.iterative.2, stephens, pra, sjw

Examples
# load MCMC simulated data
data("mcmc_output")
mcmc.pars<-data_list$"mcmc.pars"
z<-data_list$"z"
K<-data_list$"K"

# apply \code{ecr.iterative.1} algorithm
run<-ecr.iterative.1(z = z, K = 2)
# reorder the MCMC output according to this method:
reordered.mcmc<-permute.mcmc(mcmc.pars, run$permutations)
# reordered.mcmc[,1]: reordered means of the two components
# reordered.mcmc[,2]: reordered variances of the components
# reordered.mcmc[,3]: reordered weights of the two components
PRA algorithm

Description
This function reorders the MCMC output using the geometrically-based Pivotal Reordering Algorithm (PRA) (Marin et al, 2005, Marin and Robert, 2007). The method requires as input the generated MCMC sample and a pivot parameter vector. The user should be careful in order the pivot elements have the same parameters with the generated MCMC output. The simulated MCMC sample should be provided by the useR as a $m \times K \times J$ dimensional array, where $m$ denotes the number of MCMC samples, $K$ denotes the number of mixture components and $J$ corresponds to the number of different parameter types of the model. The pivot should correspond to a high-posterior density point.

Usage
pra(mcmc.pars, pivot)

Arguments
- mcmc.pars $m \times K \times J$ array of simulated MCMC parameters.
- pivot $K \times J$ array containing the parameter that will be used as a pivot.

Details
The positive integer $J$ denotes the number of different parameter types of the model. For example, in a univariate normal mixture model there are $J = 3$ different types: means, variances and weights. In a Poisson mixture there are $J = 2$ types: means and weights.

Value
permutations $m \times K$ dimensional array of permutations

Note
This algorithm is very fast for small number of components. It is not suggested for large number of components due to the calculation of $K!$ distances for each MCMC sample.

Author(s)
Panagiotis Papastamoulis

References
See Also

permute.mcmc.label.switching

Examples

# load a toy example: MCMC output consists of the random beta model
# applied to a normal mixture of \(K=2\) components. The number of
# observations is equal to \(n=5\). The number of MCMC samples is
# equal to \(m=300\). The 1000 generated MCMC samples are stored
# to array mcmc.pars.
data("mcmc_output")
mcmc.pars<-data_list$mcmc.pars"
mapindex<-data_list$mapindex"

# mcmc parameters are stored to array \texttt{mcmc.pars}
# mcmc.pars[,1]: simulated means of the two components
# mcmc.pars[,2]: simulated variances of the two components
# mcmc.pars[,3]: simulated weights of the two components
# We will apply PRA using as pivot the complete MAP estimate
# which corresponds to \texttt{mcmc.pars[mapindex,]}
run<-pra(mcmc = mcmc.pars, pivot = mcmc.pars[mapindex,])
# apply the permutations returned by typing:
reordered.mcmc<-permute.mcmc(mcmc.pars,run$permutations)
# reordered.mcmc[,1]: reordered means of the two components
# reordered.mcmc[,2]: reordered variances of the components
# reordered.mcmc[,3]: reordered weights

sjw

Probabilistic relabelling algorithm

Description

Function to apply the probabilistic relabelling strategy of Sperrin et al (2010).

Usage

sjw(mcmc.pars, z, complete, x, init)

Arguments

mcmc.pars \(m \times K \times J\) array of simulated MCMC parameters.
z \(m \times n\) integer array of the latent allocation vectors generated from an MCMC
algorithm.
complete function that returns the complete log-likelihood of the mixture model.
x \(n\)-dimensional data vector/array
init An (optional) index pointing at the MCMC iteration whose parameters will initialize the algorithm. If it is less or equal to zero, the overall MCMC mean will be used for initialization.
Details

Let \( x = (x_1, \ldots, x_n) \) denote the observed data and \( w, \theta \) denote the mixture weights and component specific parameters, respectively. Assume that \( K \) is the the number of components. Then,

\[
L(w, \theta | x) = \prod_{i=1}^{n} \sum_{k=1}^{K} w_k f_k(x_i | \theta_k),
\]

\( i = 1, \ldots, n \) is the observed likelihood of the mixture model. Given the latent allocation variables \( z = (z_1, \ldots, z_n) \), the complete likelihood of the model is defined as:

\[
L_c(w, \theta | x, z) = \prod_{i=1}^{n} w_{z_i} f_{z_i}(x_i | \theta_{z_i}).
\]

Then, complete corresponds to the log of \( L_c \) and should take as input the following: a vector of \( n \) allocations, the observed data and the parameters of the model as a \( K \times J \) array where \( J \) corresponds to the different parameter types of the model. See the example for an implementation at a univariate normal mixture.

Value

- permutations: \( m \times K \) dimensional array of permutations
- iterations: integer denoting the number of iterations until convergence

Note

This algorithm is not suggested for large number of components due to the computational overload: \( K! \) permutation probabilities are computed at each MCMC iteration. Moreover, the user should carefully provide the complete log-likelihood function of the model as input to the algorithm and this makes its use quite complicated.

Author(s)

Panagiotis Papastamoulis

References


See Also

permute.mcmc, label.switching

Examples

# load a toy example: MCMC output consists of the random beta model
# applied to a normal mixture of \( \text{code}[K=2] \) components. The number of
# observations is equal to \( \text{code}[n=5] \). The number of MCMC samples is
# equal to \( \text{code}[m=300] \).
data("mcmc_output")
mcmc_pars<-data_list$mcmc.pars"
z<-data_list"z"
K<-data_list"K"
x<-data_list"x"

# mcmc parameters are stored to array \code{mcmc_pars}
# mcmc_pars[,,1]: simulated means of the two components
# mcmc_pars[,,2]: simulated variances
# mcmc_pars[,,3]: simulated weights
# The number of different parameters for the univariate
# normal mixture is equal to J = 3: means, variances
# and weights. The generated allocations variables are
# stored to \code{z}. The observed data is stored to \code{x}.
# The complete data log-likelihood is defined as follows:
complete.normal.loglikelihood<-function(x,z,pars){
  # x: data (size = n)
  # z: allocation vector (size = n)
  # pars: K\times J vector of normal mixture parameters:
  # pars[k,1] = mean of the k-normal component
  # pars[k,2] = variance of the k-normal component
  # pars[k,3] = weight of the k-normal component
  # k = 1,...,K
  g <- dim(pars)[1] #K (number of mixture components)
  n <- length(x) #this denotes the sample size
  logl<- rep(0, n)
  logpi <- log(pars[,3])
  mean <- pars[,1]
  sigma <- sqrt(pars[,2])
  logl<-logpi[z] + dnorm(x,mean = mean[z],sd = sigma[z],log = TRUE)
  return(sum(logl))
}

#run the algorithm:
run<sjw(mcmc = mcmc_pars,z = z,
complete = complete.normal.loglikelihood,x = x, init=0)
# apply the permutations returned by typing:
reordered.mcmc<permute.mcmc(mcmc_pars,run$permutations)
# reordered.mcmc[,1]: reordered means of the two components
# reordered.mcmc[,2]: reordered variances
# reordered.mcmc[,3]: reordered weights

---

**Stephens’ algorithm**

**Description**

Stephens (2000) developed a relabelling algorithm that makes the permuted sample points to agree as much as possible on the $n \times K$ matrix of classification probabilities, using the Kullback-Leibler divergence. The algorithm’s input is the matrix of allocation probabilities for each MCMC iteration.
Usage

```
stephens(p)
```

Arguments

- `p`: \( m \times n \times K \) dimensional array of allocation probabilities of the \( n \) observations among the \( K \) mixture components, for each iteration \( t = 1, \ldots, m \) of the MCMC algorithm.

Details

For a given MCMC iteration \( t = 1, \ldots, m \), let \( w_k^{(t)} \) and \( \theta_k^{(t)} \), \( k = 1, \ldots, K \) denote the simulated mixture weights and component specific parameters respectively. Then, the \( (t, i, k) \) element of \( p \) corresponds to the conditional probability that observation \( i = 1, \ldots, n \) belongs to component \( k \) and is proportional to \( p_{ik} \propto w_k^{(t)} f(x_i | \theta_k^{(t)}) \), \( k = 1, \ldots, K \), where \( f(x_i | \theta_k) \) denotes the density of component \( k \). This means that:

\[
p_{ik} = \frac{w_k^{(t)} f(x_i | \theta_k^{(t)})}{\sum_{k=1}^{K} w_k^{(t)} f(x_i | \theta_k^{(t)})}.
\]

In case of hidden Markov models, the probabilities \( w_k \) should be replaced with the proper left (normalized) eigenvector of the state-transition matrix.

Value

- `permutations`: \( m \times K \) dimensional array of permutations
- `iterations`: integer denoting the number of iterations until convergence

Author(s)

Panagiotis Papastamoulis

References


See Also

- `permute.mcmc.label.switching`

Examples

```r
# load a toy example: MCMC output consists of the random beta model
# applied to a normal mixture of \( K=2 \) components. The number
# of observations is equal to \( n=5 \). The number of MCMC samples
# is equal to \( m=300 \). The matrix of allocation probabilities
# is stored to matrix \( p \).
data("mcmc_output")
data="mcmc_output")
# mcmc parameters are stored to array \( mcmc_pars \)
```


mcmc.pars<-data_list$mcmc.pars
# mcmc.pars[,1]: simulated means of the two components
# mcmc.pars[,2]: simulated variances
# mcmc.pars[,3]: simulated weights
# the computed allocation matrix is p
p<-data_list$p
run<-stephens(p)
# apply the permutations returned by typing:
reordered.mcmc<-permute.mcmc(mcmc.pars,run$permutations)
# reordered.mcmc[,1]: reordered means of the components
# reordered.mcmc[,2]: reordered variances
# reordered.mcmc[,3]: reordered weights
Index

*Topic \textasciitilde{}kw1
  compare.clust, 3
  label.switching, 9
  permute.mcmc, 13
  pra, 14
  sjw, 15
  stephens, 17

*Topic \textasciitilde{}kw2
  compare.clust, 3
  label.switching, 9
  permute.mcmc, 13
  pra, 14
  sjw, 15
  stephens, 17

*Topic datasets
  data_list, 4
  lamb, 12

*Topic package
  label.switching-package, 2

compare.clust, 3

data_list, 4

ecr, 5, 7, 9–11, 13
ecr.iterative.1, 5, 6, 9–11, 13
ecr.iterative.2, 5, 7, 8, 10, 11, 13

label.switching, 2–5, 7, 9, 13, 15, 16, 18
label.switching-package, 2
lamb, 12

permute.mcmc, 5, 7, 9, 13, 15, 16, 18
pra, 10, 11, 13, 14

sjw, 10, 11, 13, 15
stephens, 9–11, 13, 17