Package 'bayesImageS'

May 7, 2026

Type Package

Title Bayesian Methods for Image Segmentation using a Potts Model

Version 0.7-1

Date 2026-04-29

Encoding UTF-8

Description Various algorithms for segmentation of 2D and 3D images, such as computed tomography and satellite remote sensing. This package implements Bayesian image analysis using the hidden Potts model with external field prior of Moores et al. (2015) <doi:10.1016/j.csda.2014.12.001>. Latent labels are sampled using chequerboard updating or Swendsen-Wang. Algorithms for the smoothing parameter include pseudolikelihood, path sampling, the exchange algorithm, approximate Bayesian computation (ABC-MCMC and ABC-SMC), and the parametric functional approximate Bayesian (PFAB) algorithm. Refer to Moores, Pettitt & Mengersen (2020) <doi:10.1007/978-3-030-42553-1_6> for an overview and also to <doi:10.1007/s11222-014-9525-6> and <doi:10.1214/18-BA1130> for further details of specific algorithms.

License GPL (>= 2) | file LICENSE

URL https://bitbucket.org/Azeari/bayesimages, https://mooresm.github.io/bayesImageS/

BugReports https://bitbucket.org/Azeari/bayesimages/jira

LazyData true

Depends R (>= 3.5.0)

Imports Rcpp (>= 1.0.12)

LinkingTo Rcpp, RcppArmadillo

Suggests mcmcse, coda, PottsUtils, rstan, knitr, rmarkdown, lattice

VignetteBuilder knitr

RoxygenNote 7.3.3

NeedsCompilation yes

Author Matt Moores [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-4531-3572>),
Dai Feng [ctb],
Kerrie Mengersen [aut, ths] (ORCID:
<https://orcid.org/0000-0001-8625-9168>)

Maintainer Matt Moores <mmoores@gmail.com>

Repository CRAN

Date/Publication 2026-04-29 02:20:02 UTC

Contents

`exactPotts` *Calculate the distribution of the Potts model using a brute force algorithm.*

Description

Warning: this algorithm is O(k^n) and therefore will not scale for $k^n > 2^{31} - 1$

Usage

```
exactPotts(neighbors, blocks, k, beta)
```

Arguments

neighbors	A matrix of all neighbours in the lattice, one row per pixel.
blocks	A list of pixel indices, dividing the lattice into independent blocks.
k	The number of unique labels.
beta	The inverse temperature parameter of the Potts model.

Value

A list containing the following elements:

expectation The exact mean of the sufficient statistic.

variance The exact variance of the sufficient statistic.

exp_PL Pseudo-likelihood (PL) approximation of the expectation of S(z).

var_PL PL approx. of the variance of the sufficient statistic.

---

getBlocks *Get Blocks of a Graph*

Description

Obtain blocks of vertices of a 1D, 2D, or 3D graph, in order to use the conditional independence to speed up the simulation (chequerboard idea).

Usage

```
getBlocks(mask, nblock)
```

Arguments

mask	a vector, matrix, or 3D array specifying vertices of a graph. Vertices of value 1 are within the graph and 0 are not.
nblock	a scalar specifying the number of blocks. For a 2D graph nblock could be either 2 or 4, and for a 3D graph nblock could be either 2 or 8.

Details

The vertices within each block are mutually independent given the vertices in other blocks. Some blocks could be empty.

Value

A list with the number of components equal to nblock. Each component consists of vertices within the same block.

References

Wilkinson, D. J. (2005) "Parallel Bayesian Computation" *Handbook of Parallel Computing and Statistics*, pp. 481-512 *Marcel Dekker/CRC Press*

Examples

```
#Example 1: split a line into 2 blocks
getBlocks(mask=c(1,1,1,1,0,0,1,1,0), nblock=2)

#Example 2: split a 4*4 2D graph into 4 blocks in order
#           to use the chequerboard idea for a neighbourhood structure
#           corresponding to the second-order Markov random field.
getBlocks(mask=matrix(1, nrow=4, ncol=4), nblock=4)

#Example 3: split a 3*3*3 3D graph into 8 blocks
#           in order to use the chequerboard idea for a neighbourhood
#           structure based on the 18 neighbors definition, where the
#           neighbors of a vertex comprise its available
#           adjacencies sharing the same edges or faces.
mask <- array(1, dim=rep(3,3))
getBlocks(mask, nblock=8)
```

---

getEdges *Get Edges of a Graph*

Description

Obtain edges of a 1D, 2D, or 3D graph based on the neighbourhood structure.

Usage

```
getEdges(mask, neiStruc)
```

Arguments

mask	a vector, matrix, or 3D array specifying vertices of a graph. Vertices of value 1 are within the graph and 0 are not.
neiStruc	a scalar, vector of four components, or 3×4 matrix corresponding to 1D, 2D, or 3D graphs. It specifies the neighbourhood structure. See `getNeighbors` for details.

Details

There could be more than one way to define the same 3D neighbourhood structure for a graph (see Example 4 for illustration).

Value

A matrix of two columns with one edge per row. The edges connecting vertices and their corresponding first neighbours are listed first, and then those corresponding to the second neighbours, and so on and so forth. The order of neighbours is the same as in `getNeighbors`.

References

Winkler, G. (2003) "Image Analysis, Random Fields and Markov Chain Monte Carlo Methods: A Mathematical Introduction" (2nd ed.) *Springer-Verlag*

Feng, D. (2008) "Bayesian Hidden Markov Normal Mixture Models with Application to MRI Tissue Classification" *Ph. D. Dissertation, The University of Iowa*

Examples

```
#Example 1: get all edges of a 1D graph.
mask <- c(0,0,rep(1,4),0,1,1,0,0)
getEdges(mask, neiStruc=2)

#Example 2: get all edges of a 2D graph based on neighbourhood structure
#           corresponding to the first-order Markov random field.
mask <- matrix(1 ,nrow=2, ncol=3)
getEdges(mask, neiStruc=c(2,2,0,0))

#Example 3: get all edges of a 2D graph based on neighbourhood structure
#           corresponding to the second-order Markov random field.
mask <- matrix(1 ,nrow=3, ncol=3)
getEdges(mask, neiStruc=c(2,2,2,2))

#Example 4: get all edges of a 3D graph based on 6 neighbours structure
#           where the neighbours of a vertex comprise its available
#           N,S,E,W, upper and lower adjacencies. To achieve it, there
#           are several ways, including the two below.
mask <- array(1, dim=rep(3,3))
n61 <- matrix(c(2,2,0,0,
                0,2,0,0,
                0,0,0,0), nrow=3, byrow=TRUE)
n62 <- matrix(c(2,0,0,0,
                0,2,0,0,
                2,0,0,0), nrow=3, byrow=TRUE)
e1 <- getEdges(mask, neiStruc=n61)
e2 <- getEdges(mask, neiStruc=n62)
e1 <- e1[order(e1[,1], e1[,2]),]
e2 <- e2[order(e2[,1], e2[,2]),]
all(e1==e2)

#Example 5: get all edges of a 3D graph based on 18 neighbours structure
#           where the neighbours of a vertex comprise its available
#           adjacencies sharing the same edges or faces.
#           To achieve it, there are several ways, including the one below.

n18 <- matrix(c(2,2,2,2,
```

```
                0,2,2,2,
                0,0,2,2), nrow=3, byrow=TRUE)
mask <- array(1, dim=rep(3,3))
getEdges(mask, neiStruc=n18)
```

---

getNeighbors *Get Neighbours of All Vertices of a Graph*

Description

Obtain neighbours of vertices of a 1D, 2D, or 3D graph.

Usage

```
getNeighbors(mask, neiStruc)
```

Arguments

mask	a vector, matrix, or 3D array specifying vertices within a graph. Vertices of value 1 are within the graph and 0 are not.
neiStruc	a scalar, vector of four components, or 3×4 matrix corresponding to 1D, 2D, or 3D graphs. It gives the definition of neighbours of a graph. All components of neiStruc should be positive (≥ 0) even numbers. For 1D graphs, neiStruc gives the number of neighbours of each vertex. For 2D graphs, neiStruc[1] specifies the number of neighbours on vertical direction, neiStruc[2] horizontal direction, neiStruc[3] north-west (NW) to south-east (SE) diagonal direction, and neiStruc[4] south-west (SW) to north-east (NE) diagonal direction. For 3D graphs, the first row of neiStruc specifies the number of neighbours on vertical direction, horizontal direction and two diagonal directions from the 1-2 perspective, the second row the 1-3 perspective, and the third row the 2-3 perspective. The index to perspectives is represented with the leftmost subscript of the array being the smallest.

Details

There could be more than one way to define the same 3D neighbourhood structure for a graph (see Example 3 for illustration).

Value

A matrix with each row giving the neighbours of a vertex. The number of the rows is equal to the number of vertices within the graph and the number or columns is the number of neighbours of each vertex.

For a 1D graph, if each vertex has two neighbours, The first column are the neighbours on the left-hand side of corresponding vertices and the second column the right-hand side. For the vertices on boundaries, missing neighbours are represented by the number of vertices within a graph plus 1. When neiStruc is bigger than 2, The first two columns are the same as when neiStruc is equal to

2; the third column are the neighbours on the left-hand side of the vertices on the first column; the forth column are the neighbours on the right-hand side of the vertices on the second column, and so on and so forth. And again for the vertices on boundaries, their missing neighbours are represented by the number of vertices within a graph plus 1.

For a 2D graph, the index to vertices is column-wised. For each vertex, the order of neighbours are as follows. First are those on the vertical direction, second the horizontal direction, third the NW to SE diagonal direction, and forth the SW to NE diagonal direction. For each direction, the neighbours of every vertex are arranged in the same way as in a 1D graph.

For a 3D graph, the index to vertices is that the leftmost subscript of the array moves the fastest. For each vertex, the neighbours from the 1-2 perspective appear first and then the 1-3 perspective and finally the 2-3 perspective. For each perspective, the neighbours are arranged in the same way as in a 2D graph.

References

Winkler, G. (2003) "Image Analysis, Random Fields and Markov Chain Monte Carlo Methods: A Mathematical Introduction" (2nd ed.) *Springer-Verlag*

Feng, D. (2008) "Bayesian Hidden Markov Normal Mixture Models with Application to MRI Tissue Classification" *Ph. D. Dissertation, The University of Iowa*

Examples

```
#Example 1: get all neighbours of a 1D graph.
mask <- c(0,0,rep(1,4),0,1,1,0,0,1,1,1)
getNeighbors(mask, neiStruc=2)

#Example 2: get all neighbours of a 2D graph based on neighbourhood structure
#           corresponding to the second-order Markov random field.
mask <- matrix(1, nrow=2, ncol=3)
getNeighbors(mask, neiStruc=c(2,2,2,2))

#Example 3: get all neighbours of a 3D graph based on 6 neighbours structure
#           where the neighbours of a vertex comprise its available
#           N,S,E,W, upper and lower adjacencies. To achieve it, there
#           are several ways, including the two below.
mask <- array(1, dim=rep(3,3))
n61 <- matrix(c(2,2,0,0,
                0,2,0,0,
                0,0,0,0), nrow=3, byrow=TRUE)
n62 <- matrix(c(2,0,0,0,
                0,2,0,0,
                2,0,0,0), nrow=3, byrow=TRUE)
n1 <- getNeighbors(mask, neiStruc=n61)
n2 <- getNeighbors(mask, neiStruc=n62)
n1 <- apply(n1, 1, sort)
n2 <- apply(n2, 1, sort)
all(n1==n2)

#Example 4: get all neighbours of a 3D graph based on 18 neighbours structure
#           where the neighbours of a vertex comprise its available
```

```
#           adjacencies sharing the same edges or faces.
#           To achieve it, there are several ways, including the one below.

n18 <- matrix(c(2,2,2,2,
                0,2,2,2,
                0,0,2,2), nrow=3, byrow=TRUE)
mask <- array(1, dim=rep(3,3))
getNeighbors(mask, neiStruc=n18)
```

---

gibbsGMM *Fit a mixture of Gaussians to the observed data.*

Description

Fit a mixture of Gaussians to the observed data.

Usage

```
gibbsGMM(y, niter = 1000, nburn = 500, priors = NULL)
```

Arguments

y	A vector of observed pixel data.
niter	The number of iterations of the algorithm to perform.
nburn	The number of iterations to discard as burn-in.
priors	A list of priors for the parameters of the model.

Value

A matrix containing MCMC samples for the parameters of the mixture model.

---

gibbsNorm *Fit a univariate normal (Gaussian) distribution to the observed data.*

Description

Fit a univariate normal (Gaussian) distribution to the observed data.

Usage

```
gibbsNorm(y, niter = 1000, priors = NULL)
```

Arguments

y	A vector of observed pixel data.
niter	The number of iterations of the algorithm to perform.
priors	A list of priors for the parameters of the model.

Value

A list containing MCMC samples for the mean and standard deviation.

Examples

```
y <- rnorm(100,mean=5,sd=2)
res.norm <- gibbsNorm(y, priors=list(mu=0, mu.sd=1e6, sigma=1e-3, sigma.nu=1e-3))
summary(res.norm$mu[501:1000])
summary(res.norm$sigma[501:1000])
```

`gibbsPotts` *Fit a hidden Potts model to the observed data, using a fixed value of beta.*

Description

Fit a hidden Potts model to the observed data, using a fixed value of beta.

Usage

```
gibbsPotts(y, labels, beta, mu, sd, neighbors, blocks, priors, niter = 1)
```

Arguments

`y`	A vector of observed pixel data.
`labels`	A matrix of pixel labels.
`beta`	The inverse temperature parameter of the Potts model.
`mu`	A vector of means for the mixture components.
`sd`	A vector of standard deviations for the mixture components.
`neighbors`	A matrix of all neighbors in the lattice, one row per pixel.
`blocks`	A list of pixel indices, dividing the lattice into independent blocks.
`priors`	A list of priors for the parameters of the model.
`niter`	The number of iterations of the algorithm to perform.

Value

A matrix containing MCMC samples for the parameters of the Potts model.

`initSedki` *Initialize the ABC algorithm using the method of Sedki et al. (2013)*

Description

Initialize the ABC algorithm using the method of Sedki et al. (2013)

Usage

```
initSedki(y, neighbors, blocks, param = list(npart = 10000), priors = NULL)
```

Arguments

`y`	A vector of observed pixel data.
`neighbors`	A matrix of all neighbours in the lattice, one row per pixel.
`blocks`	A list of pixel indices, dividing the lattice into independent blocks.
`param`	A list of options for the ABC-SMC algorithm.
`priors`	A list of priors for the parameters of the model.

Value

A matrix containing SMC samples for the parameters of the Potts model.

References

Sedki, M.; Pudlo, P.; Marin, J.-M.; Robert, C. P. & Cornuet, J.-M. (2013) "Efficient learning in ABC algorithms" arXiv:1210.1388

`mcmcPotts` *Fit the hidden Potts model using a Markov chain Monte Carlo algorithm.*

Description

Fit the hidden Potts model using a Markov chain Monte Carlo algorithm.

Usage

```
mcmcPotts(
  y,
  neighbors,
  blocks,
  priors,
  mh,
  niter = 55000,
  nburn = 5000,
  truth = NULL
)
```

Arguments

y	A vector of observed pixel data.
neighbors	A matrix of all neighbors in the lattice, one row per pixel.
blocks	A list of pixel indices, dividing the lattice into independent blocks.
priors	A list of priors for the parameters of the model.
mh	A list of options for the Metropolis-Hastings algorithm.
niter	The number of iterations of the algorithm to perform.
nburn	The number of iterations to discard as burn-in.
truth	A matrix containing the ground truth for the pixel labels.

Value

A matrix containing MCMC samples for the parameters of the Potts model.

---

mcmcPottsNoData *Simulate pixel labels using chequerboard Gibbs sampling.*

Description

Simulate pixel labels using chequerboard Gibbs sampling.

Usage

```
mcmcPottsNoData(beta, k, neighbors, blocks, niter = 1000, random = TRUE)
```

Arguments

beta	The inverse temperature parameter of the Potts model.
k	The number of unique labels.
neighbors	A matrix of all neighbors in the lattice, one row per pixel.
blocks	A list of pixel indices, dividing the lattice into independent blocks.
niter	The number of iterations of the algorithm to perform.
random	Whether to initialize the labels using random or deterministic starting values.

Value

A list containing the following elements:

`alloc` An n by k matrix containing the number of times that pixel i was allocated to label j.

`z` An `(n+1)` by k matrix containing the final sample from the Potts model after niter iterations of chequerboard Gibbs.

`sum` An `niter` by 1 matrix containing the sum of like neighbors, i.e. the sufficient statistic of the Potts model, at each iteration.

Examples

```
# Swendsen-Wang for a 2x2 lattice
neigh <- matrix(c(5,2,5,3,  1,5,5,4,  5,4,1,5,  3,5,2,5), nrow=4, ncol=4, byrow=TRUE)
blocks <- list(c(1,4), c(2,3))
res.Gibbs <- mcmcPottsNoData(0.7, 3, neigh, blocks, niter=200)
res.Gibbs$z
res.Gibbs$sum[200]
```

`res` *Simulation from the Potts model using single-site Gibbs updates.*

Description

100 iterations of Gibbs sampling for a 500×500 lattice with $\beta = 0.22$ and $k = 2$.

Usage

```
res
```

Format

A `list` containing 7 variables.

See Also

`mcmcPotts`

res2 *Simulation from the Potts model using single-site Gibbs updates.*

Description

100 iterations of Gibbs sampling for a 500×500 lattice with $\beta = 0.44$ and $k = 2$.

Usage

```
res2
```

Format

A `list` containing 7 variables.

See Also

`mcmcPotts`

res3 *Simulation from the Potts model using single-site Gibbs updates.*

Description

100 iterations of Gibbs sampling for a 500×500 lattice with $\beta = 0.88$ and $k = 2$.

Usage

```
res3
```

Format

A `list` containing 7 variables.

See Also

`mcmcPotts`

`res4` *Simulation from the Potts model using single-site Gibbs updates.*

Description

100 iterations of Gibbs sampling for a 500×500 lattice with $\beta = 1.32$ and $k = 2$.

Usage

```
res4
```

Format

A `list` containing 7 variables.

See Also

`mcmcPotts`

`res5` *Simulation from the Potts model using single-site Gibbs updates.*

Description

5000 iterations of Gibbs sampling for a 500×500 lattice with $\beta = 1.32$ and $k = 2$.

Usage

```
res5
```

Format

A `list` containing 4 variables.

See Also

`mcmcPottsNoData`

`smcPotts` *Fit the hidden Potts model using approximate Bayesian computation with sequential Monte Carlo (ABC-SMC).*

Description

Fit the hidden Potts model using approximate Bayesian computation with sequential Monte Carlo (ABC-SMC).

Usage

```
smcPotts(
  y,
  neighbors,
  blocks,
  param = list(npart = 10000, nstat = 50),
  priors = NULL
)
```

Arguments

`y`	A vector of observed pixel data.
`neighbors`	A matrix of all neighbors in the lattice, one row per pixel.
`blocks`	A list of pixel indices, dividing the lattice into independent blocks.
`param`	A list of options for the ABC-SMC algorithm.
`priors`	A list of priors for the parameters of the model.

Value

A matrix containing SMC samples for the parameters of the Potts model.

`sufficientStat` *Calculate the sufficient statistic of the Potts model for the given labels.*

Description

Calculate the sufficient statistic of the Potts model for the given labels.

Usage

```
sufficientStat(labels, neighbors, blocks, k)
```

Arguments

labels	A matrix of pixel labels.
neighbors	A matrix of all neighbors in the lattice, one row per pixel.
blocks	A list of pixel indices, dividing the lattice into independent blocks.
k	The number of unique labels.

Value

The sum of like neighbors.

---

swNoData *Simulate pixel labels using the Swendsen-Wang algorithm.*

Description

The algorithm of Swendsen & Wang (1987) forms clusters of neighbouring pixels, then updates all of the labels within a cluster to the same value. When simulating from the prior, such as a Potts model without an external field, this algorithm is very efficient.

Usage

```
swNoData(beta, k, neighbors, blocks, niter = 1000, random = TRUE)
```

Arguments

beta	The inverse temperature parameter of the Potts model.
k	The number of unique labels.
neighbors	A matrix of all neighbors in the lattice, one row per pixel.
blocks	A list of pixel indices, dividing the lattice into independent blocks.
niter	The number of iterations of the algorithm to perform.
random	Whether to initialize the labels using random or deterministic starting values.

Value

A list containing the following elements:

`alloc` An n by k matrix containing the number of times that pixel i was allocated to label j.

`z` An `(n+1)` by k matrix containing the final sample from the Potts model after niter iterations of Swendsen-Wang.

`sum` An `niter` by 1 matrix containing the sum of like neighbors, i.e. the sufficient statistic of the Potts model, at each iteration.

References

Swendsen, R. H. & Wang, J.-S. (1987) "Nonuniversal critical dynamics in Monte Carlo simulations" *Physical Review Letters* **58**(2), 86–88, DOI: doi:10.1103/PhysRevLett.58.86

Examples

```
# Swendsen-Wang for a 2x2 lattice
neigh <- matrix(c(5,2,5,3,  1,5,5,4,  5,4,1,5,  3,5,2,5), nrow=4, ncol=4, byrow=TRUE)
blocks <- list(c(1,4), c(2,3))
res.sw <- swNoData(0.7, 3, neigh, blocks, niter=200)
res.sw$z
res.sw$sum[200]
```

`synth` *Simulation from the Potts model using Swendsen-Wang.*

Description

Simulations for a 500×500 lattice for fixed values of the inverse temperature parameter, β.

Usage

```
synth
```

Format

A `list` containing 5 variables:

0.22 simulations for $\beta = 0.22$

0.44 simulations for $\beta = 0.44$

0.88 simulations for $\beta = 0.88$

1.32 simulations for $\beta = 1.32$

tm time taken by the simulations

See Also

`swNoData`

`testResample` *Test the residual resampling algorithm.*

Description

Test the residual resampling algorithm.

Usage

```
testResample(values, weights, pseudo)
```

Arguments

values	A vector of SMC particles.
weights	A vector of importance weights for each particle.
pseudo	A matrix of pseudo-data for each particle.

Value

A list containing the following elements:

beta A vector of resampled particles.

wt The new importance weights, after resampling.

pseudo A matrix of pseudo-data for each particle.

idx The indices of the parents of the resampled particles.

References

Liu, J. S. & Chen, R. (1998) "Sequential Monte Carlo Methods for Dynamic Systems" *J. Am. Stat. Assoc.* **93**(443): 1032–1044, DOI: doi:10.1080/01621459.1998.10473765

Index