Package ‘potts’

Version 0.5-7
Date 2017-03-16
Title Markov Chain Monte Carlo for Potts Models
Author Charles J. Geyer <charlie@stat.umn.edu> and Leif Johnson <ltjohnson@google.com>
Maintainer Charles J. Geyer <charlie@stat.umn.edu>
Depends R (>= 3.0.2)
Imports stats, graphics, compiler
Suggests pooh (>= 0.2)
URL http://www.stat.umn.edu/geyer/mcmc/
NeedsCompilation yes
Repository CRAN
Date/Publication 2017-03-16 22:58:12 UTC

R topics documented:

  Cache T ................................................................. 2
  calc_t .............................................................. 4
  composite.ll .................................................... 6
  image.raw ........................................................ 8
  packPotts ......................................................... 8
  potts .............................................................. 9
Cache calculated Canonical Statistics for Potts Models.

Description

Variety of functions to support caching of calculated canonical statistics for Potts Models. There is some attempt at being 'smart' with when to regenerate the statistics.

Usage

```r
generate_t_cache(x, ncolor, t_stat, sizeA, npixel, f,
    fapply=lapply, gridcache=NULL)

gengridcache(ncolor, sizeCA, ncol)

gensingleton(ncolor)

singleton(x, ncolor, a, idx, gridcache=NULL)

gentwopixel(ncolor)

twopixel(x, ncolor, a, idx, gridcache=NULL)

twopixel.nonoverlap(x, ncolor, a, idx, gridcache=NULL)

genfourpixel(ncolor)

fourpixel(x, ncolor, a, idx, gridcache=NULL)

fourpixel.nonoverlap(x, ncolor, a, idx, gridcache=NULL)

genthreebythree(ncolor)

ninepixel.nonoverlap(x, ncolor, a, idx, gridcache=NULL)

genfourbyfour(ncolor)

sixteenpixel.nonoverlap(x, ncolor, a, idx, gridcache=NULL)
```

Arguments

- **t_stat**: numerical vector of length `ncolor`. Contains the canonical statistic for the whole image.
- **sizeA**: numerical. The number of elements in $\mathcal{A}$.
- **sizeCA**: numerical. The number of elements in $C^A$. 
Cache $T$

**rpixel**

Numerical. The number of pixels in one element of $A$.

**f**

Function. Takes arguments $x$, ncolor, a, idx and ncolor. Returns value of $t_{\text{calc}_\text{innergrid}}$ with window $A_a$ replaced by the $idx^{th}$ element of $C^A$.

**fapply**

Function. It should behave exactly as `lapply` does. You can use this argument to enable parallel computing.

**gridcache**

List. Optional. If non-null, it is a list of the elements of $C^A$.

**x**

Numeric, 2 dimensional matrix, elements in 1, ..., ncolor, representing a Potts model.

**ncolor**

Numeric. Number of colors in this Potts Model.

**ncol**

Numeric. Gives the number of columns in a rectangular window.

**a**

Numeric. Indicates which member of $A$ is being referenced.

**idx**

Numeric. Indicates which element of $C^A$ is being referenced.

**Details**

For a description of notation and terminology, see `composite.ll`.

This set of functions is used to generate cached calculations of the canonical statistic of a Potts model suitable for passing into `composite.ll` or `gr.composite.ll`.

All of the calculations using `composite.ll` and these caching functions need one of the color components to be dropped for the model to be identifiable. For simplicity, the first color is dropped by `generate_t_cache`. In computing the composite log likelihood for a Potts model with ncolor colors, we are interested in many calculations across $C^A$, the set of all permutations of colors across a window. These functions facilitate those calculations. `gridcache` is a list of $C^A$.

`generate_t_cache` is the main function, and the others are intended to be used in conjunction with it. `generate_t_cache` creates a list of arrays. Each array represents one window in the image, and each row of the array contains the value of $t(x)$ (with one component dropped) found by replacing the pixels in that window with one of the elements of $C^A$.

`gengridcache` can generate the gridcache for any rectangular window, give the number of colors, size of $C^A$, and number of columns in the window. `gensingleton`, `gentwopixel`, `genfourpixel`, `genthreebythree` and `genfourbyfour` are all just simple wrappers for `gengridcache`.

`singleton`, `twopixel`, `twopixel.nonoverlap`, `fourpixel`, `fourpixel.nonoverlap`, `ninepixel.nonoverlap` and `sixteenpixel.nonoverlap` are intended to be passed to `generate_t_cache` in the argument $f$. They are used to calculate $t(ca_{idx} \cup X \setminus A_a)$ for the $idx^{th}$ element of $C^A_a$.

Functions that have `overlap` and `nonoverlap` versions generate a overlapping and nonoverlapping set of windows respectively.

`singleton` is for a single pixel window (Besag or MLE).

`twopixel` does a two horizontal pixel window.

`fourpixel` does a two by two pixel window.

`ninepixel` does a three by three pixel window.

`sixteenpixel` does a four by four pixel window.
Value

Functions that start with gen return a list of the elements of $C^A$.
The other functions (e.g. twopixel, fourpixel,...) return the result of replacing the $a$-th window
of $x$ with the $idx$-th element of $C^A$ and calculating $\text{calc}_t_{\text{innergrid}}$ for that window.

See Also

composite, calc_t.

Examples

```r
ncolor <- 4
beta <- log(1+sqrt(ncolor))
theta <- c(rep(0, ncolor), beta)
nrow <- 32
ncol <- 32
x <- matrix(sample(ncolor, nrow*ncol, replace=TRUE), nrow=nrow, ncol=ncol)
foo <- packPotts(x, ncolor)
out <- potts(foo, theta, nbatch=10)
x <- unpackPotts(out$final)
t_stat <- calc_t(x, ncolor)
t_cache_mple <- generate_t_cache(x, ncolor, t_stat, nrow*ncol, 1, singleton)
```

## Not run:
# use multicore to speed things up.
library(multicore)
t_cache_mple <- generate_t_cache(x, ncolor, t_stat, nrow*ncol, 1, singleton, fapply=mclapply)

## End(Not run)

---

calc_t

Calculate Canonical Statistic for Potts Model

Description

Calculate the canonical statistic ‘t’ for a realization of a Potts Model

Usage

```r
calc_t_full(x, ncolor)
calc_t_innergrid(x, ncolor, grid, i, j)
calc_t(x, ncolor, grid==NULL, i=NULL, j=NULL)
```
calc_t

Arguments

- **x**: 2 dimensional matrix, elements in 1,...,ncolor, representing a Potts model
- **ncolor**: numeric. Number of colors in this Potts Model.
- **grid**: numeric. 2 dimensional matrix, elements in 1,...,ncolor. If non-NULL it is placed into x at the location x[i,j].
- **i**: numeric. Row to place the grid.
- **j**: numeric. Column to place the grid.

Details

For a description of notation and terminology, see composite.ll.

Calculates the canonical statistics for a realized Potts Model. calc_t calls calc_t_full if grid is NULL and calc_t_innergrid otherwise.

- **calc_t_full**: calculates the canonical statistics for the full image.
- **calc_t_innergrid**: calculates the canonical statistics for the a window of the image, but with that window replaced by grid, with the upper left corner of grid located at x[i,j].

Value

For a description of notation and terminology, see composite.ll.

All functions return a vector of length ncolor+1. Elements 1,...,ncolor contain the number of pixels of each color. Element ncolor+1 contains the number of matching neighbor pairs for the image.

- **calc_t_full** returns the values for the whole image.
- **calc_t_innergrid** returns the value for just the selected window, but this includes the number of matching pairs from the replaced window to it’s neighbors. E.g. if X is the full image, and Aa is the value of some window in the image and we want to know the value of t(y∪X\Aa) this would be calc_t_full(X, ncolor) + calc_t_innergrid(X, ncolor, y, i, j) - calc_t_innergrid(X, ncolor, A(a), i, j).

See Also

generate_t_cache, composite.ll.

Examples

```r
ncolor <- 4
beta <- log(1+sqrt(ncolor))
theta <- c(rep(PLncolor), beta)
nrow <- 32
ncol <- 32
x <- matrix(sample(ncolor, nrow*ncol, replace=TRUE), nrow=nrow, ncol=ncol)
foo <- packPotts(x, ncolor)
out <- potts(foo, theta, nbatch=10)
x <- unpackPotts(out$final)
```
t_stat <- calc_t(x, ncolor)
t_stat_inner <- calc_t(x, ncolor, matrix(1, nrow=2, ncol=2), 1, 1)

---

**composite.ll**  
*Composite Log Likelihood for Potts Models*

**Description**

Calculate Composite Log Likelihood (CLL) and the gradient of the CLL for Potts models.

**Usage**

```r
composite.ll(theta, t_stat, t_cache=NULL, fapply=lapply)
gr.composite.ll(theta, t_stat, t_cache=NULL, fapply=lapply)
```

**Arguments**

- `theta` numeric canonical parameter vector. The CLL will be evaluated at this point. It is assumed that the component corresponding to the first color has been dropped.
- `t_stat` numeric, canonical statistic vector. The value of the canonical statistic for the full image.
- `t_cache` list of arrays. `t_cache[[i]][j]` = the value of `t` with window `A_i` replaced by the `j`th element of `C_A`.
- `fapply` function. Expected to function as `lapply` does. Useful for enabling parallel processing. E.g. use the `mclapply` function from the `multicore` package.

**Details**

For the given value of `theta` `composite.ll` and `gr.composite.ll` calculate the CLL and the gradient of the CLL respectively for a realized Potts model represented by `t_stat` and `t_cache`.  

`A` is the set of all windows to be used in calculating the Composite Log Likelihood (CLL) for a Potts model. A *window* is a collection of adjacent pixels on the lattice of the Potts model. `A` is used to represent a generic window in `A`, the code in this package expects that all the windows in `A` have the same size and shape. `|A|` is used to denote the size, or number of pixels in a window. Each pixel in a Potts takes on a value in `C`, the set of possible colors. For simplicity, this implementation takes `C = \{1, \ldots, ncolor\}`. Elements of `C` will be referenced using `c_j` with `j \in \{1, \ldots, ncolor\}`. `C^A` is used to denote all the permutations of `C` across the window `A`, and `|C|^{|A|}` is used to denote the size of `C^A`. In an abuse of notation, we use `A_a` to refer to the `a^{th}` element of `A`. No ordinal or numerical properties of `A`, `C` or `C^A` are used, only that each element in the sets are referenced by one and only one indexing value.

**Value**

- `composite.ll` returns CLL evaluated at `theta`.
- `gr.composite.ll` returns a numeric vector of length `length(theta)` containing the gradient of the CLL at `theta`. 

---

```r
t_stat <- calc_t(x, ncolor)
t_stat_inner <- calc_t(x, ncolor, matrix(1, nrow=2, ncol=2), 1, 1)
```
See Also

generate_t_cache, calc_t.

Examples

ncolor <- 4
beta <- log(1+sqrt(ncolor))
theta <- c(rep(0, ncolor), beta)

nrow <- 32
ncol <- 32
x <- matrix(sample(ncolor, nrow*ncol, replace=TRUE), nrow=nrow, ncol=ncol)
foo <- packPotts(x, ncolor)
out <- potts(foo, theta, nbatch=10)
x <- unpackPotts(out$final)
t_stat <- calc_t(x, ncolor)
t_cache_mple <- generate_t_cache(x, ncolor, t_stat, nrow*ncol, 1, singleton)
t_cache_two <- generate_t_cache(x, ncolor, t_stat, nrow*ncol/2, 2, twopixel.nonoverlap)

composite.ll(theta[-1], t_stat, t_cache_mple)
gr.composite.ll(theta[-1], t_stat, t_cache_mple)

## Not run:
optim.mple <- optim(theta.initial, composite.ll, gr=gr.composite.ll,
t_stat, t_cache_mple, method="BFGS",
control=list(fnscale=-1))
optim.mple$par

optim.two <- optim(theta.initial, composite.ll, gr=gr.composite.ll,
t_stat, t_cache_two, method="BFGS",
control=list(fnscale=-1))
optim.two$par

## Not run:
# or use mclapply to speed things up.
library(multicore)
optim.two <- optim(theta.initial, composite.ll, gr=gr.composite.ll,
t_stat, t_cache_two, mclapply, method="BFGS",
control=list(fnscale=-1))
optim.two$par

## Not run
Plot Potts Model Data

Description

plot Potts model data.

Usage

```r
## S3 method for class 'raw'
image(x, col = c("white", "red", "blue", "green",
               "black", "cyan", "yellow", "magenta"), ...)
```

Arguments

- `x`: an R vector of class "raw" that encodes a realization of a Potts model, typically the output of `packPotts` or of `potts`.
- `col`: a vector of colors. Must be as many as number of colors of Potts model.
- `...`: other arguments passed to `image.default`.

Bugs

Too slow for large images. Needs to be rewritten for efficient plotting.

See Also

- `potts`

Transform Potts Model Data

Description

transform Potts model data from integer matrix to raw vector and vice versa.

Usage

```r
packPotts(x, ncolor)
inspectPotts(raw)
unpackPotts(raw)
```

Arguments

- `x`: integer matrix containing Potts model data. Colors are coded from one to `ncolor`.
- `ncolor`: integer scalar, number of colors.
- `raw`: vector of type "raw".
Value

for packPotts a vector of type "raw". for inspectPotts a list containing components ncolor, nrow, and ncol. for unpackPotts an integer matrix.

Examples

x <- matrix(sample(4, 2 * 3, replace = TRUE), nrow = 2)
x
foo <- packPotts(x, ncolor = 4)
foo
inspectPotts(foo)
unpackPotts(foo)

potts

Potts Models

Description

Simulate Potts model using Swendsen-Wang algorithm.

Usage

potts(obj, param, nbatch, blen = 1, nspac = 1,
      boundary = c("torus", "free", "condition"), debug = FALSE,
      outfun = NULL, ...)

Arguments

obj

an R vector of class "raw" that encodes a realization of a Potts model, typically the output of packPotts. Alternatively, an object of class "potts" from a previous invocation of this function can be supplied, in which case any missing arguments are taken from this object.

param

numeric, canonical parameter vector. Last component must nonnegative (see Details below).

nbatch

the number of batches.

blen

the length of batches.

nspac

the spacing of iterations that contribute to batches.

boundary

type of boundary conditions. The value of this argument can be abbreviated.

default

return additional debugging information.

outfun

controls the output. If a function, then the batch means of outfun(tt, ...) are returned. The argument tt is the canonical statistic of the Potts model having the same length as the argument param of this function. If NULL, the batch means of the canonical statistic are returned.

... additional arguments for outfun.
Details

Runs a Swendsen-Wang algorithm producing a Markov chain with equilibrium distribution having the specified Potts model. The state of a Potts model is a collection of random variables taking values in a finite set. Here the finite set is 1,..., ncolor and the elements are called "colors". The random variables are associated with the nodes of a rectangular lattice, represented by *unpackpotts* as a matrix. In keeping with calling the values "colors", the random variables themselves are often called "pixels". The probability model is an exponential family with canonical statistic vector of length $ncolor + 1$. The first $ncolor$ components are the counts of the number of pixels of each color. The last component is the number of pairs of neighboring pixels colored the same. The corresponding canonical parameter, last component of the canonical parameter vector (argument *param*) must be nonnegative for the Swendsen-Wang algorithm to work (Potts models are defined for negative dependence parameter, but can't be simulated using this algorithm).

In the default boundary specification ("torus"), also called toroidal or periodic boundary conditions, the vertical edges of the pixel matrix are considered glued together, as are the horizontal edges. Thus corresponding pixels in the first and last rows are considered neighbors, as are corresponding pixels in the first and last columns. In the other boundary specifications there is no such gluing: pixels in the the relative interiors of the first and last rows and first and last columns have only three neighbors, and the four corner pixels have only two neighbors.

In the "torus" and "free" boundary specifications, all pixels are counted in determining the color count canonical statistics, which thus range from zero to $nrow * ncol$, where $nrow$ and $ncol$ are the number of rows and columns of the pixel matrix. In the "condition" boundary specification, all pixels in the first and last rows and first and last columns are fixed (conditioned on), and only the random pixels are counted in determining the color count canonical statistics, which thus range from zero to $(nrow - 2) * (ncol - 2)$.

In the "torus" boundary specification, all pixels have four neighbors, so the neighbor pair canonical statistic ranges from zero to $2 * nrow * ncol$. In the "free" boundary specification, pixels in the interior have four neighbors, those in the relative interior of edges have three, and those in the corners have two, so the neighbor pair canonical statistic ranges from zero to $nrow * (ncol - 1) + (nrow - 1) * ncol$. In the "condition" boundary specification, only neighbor pairs in which at least one pixel is random are counted, so the neighbor pair canonical statistic ranges from zero to $(nrow - 2) * (ncol - 1) + (nrow - 1) * (ncol - 2)$.

Value

an object of class "potts", which is a list containing at least the following components:

- **initial**: initial state of Markov chain in the format output by *packPotts*.
- **final**: final state of Markov chain in the same format.
- **initial.seed** value of .Random.seed before the run.
- **final.seed** value of .Random.seed after the run.
- **time** running time of Markov chain from *system.time*.
- **param** canonical parameter vector.
- **nbatch** the number of batches.
- **blen** the length of batches.
- **nspac** the spacing of iterations that contribute to batches.
- **boundary** the argument boundary.
potts

batch  an nbatch by nout matrix, where nout is the length of the result returned by outfun or length(param) if outfun == NULL; each row is the batch means for the result of outfun or the canonical statistic vector for one batch of Markov chain iterations.

Examples

ncolor <- as.integer(4)
beta <- log(1 + sqrt(ncolor))
theta <- c(rep(0, ncolor), beta)
nrow <- 100
ncol <- 100
x <- matrix(1, nrow = nrow, ncol = ncol)
foo <- packPotts(x, ncolor)

out <- potts(foo, theta, nbatch = 10)
out$batch
## Not run: image(out$final)
Index

*Topic misc
- Cache T, 2
- calc_t, 4
- composite.ll, 6
- image.raw, 8
- packPotts, 8
- potts, 9

Cache T, 2
calc_t, 4, 4, 7
calc_t_full(calc_t), 4
calc_t_innergrid(calc_t), 4
composite.ll, 3–5, 6

fourpixel(Cache T), 2
generate_t_cache, 5, 7
generate_t_cache(Cache T), 2
genfourbyfour(Cache T), 2
genfourpixel(Cache T), 2
gengridcache(Cache T), 2
gensingleton(Cache T), 2
genthreebythree(Cache T), 2
genwopixel(Cache T), 2
grcomposite.ll(composite.ll), 6

image.raw, 8
inspectPotts(packPotts), 8

ninepixel.nonoverlap(Cache T), 2
packPotts, 8, 8, 9, 10
potts, 8, 9

singleton(Cache T), 2
sixteenpixel.nonoverlap(Cache T), 2
system.time, 10

twopixel(Cache T), 2
unpackPotts, 10
unpackPotts(packPotts), 8